Key Activities for a Development Methodology of Interactive Applications

François Bodart[*], *Anne-Marie Hennebert*[*], *Jean-Marie Leheureux*[*],
Isabelle Provot[*], *Jean Vanderdonckt*[*], *Giovanni Zucchinetti*[**]

(*) Facultés Universitaires Notre-Dame de la Paix, Institut d'Informatique,
rue Grandgagnage, 21, B-5000 NAMUR (Belgium)
Tel: +32 (0)81-72.49.75 - Fax: +32 (0)81-72.49.67 - Telex: 59.222 FacNamB
Email: {fbodart, amhennebert, jmleheureux, jvanderdonckt}@ info.fundp.ac.be

(**) Université de Lausanne, Ecole des Hautes Etudes Commerciales,
CH-1015 LAUSANNE (Switzerland)
Tel: +41 21-692.34.17 - Fax: +41 21-692.34.05 - Email: gzucchin@hec.unil.ch

Abstract
Traditional system development methodologies have many shortcomings for highly interactive business applications; development phases can be poorly integrated and lack continuity; design is not grounded in task analysis. This paper presents five key activities that span several life cycle phases to address existing shortcomings. The activities form the methodological framework from which a specification framework can be derived. The methodological framework supports selection of design options, creating a design space for a specific class of problems. It lets user interface design be coupled to software engineering in a framework that supports effective decisions and actions. The five key activities support two transformations: the first from a methodological framework to a specification framework; the second from a specification framework to an implementation of an interactive application.

Keywords
Automatic generation, development methodology, dialogue, framework, guideline, interactive application, presentation, task analysis, user interface design.

1. Introduction
User Interface (UI) development tools (e.g., toolboxes, user interface management systems), design guides and guidelines provide essential support for interactive systems development, but they are not enough in isolation, since no fruitful outcome of an interactive application development process can be guaranteed. Ideally, development methodologies for interactive business applications should combine searching analyses of business processes, with design models and appropriate development tools that impose structure on development and guarantee high quality solutions, in both their technical and ergonomic aspects. This ideal may be utopian. A more feasible and profitable goal is for a methodology's phases to satisfy some minimal properties. Such properties require key activities to be integrated into development methodologies for interactive systems. Each activity can ensure coherent integration of the outputs of different phases, preserving these through subsequent phases without loss of information, yet entailing no further work. Each activity supports the convergence of development phases at a satisfactory solution.

2. A methodological framework
The methodological framework developed on the TRIDENT project had to be informal. Task analysis, model specification and tool use are all overlapped within the framework of a methodology that supports a wide range of design situations and also provides a design space.

The aim is to provide a general framework that covers various interaction styles (e.g., menu selection, form filling, command language, direct manipulation, multi-windowing) and dialogue structures (e.g., synchronous, asynchronous, multi-threaded dialogues).

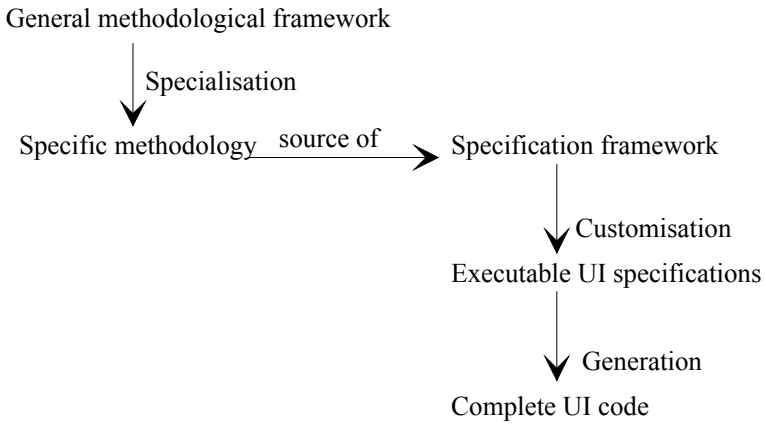

Fig. 1. Structure of TRIDENT methodology.

This general framework can be specialised to produce a specific methodological framework. The specialisation produces a design space that is specific to a given class of problems. This design space replaces the full range of different groups of user interface design options with the narrower choice of one group of option (fig. 1). The specific methodology can also be used to generate a specification framework. This in turn can be customised to produce executable UI specifications allowing to generate the code of the interactive application (fig. 1).

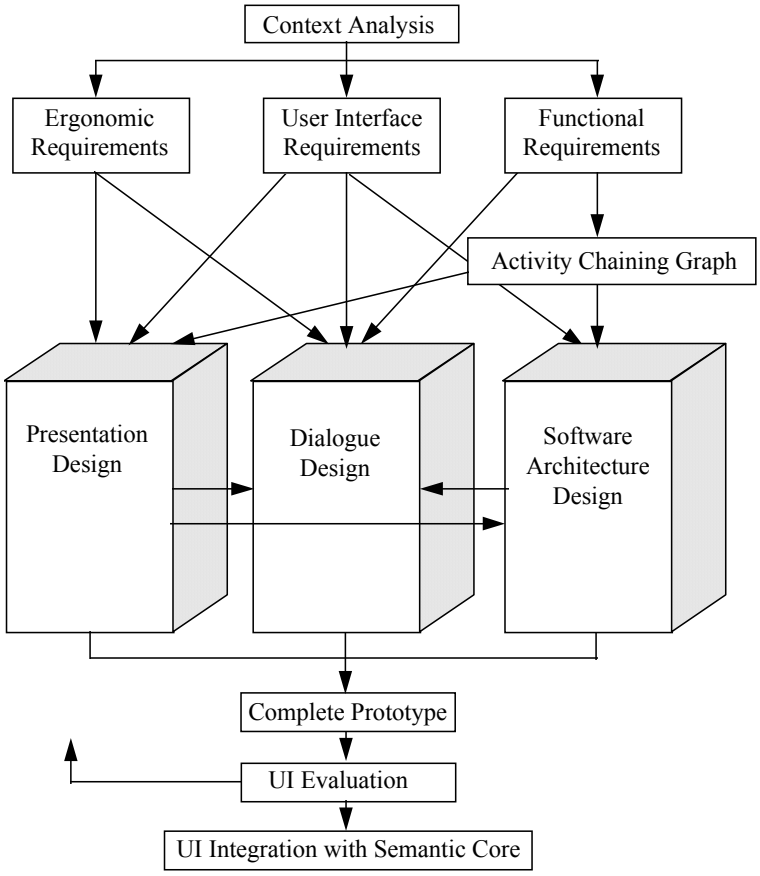

Fig. 2. Structure of TRIDENT methodological framework.

The structure of TRIDENT methodological framework is due to five key activities. Each is described in the following sections :

1. forming UI specifications from the output of task analysis (section 3);
2. guiding presentation design using ergonomic rules (section 4);
3. deriving a software architecture from task analysis and presentation components (section 5);
4. forming high level dialogue specification from the output of task analysis (section 6);
5. transforming the methodological framework into a specification framework (section 7);

Figure 2 outlines the main parts of TRIDENT methodology. We will show throughout this paper how suggested key activities are set along this general structure.

3. Activity 1: forming user interface specifications from the output of task analysis

Starting from an extensive context analysis (task analysis, user stereotypes and workplace description) [17], our goal is to define a systematic approach for specifying UI components from this contextual information (fig. 3). Such a systematic approach should ensure preservation of information between development phases, and thus we express contextual information using concepts and formalisms that are appropriate for UI designers.

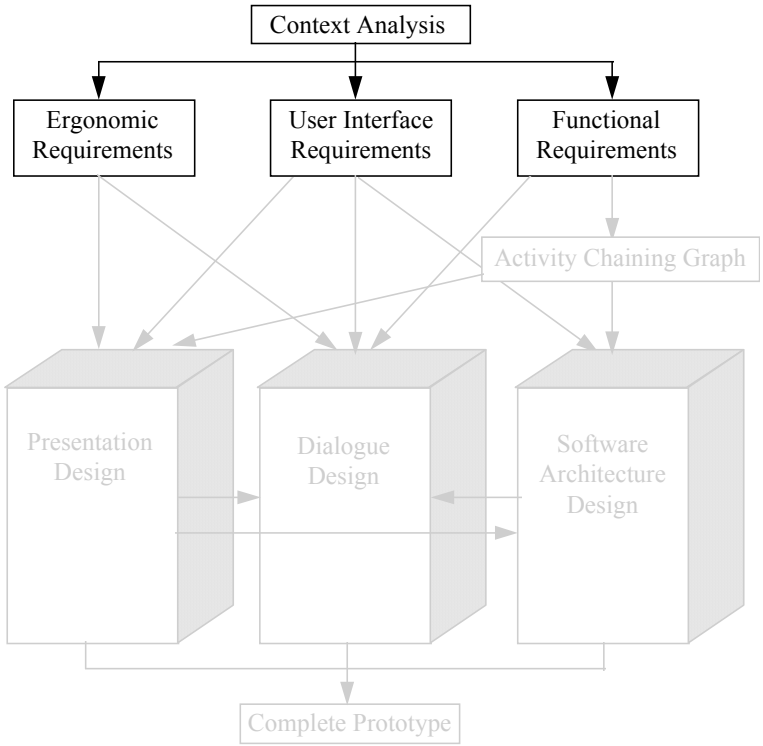

Fig. 3. Outline of Activity 1.

3.1 Task analysis

A *task* is defined as any activity carried out by an operator (often named user) that results into a significant state change in a given activity domain in a given contextual situation. Hierarchical task analysis [28] could be appropriate for highly-structured tasks, but event-driven analysis [14] or constraint-based approaches [13] may be more effective for weakly structured tasks, whether they are simple or complex.

Though we will limit the scope of this paper to hierarchical task analysis (we currently use TKS [16] method for this type of task), weakly structured tasks will be addressed in future work since such tasks dominate interaction with decision support systems. Task analysis will therefore be expanded to event-driven analysis and constraint-based approach in order to compare both their outputs and the UI components that are derived from these outputs.

In hierarchical task analysis, the significant state change of domain activity is related to a main goal. A *goal* is a particular state of the domain to be fulfilled. A goal explains why a particular task should be carried out. Any goal could be decomposed recursively into smaller units called *sub-goals*. Each sub-goal consists of any intermediary state of the main goal through which the user should pass. This approach includes three steps :

1. *identifying goals and sub-goals*: this identification is achieved by asking specific questions, by examining documents produced by the organisation, by interview, by basic protocol,...

   This step produces a hierarchical decomposition of the task into sub-goals.

2. *identifying procedures*: a task is decomposed into *procedures*, too. A *procedure* establishes an observable and executable behaviour which is the combination of actions on objects resulting from a particular state of the domain activity. Some procedures are qualified according to the level of their constituting actions: *sub-task*, *intermediary*, or *action*. A set of identification criteria is to be considered for this purpose. The action level is the last level in the hierarchy, leading to only one action. The sub-task level is a privileged level in the hierarchy since it satisfies special properties resulting from the action combination. Other levels (between task and sub-task, between sub-task and action) are the intermediary levels ; they do not have any special status.

   This identification is achieved by asking specific questions, by filling questionnaires, by card sorting, by observation, by basic protocol,... [28]

   This step produces a hierarchical decomposition of an interactive task into procedures, a definition of inter-procedure relationships (e.g., sequence, iteration, parallelism, concurrence) with the links between procedures and sub-goals.

3. *identifying objects and actions related to the task*: each action handles one or several *objects*. An object designates any abstract or concrete entity belonging to the domain activity, which is related to one or many actions within one or many procedures. Each object possesses a predefined set of properties depending on the activity domain.

   This step produces a list of task objects linked with their relationships and their actions.

3.2 Expressing the product of task analysis

3.2.1 Writing an object-oriented entity-relationship model

Task analysis not only produces task objects and actions (as in TKS [16]), but also identifies relationships between objects (e.g., is-a, is-member-of, has-a, is-used-by, is-a-group) and relationships between actions (e.g., sequential, parallel, deterministic or non-deterministic alternative, interleaved, repetition, disable). Objects produced by task analysis and their relationships are further specified in a schema based on an object-oriented entity-relationship

model [27]. It describes not only simple entities and semantic n-ary relationships with attributes, but also complex entities defined as aggregates of entities and/or semantic relationships between entities. Similarly, any attribute from any entity or relationship can itself form a new entity. This schema is supposed to fill the gap between the cognitive universe of the task analyst and the conceptual world of the software engineer. This schema can be obtained as follows :

- entities and attributes come from the list of task objects with their relationships;
- relationships come from object relationships: they include inheritance, aggregation, and semantic relationships of the activity domain.

3.2.2 Identification of semantic functions of the application

Task analysis produces actions that deal with task objects. These task concepts should be transformed into concepts that are relevant to software engineering (i.e., functions that deal with entities/relationships attributes. For this purpose, actions identified during task analysis are modified in accordance with software properties (e.g., precision, completeness, consistency, redundancy, generalisation) in order to get functional specifications of the application.

This critical process transforms actions into functions by abstraction mechanism based on both generalisation and consolidation (fig. 4). For example, actions that are performed on objects possessing common slots can become a more general function dealing with two entities. These functions will become methods attached to the application objects according to object-oriented programming.

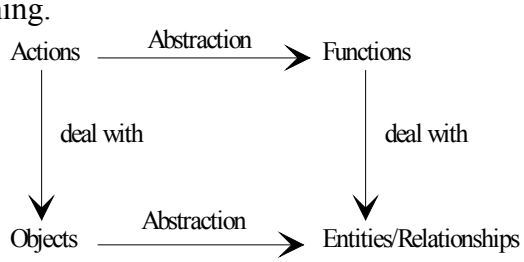

Fig. 4. The abstraction mechanism.

3.2.3 Constructing an Activity Chaining Graph

After having described static task aspects, dynamic aspects remain to be further detailed. This task behaviour model can be graphically represented with an *activity chaining graph* (ACG). This ACG institutes a contract between the programmer who is responsible for the semantic functions of the application and the designer who is responsible of the UI [23,24,25,26].

It expresses the information flow between functions to be executed for achieving the main goal associated with an interactive task. From the graph theory viewpoint, this graph is a 1-graph without loops, that is simple.

The ACG is exemplified in fig. 5. Each function receives input information, representing data required for the good execution of the function. Each function receives input information, produces output information which are either external (to the user) or internal (to another function). External information input to a function must come from an interactive dialogue. External information output from a function must go to an interactive display. Input and output information can be related using OR (no arc of a circle), AND (simple arc) or XOR (double arc) links .

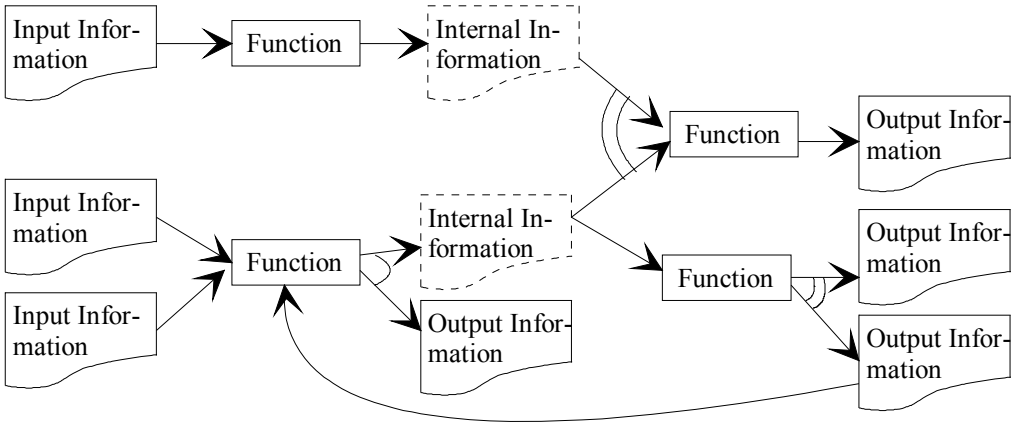

Fig. 5. Example of Activity Chaining Graph.

An ACG is constructed as follows:

- the elementary input and output information is determined for each identified function: this information corresponds to attributes either of entities or of associated relationships; input and output information also respectively support specification of function pre- and post-conditions.
- each inter-procedure relationship is classified as AND, OR, or XOR;

We intend to extend ACGs to include task execution information such as decision points, synchronisation points, error messages, indications and dialogue information (e.g., prompts, acknowledgements, progress indicators).

3.2.4 Derivation of dialogue attributes

For each interactive task, an appropriate (hybrid) dialogue style should be selected. This choice could be guided by a table that mapped task, user and workplace characteristics to a (hybrid) dialogue style.

(Hybrid) dialogue styles are examples of design options and their principled selection is the first example of restricted choice within our methodological framework.

For each interactive task, four dialogue attributes are derived from these interaction styles: a dialogue mode (sequential, asynchronous, or mixed), a dialogue control (internal, external, or mixed), a function triggering mode (automatic, implicit or explicit manual, displayed or not) and a metaphor (conversation based, universe based or both). These four attributes are introducing four new design options to be considered. They usually have their own default value according to the chosen interaction styles.

Default dialogue attributes provide initial choices of options. They can be reviewed for specific subtasks, although the attributes for really apply at task level, where they can provide UI consistency and compatibility with the ergonomic needs of both the user and their tasks.

4. Activity 2: guiding presentation design using ergonomic rules

Single threaded interaction with alphanumeric VDUs can not match the potential usability of multi-threaded interaction with current bitmapped displays. However, the use of these displays alone will not deliver high quality interfaces that users want to be able do their work.

This is why all display layouts must be considered by the judicious use of ergonomic rules (or guidelines).
The ergonomic rules that guide the design of display layouts must not be restricted to considerations of the physics of human perception. However, the many advantages of such guidelines are dependent on intelligent interpretation of cognitive psychology principles [19], in order to adjust them for specific task contexts and design situations.

To address this shortcoming, a pre-defined set of domain relevant ergonomic rules should be extracted: here, the area of business applications is considered.

The goal of this activity is to find a systematic approach for specifying UI presentation components on a pre-defined set of ergonomic rules in order

- to meet ergonomic criteria that are relevant to the task;
- to drive this process with computer-aided active and intelligent tools (fig. 6).

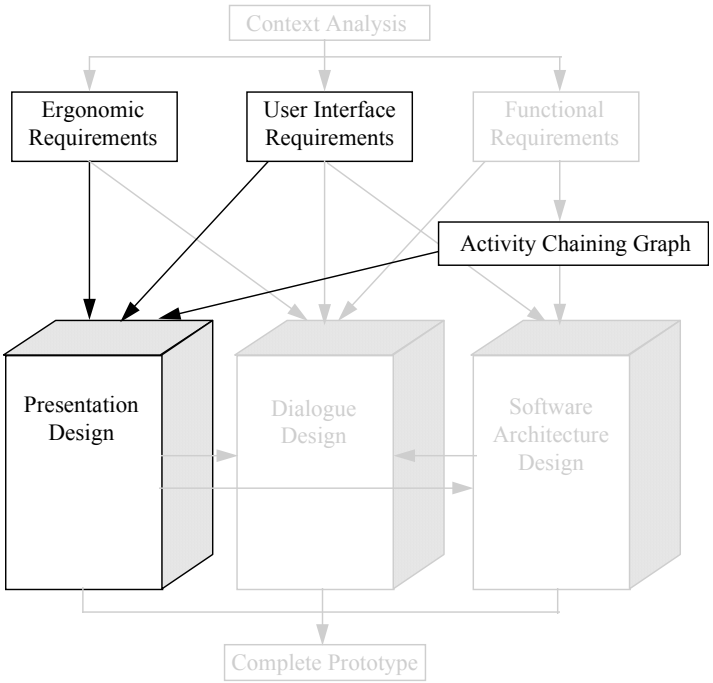

Fig. 6. Outline of Activity 2.

4.1. Presentation content
UI presentation can be decomposed using the following four concepts:

- *concrete interaction object* (CIO): this is a real object belonging to the UI world that any user can manipulate such as a push button, a list box, a check box. A CIO is simple if it cannot be decomposed into smaller CIOs. A CIO is composite if it can de decomposed into smaller units;
- *abstract interaction object* (AIO): this consists of an abstraction of all CIOs from both presentation and behavioural viewpoints that are independent of target environments;
- *window*: this is a root window either considered as a logical window for AIOs or corresponding to a physical window, a dialogue box or a panel for CIOs. Every window is itself a composite AIO at logical level or CIO at physical level, composed of other simple or composite AIOs/CIOs. All windows are geographically delimited on the user's screen;

- *presentation unit* (PU): this comprises of an input/output facilities required for execution of specific sub-tasks. Each presentation unit can be decomposed into one or many windows which may not be all displayed on the screen simultaneously. Each PU is composed by at least one window called the *basic window* from which other windows are chained.

4.2. Systematic approach for specifying presentation

The approach here defines a presentation by iterative refinement starting from global objects to end with simple objects according to the structure reproduced in fig. 7.

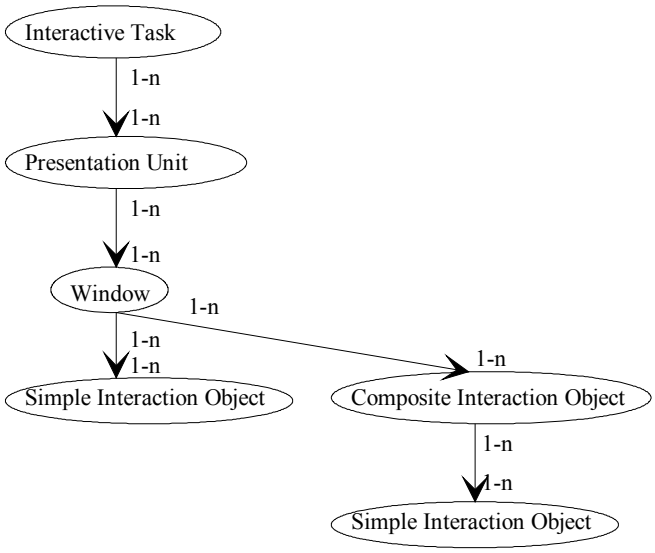

Fig. 7. Structure of presentation.

The steps of this approach are the following :

- *Identification of PUs*: each sub-task of the interactive task is mapped onto a PU. This mapping is achieved by a series of sub-task identification criteria (for instance, a different work skill). Since the ACG graphically represents the function chaining within a particular task, a PU can correspond to a sub-graph of the ACG that can be drawn graphically distinguished on the ACG (fig. 8).

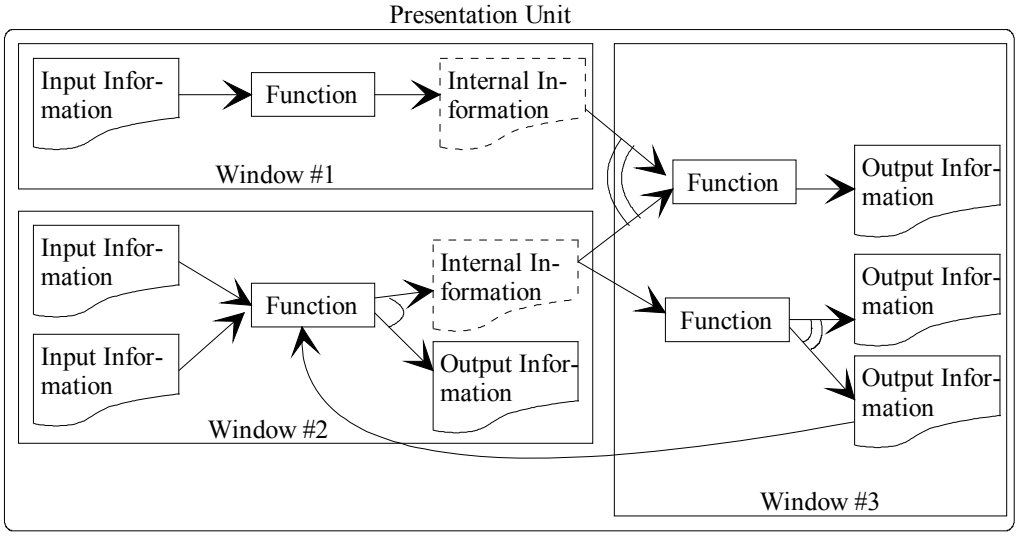

Fig. 8. Overlapping of a PU with its three windows onto the ACG.

- *Identification of windows*: for each sub-graph associated with a PU, partitioning into sub-sub-graphs identifies windows of the PU. Correct identification of these windows requires the application of precise ergonomic rules to ensure satisfaction of several ergonomic criteria (e.g., compatibility, work load, guidance) [33]. With this done, designers then know which functions will be triggered from the window, and thus which inputs and outputs must be associated with the window. Like PUs, windows correspond to sub-graphs of the ACG and can again be graphically distinguished (fig. 8).

- *Selection of AIOs*: each identified window can be associated with a physical window, a dialogue box or a panel ; within each window, each piece of input and output information is mapped to a simple or composite AIO (e.g., a text widget), as are functions (e.g., to push buttons or icons). Elementary information is always mapped onto a simple AIO. Composite information is mapped onto a composite AIO resulting in a hierarchy of simple AIOs. Similarly, a PU is completely defined by a hierarchy of windows.

  Selection of AIOs is supported by a set of selection rules with a scope exceeding the simple physical emphasis of style guides [29]. The rules are initially based on empirically validated cognitive principles. They are subsequently specialised in accordance with user's habits and established conventions.

  Within the scope of TRIDENT project methodology, the process of selecting AIOs consists of:

  - an initial proposed set of AIOs is generated automatically from the specification contained in the object-oriented entity-relationship model [32]: for example, an edit box is selected for two digit positive integer;
  - a second step extends the initially proposed AIOs on the basis of information attributes: for example, a scale is preferred when this integer is a value bounded in a specific range;
  - a third step modifies the extended AIOs on the basis of user and domain preference (e.g., a thermometer is useful in medicine).

  This selection process is supported by an expert system containing about 300 rules [7]. The expert system is:

  - interactive because one or many AIOs are presented to the user within the scope of the three steps mentioned above;
  - visual because the different alternatives are visually displayed on the screen to develop the developer's understanding;
  - able to explain choices with full reference to benefits/shortcomings, enabling the designer to refine the decision by enlarging his/her own knowledge.

- *Transformation of AIOs into CIOs*: the complete PU hierarchy is automatically transformed into a hierarchy of CIOs depending on the particular target environment in which the designer is working. Simple and composite AIOs are mapped respectively to simple and composite CIOs.

- *Placement of CIOs*: the co-ordinates of CIOs are calculated precisely for each window by applying placement strategies [5]. These strategies include visual design principles for

three aspects: localisation, scaling, and arrangement. At the end of this step, each PU hierarchy is completed with relevant positions.
Within the scope of the TRIDENT project methodology, the process of placing CIOs is computer-aided by two placement strategies :

1. a static strategy that places CIOs according to a predefined layout grid [5]; this strategy allows a completely automatic generation of placement within each window;
2. a dynamic strategy that places CIOs according to heuristics [5] that let visual designers control the generated layout of each window.

Using ergonomic rules is believed to encourage presentation aesthetics, but does not lead inevitably to a perfect result under every circumstances. We claim that only a multiple-strategy approach is useful in the long-term. Other placement strategies, whether they are automatic or computer-aided, can also be investigated.

- *Editing presentation*: once all CIOs have been placed into composite CIOs, they can be read into a graphical presentation editor. Direct manipulation of CIOs lets designer tailor the presentation to user's needs, in part by attending aspects ignored during the above selection and placement.

5. Activity 3: Deriving a software architecture from task analysis and presentation components

Many architectural models have been proposed for interactive systems, but none are derived systematically from a task analysis. However, some studies have some task elements in it [20,22].

The aim of this activity is to systematically derive an architecture skeleton which respects architectural software quality criteria: high internal cohesion; weak coupling; independent components (fig. 9).

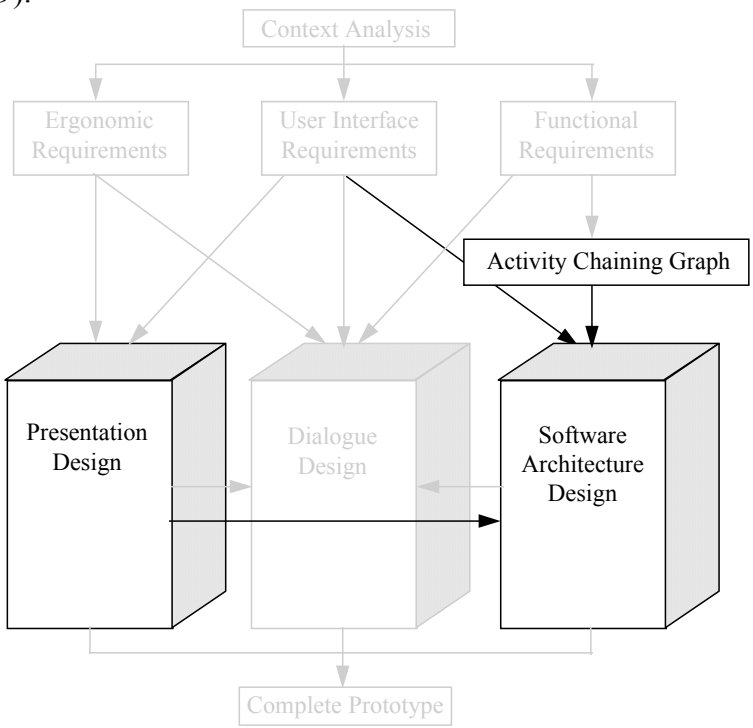

Fig. 9. Outline of Activity 3.

5.1. Assumptions

The proposed architecture model consists of a hierarchy that should match the following methodological assumptions :

- each hierarchy element should be derived - directly or indirectly - from task analysis;
- elements representing application and UI components should be as independent as possible. There should be further independence within UI components, i.e., between dialogue subcomponents (which realise behaviour) and presentation subcomponents (which realise appearance) [2,25].

The latter assumptions are possible for business oriented applications, since generally there is no semantic role for the interaction objects.

5.2. Content of the model

The proposed architecture model [3,4] consists of a hierarchy of generic elements illustrated in fig. 10.

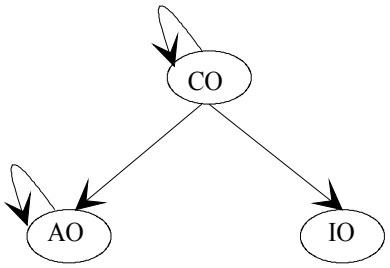

Fig. 10. Generic scheme of the architecture model.

There are three classes of object in this architectural model. Each is distinguished, as follows :

1. The *Control Objects* (CO) class is generic with instances decomposed into COs of different types that both manage dialogue and preserve the correspondence between application data and the presentation. Each CO has a specific behaviour that combines management of a portion of the dialogue and some application-presentation correspondences. A rule-based language [3,4] configures CO behaviour in *scripts* that have a partial graphical representation as state transition diagrams.
2. The *Application Objects* (AO) class is not generic: instances cannot be decomposed, since they represent the application functions.
3. The *Interaction Objects* (IO) class is generic and provides two types of CIOs: application-dependent CIOs that translate input and output information for functions; and application-independent CIOs that are required for a dialogue (e.g., command buttons that trigger functions).

Identical rules for behaviour and interaction apply to all these three object classes. Similarly, identical relationships link any pair of these from these three objects: each object is an *agent* [1,3].

Objects are composed in a hierarchy where parent objects "use" child objects, as follows:

- child objects send events related to signal behaviour states to their parent;
- parent objects obtain the information needed for the next interaction step by calling a child object's primitive methods.

5.2.1. Application Objects (AO)

Each function in the ACG has a single corresponding AO. Where an existing function is to be re-used, but is not implemented in an object-oriented language, it must be encapsulated in a AO, when it becomes one of the AO's methods. Otherwise, the AO should be implemented in an object-oriented language. Every AO should be the child of one (and only one) CO (i.e., the CO-Fc as we will see in next subsection), which becomes responsible for the execution of the corresponding function.

5.2.2. Control Objects (CO)

There should be a CO in the hierarchy for every composition in the presentation. The composition of the presentation encourages autonomy for windows, which are linked dynamically into PUs. These PUs realise the context for execution of an interactive task. The CO hierarchy is thus built according to our presentation structure (fig. 7), and thus four types of CO result [15]:

- CO-IT: the lonely control object corresponding to the interactive task;
- CO-PU: the control objects corresponding to the presentation units;
- CO-W: the control objects corresponding to the windows;
- CO-Fc: the control objects corresponding to the application functions.

The resulting structure is shown in figure 11.

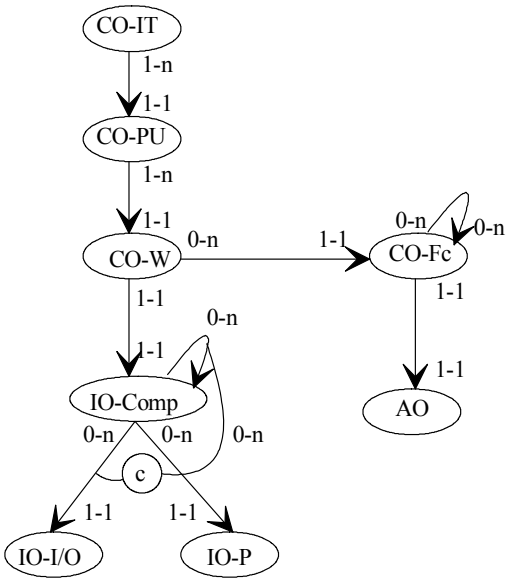

Fig. 11. Structure of Control Objects.

Every CO corresponding to an application function (CO-Fc) is a child object of the window control object (CO-W) that holds all CIOs required to trigger the function from the functional machine. The set of AOs and corresponding CO-Fc's forms the *functional machine* [23], where Co-Fc's mirror the business activities represented in the ACG by the way they chain functions.

5.2.3. Interaction Objects (IO)

According to fig. 7, presentation is structured into simple and composite IOs. *Composite IOs* (IO-Comp) are all IOs corresponding to logical windows (e.g., dialogue boxes, physical windows, panels) or any grouping of simple IOs (e.g., child dialogue boxes, group boxes).

IO-Comp use simple IOs for creating the UI's presentation: as *input-output interaction objects* (IO-I/O, e.g., an edit box, radio button, check box); or as *presentation interaction objects* (P-IO, e.g., push buttons, icons). IOs are selected by an expert system on the basis of several parameters. They correspond to specific toolkit CIOs in physical environments like Ms-Windows, OSF/Motif. IOs are always children of CO-W control objects, to which they send events representing significant state changes. Primitive IO methods pass relevant values to parent objects. These primitive methods tend to be native to specific toolkits.

5.2.4. Inter-object relations

- In CO-IT→CO-PU interactions, the dynamic chaining of PUs is managed by the CO-IT, which loads and unloads - sequentially or concurrently - a particular PU_i according to events received from a PU_j.

- In CO-PU→CO-W interactions, the dynamic chaining of windows is managed by the CO-PU, which displays and undisplays - with tiling or overlapping - a particular window W_i according to events received from another window W_j.

- In CO-W→CO-Fc interactions, the CO-W calls the CO-Fc in order to call a semantic function, when all data required by the function to be performed are available in the child objects of the CO-W. The CO-Fc is the control object which is responsible for the final call of function contained in AO. On completion of processing, CO-Fc's send an event to their CO-W.

- In CO-Fc→AO interactions, the CO-Fc are responsible for the final triggering of functions contained in AOs. A CO-W object calls a function when all required information is input, but the final triggering will be effective only if all triggering conditions expressed in the ACG are fulfilled, that is if all termination events have been sent and internal information has been transmitted. The corresponding CO-Fc only verifies the complete pre-condition before executing the function.

- In recursive CO-Fc interactions, CO-Fc's exchange events to maintain ACG dynamics and to exchange information by calling on each others services.

- In IO (for IO-I/O or IO-P) and CO-W interactions, the parent object polls children to find out what users have input, and in turn send events to their parents that indicate their contents.

5.3. Systematic approach to hierarchy building

We assume that task analysis (subsection 3.1), constructing an ACG (sub-subsection 3.2.3), the development of the functional machine (sub-subsection 5.2.1) and the definition of presentation (sub-subsection 5.2.3) are all completed [15].

5.3.1. List of hierarchy objects

Here, we summarize the complete list of objects of the three kinds. It holds:

- a control object corresponding to the interactive task (CO-IT);
- a control object corresponding to each presentation unit ($CO\text{-}PU_1$,...,$CO\text{-}PU_n$);
- a control object corresponding to each window of each presentation unit ($CO\text{-}PU_1F_1$,..., $CO\text{-}PU_1F_m$,...,$CO\text{-}PU_nF_1$,...$CO\text{-}PU_nF_p$);
- a control object for each function in the ACG (CO-Fc);

- an interaction object for each input/output information for all functions (IO-I/O);
- an interaction object for each object induced by the presentation (IO-P).

5.3.2. Relationships between hierarchy objects

"Uses" relationships linking the different objects can be systematically created as follows:

- connect any control object corresponding to a presentation unit to the control object corresponding to the interactive task from which it depends (CO-IT→CO-PU);
- connect any control object corresponding to a window to the control object corresponding to the presentation unit which contains this window (CO-PU→CO-W);
- connect any interaction object corresponding to an input/output information to the window that uses the function with this information as parameter (CO-W→IO-I/O);
- connect any interaction object corresponding to an information induced by the presentation to the window where it appears (CO-W→IO-P);
- connect any object control corresponding to an application function to the control object corresponding to the window in which this function is represented (CO-W→CO-Fc).

6. Activity 4: forming high level dialogue specifications from the output of task analysis

Normally, each dialogue layer should be specified in a notation that task analysts can understand, but can also be executed by some software tools. Multiple formalisms could be avoided by using ACG constructs. Most current approaches do not use common specifications: task analysts tend to use natural language descriptions for dialogue or task-related notations; construction tools that execute (or generate) dialogues use formal languages (e.g., Petri nets, rule languages [3], attributed grammars), although they are unfamiliar to task analysts.

In this section, we will investigate some perspectives for modelling the dialogue in order to fill this important gap (fig. 13).

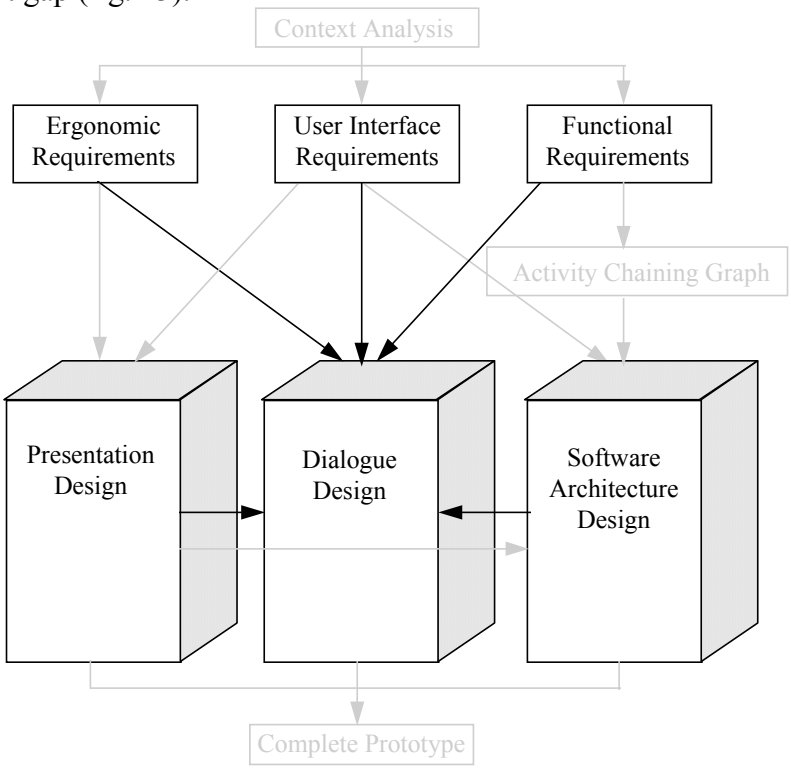

Fig. 12. Outline of Activity 4.

We retain the need for independence between the dialogue and the application functions. Such independence is easy to achieve for the presentation components, since control objects guarantee correspondence between these and the application functions [11]. Control objects thus can guarantee the separation of the presentation and application objects [11,12]. However, separation between dialogue is more critical: the functional logic of the ACG should not govern the progress of the dialogue, but the dialogue should be compatible with it, and clearly the ACG cannot be achieved.

6.1. Dialogue content

Dialogue is related to four types of element (i.e., PU, window, CIO, and function) and can be described at three levels of abstraction (fig. 13):

1. *inter-PU level*: the triggering of sub-tasks represented by PUs at the interactive task level;
2. *intra-PU level* (or inter-window level): the chaining of windows at the PU level;
3. *intra-window level*: the behavioural dependencies among CIOs in the same window.

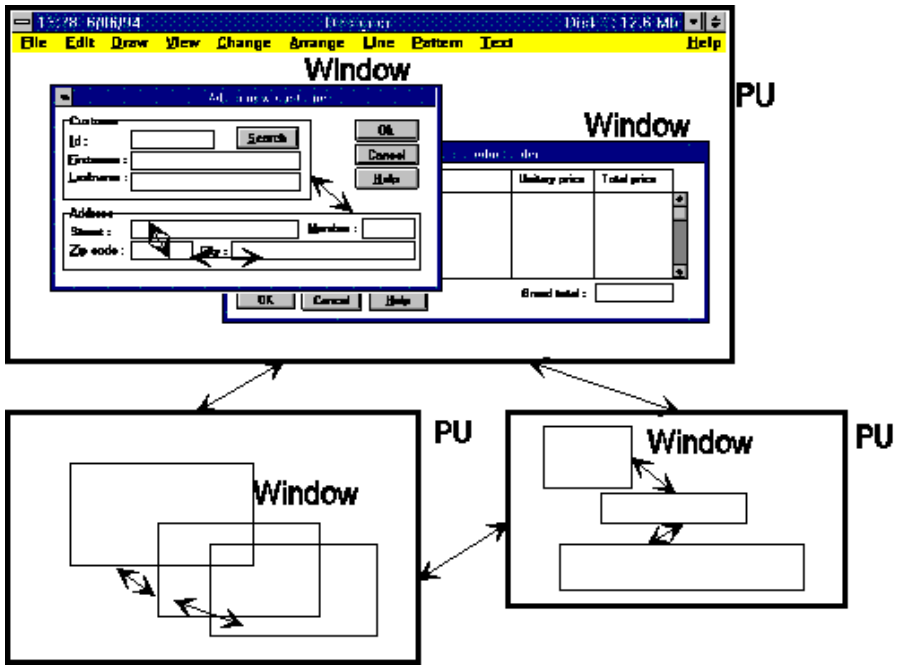

Fig. 13. The dialogue levels.

CIOs linked by a bi-directional link (arrow) in figure 13 become involved in the same dialogue at the inter-window level; similarly, windows linked by a bi-directional arrow are involved in intra-PU dialogue, just as PUs involved at the inter-PU level are linked by such arrows.

6.2. Systematic approach for specifying a dialogue

Initially, the approach should be driven by task analysis:

- a list of dialogue states (e.g., default, display, history, object, initial, final, related/alternative) can be derived from the decomposition of a task into goals and sub-goals;
- a list of dialogue transitions (e.g., triggers, actions, constraints) can be derived from the relationships between procedures implied by associated (sub-)task goals.

The approach iteratively refines dialogues, starting with high-level dialogues and ending with low-level ones. The following steps will achieve this refinement:

- *specifying inter-PU dialogues*, by using *presentation unit chaining graphs* that specify the conditions under which PUs are initialised, activated, and terminated [26]. This graph should be expressed in terms of activation/deactivation of PU, serialisation/parallelism, foreground/background, suspend/restore,...
  The graph can be derived from the PU's definition and from the decomposition of the interactive task into subtasks. Further support comes from dialogue ergonomic rules and from constructs for sequence and concurrency.

- *specifying intra-PU dialogues*, by using *window chaining graphs* that specify window manipulation operations. This graph should be expressed in terms of multi-windowing operations since multi-windowing is widely recognised as an effective support for multiple activities [19]: creation/deletion, activation/deactivation, minimising/maximising, iconifying/restoring,... Also, one can decide the window configuration and screen layout: partial or total tiling, partial or total overlapping, cascading.

  Specifying this dialogue level should be driven by cognitive psychology principles related to multiple activities [19] and ergonomic rules on multi-windowing [29]. A tool that would be able to suggest the designer appropriate window chaining would be a significant contribution if this tool could intrinsically take care of ergonomic criteria (e.g., work load) [8].

- *specifying intra-window dialogues*, by using interaction object graphs that should be expressed in terms of the many primitives provided by CIOs in a window: (un)display, (de)activation, (de)highlighting, inter-CIO influences, dynamic or function update,...

  Derivation of these dialogues should be driven by the four dialogue attributes for PUs, by interaction styles and by the functional logic underlying the ACG. Initially, the derivation only produces a skeleton of the interaction graph, leaving the designer with many further decisions on syntactic and lexical level details. Decisions can be organised by formalisms such as the state transition diagram [34] and the interaction object graph [10]. UAN [26] can also specify this detail.

One problem is the diversity of formalisms used across the different levels of dialogue specification.

7. Activity 5: Transforming the methodological framework into a specification framework

7.1. A Specification Framework based on the TRIDENT Methodology

General assumptions in the TRIDENT methodology can be satisfied by the detail of concrete specification frameworks. Once these frameworks are specialised for a specific problem, they can support automatic UI generation. In their general form, they can underpin industrial approaches to UI development that are based on current technologies (e.g., bitmap asynchronous screens, pull-down menus, multi-windowing techniques); these approaches should be similar to previous prevailing approaches (e.g., for text screens).

Several approaches can be envisaged for defining specification frameworks. They could be domain-specific: bank, insurance, hospital,... 4GL environments, objects librairies (e.g. Microsoft Foundations Classes) could also be considered as implementation environments. A

specification framework could be based on restricted UI functions: interaction style, dialogue mode, choose of interaction objects,...

This third alternative is illustrated by an extension of MacIDA project developed at University of Lausanne and lead by I. Petoud and Y. Pigneur [23,24,25]. MacIDA's aim is to allow a quasi automatic derivation of a UI from a task's entity-relationship model, from attached functions and their activity chaining graph (cfr. supra 3.2.1, 3.2.2 and 3.2.3). In terms of the general methodological framework presented above, the framework based on this third approach involves three restrictions which are described below.

7.1.1. Supported Applications

This framework supports highly interactive business oriented applications that enable the user to follow multiple dialogue threads, in contrast to previous applications that were purely transactional and did not allow any user freedom.

7.1.2. Interaction style

MacIDA supports forms filling for graphical UI. Therefore, control is basically external (i.e., user-driven), although mixed input information is also possible. The dialogue mode is asynchronous, letting users enter data in any convenient order, entering or modifying input information at any time. Functions are triggered by explicit user interaction with display objects. A menu bar is displayed in the upper part of the screen letting users select general items (e.g., quit the application, cut/paste information). A pull-down menu lets users navigate between the different windows containing the information.

7.1.3. Assumptions made

MacIDA uses an entity-relationship model with an ACG without AND, OR, XOR links. The entity-relationship model is not object oriented, although it could be extended to become one. The main goal of MacIDA is to permit automated UI generation. This approach is grounded on two basic assumptions :

1. **Assumption 1: each entity and each relationship appearing in the entity-relationship model, that has attributes and/or is attached to at least one function, should be mapped onto a UI root window.**
   The generated root window contains consequently all the CIOs required for the input/display of elementary data used in this task. The set of these windows, which contains the set of all input/output information for the task, is therefore required for carrying out the task efficiently. This set is automatically created when the task is initiated. Each window, containing the required CIOs, is always present on the screen during run-time, but not necessarily visible nor active. This is why each window is associated with an item in the pull-down menu that lets the window be activated. We have to point out that the activation of a window does not imply the activation of the contained CIOs. The two mechanisms responsible for this activation are described later.

2. **Assumption 2: dialogue aspects are completely subordinated to functional aspects.**
   As soon as information are input or modified, the conversation management system is able to evaluate the current input state. This evaluation is instantly applied to particular CIOs, called *action CIOs*, such as a push button, an icon, a drawn button. When conditions required for triggering a typical application function are fulfilled, the user is warned by a presentation change of action CIOs: its activation alerts the user to their ability to trigger the function. As soon as the user pushes on that action CIO, the function is triggered and

executed: for instance, if a user pushes on a (Save) button, then the associated information will be stored in the database. Action CIOs are included in the IO-P (fig. 15) part since they are implied by the presentation.

This choice is compatible with the function triggering mode (explicit manual displayed, i.e. explicit user interaction with display objects, sub-subsection 7.1.2 above). If an implicit mode has been used instead (e.g., when the cursor is leaving an information field) or explicit manual undisplayed, i.e. user interaction with invisible objects (e.g., when pressing a function key such as [F10]), then action CIOs would have been replaced by control objects responsible for receiving these events to trigger the same function.

As the user proceeds to input new information and to trigger new functions, the main goal assigned to the task becomes closer. But the user can be forced to modify previously input information to satisfy validation rules. For example, when recording an order, a customer whose credit is no longer sufficient may be forced to reduce the ordered quantity or to forego making as many purchases. As indicated, the user has to modify the related information. The user does not need to specify this value change. Any information can be accessed or modified at any time. The only thing the user has to do is to locate the related information and to modify it. The system again automatically evaluates the input state, as existing validations may not now hold. The system then responds to the new value.

Two mechanisms can ensure that dialogues will be determined by functionality: firing/disabling action CIOs; consistency control between UI and functional machine. Details of these mechanisms will be given later. They have to allow designers a free choice of:

- the nature of events required for triggering functions: here, pushing on an action CIO is the only option because of the second assumption and because of the choice of function triggering mode (explicit manual displayed);
- the scope of triggering: e.g., a combined push button PB_1 for triggering functions Fc_1, Fc_2, and Fc_3 or a single push button PB_1 for triggering Fc_1, another one PB_2 for triggering Fc_2 and a last one PB_3 for triggering Fc_3.

Choices here are however constrainted by the ACG and the associated temporal relations (e.g., sequence, parallelism, condition).

The scope of MacIDA and its associated interaction styles are more restrictive than the general TRIDENT methodological framework. However, the above two restrictions, coupled with new specification constructs, allows complete automatic generation of the UI from the entity-relationship model. UI presentation design is computer-aided, whereas UI dialogue generation is fully automated.

7.2. Impact of the restrictions on specifying and generating a presentation

From hypothesis 1, one can deduce that UI presentation is derived from ACG and described by screen masks basically made up of CIOs for input/display information and actions CIOs. CIOs displaying error and help messages, pure static CIOs (e.g., a group box) can also be included.

The first hypothesis constraints AIO selection and AIO→CIO transformation steps (cfr. activity 2 introduced in section 4) to the applicability of the following selection rules :

- each entity is mapped onto a root window;

- each relationship is mapped onto a root window;
- each simple attribute is mapped onto a single-line edit box;
- each repetitive attribute is mapped onto a table;
- each multiple attribute is mapped onto a group box surrounding CIOs resulting from composing the multiple attribute (into edit boxes and/or tables);
- each function is mapped onto an action CIO, most of the time onto a push button.

However, in the case of more complex forms, the previously introduced expert system can become more helpful (cfr. subsection 4.2).

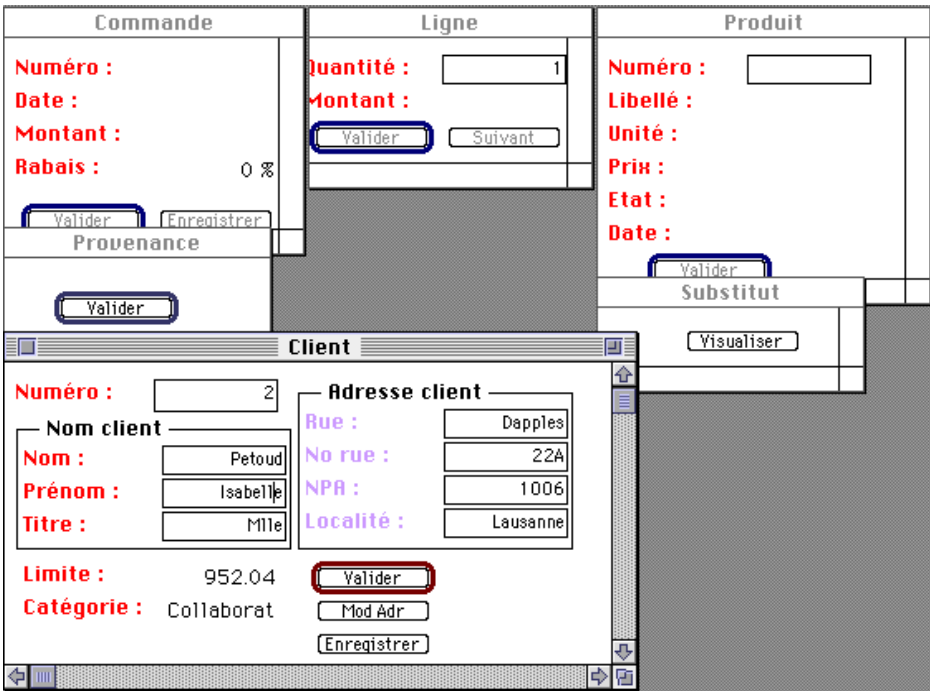

Fig. 14. UI generation.

The first hypothesis can also constraint the CIO placement step to the applicability of the following placement rules (fig. 14):

- single-line edit boxes, tables and group boxes are arranged in a single vertical column, are left justified in the root window to which they belong;
- identification labels of previous CIOs are placed in a single vertical column, are left justified between themselves and are bottom justified with respect to the CIOs they identify in the root window;
- action CIOs are vertically or horizontally justified at the bottom of each root window;
- root windows are arranged according to the location of entities and relationships in the entity-relationship model.

Again, in the case of more complex forms, the previously detailed placement strategies can become more helpful (cfr. subsection 4.2). The presentation could be adapted according to the user's needs by applying these rules and strategies within a presentation editor (cfr. subsection 4.1) such as the one developed by G. Zucchinetti [35].

7.3. Impact of the restrictions on specifying and generating an architecture

Constant presence of all root windows on the screen, resulting from the first assumption, entails that the interactive task only consists of one PU composed of these windows. Task decomposition into sub-tasks is no longer indispensable in this case. The resulting architecture (fig. 15) still conforms to the generic schema of architecture model (fig. 10), but the object structure is far simplified with respect to fig. 11 since only one CO-PU remains.

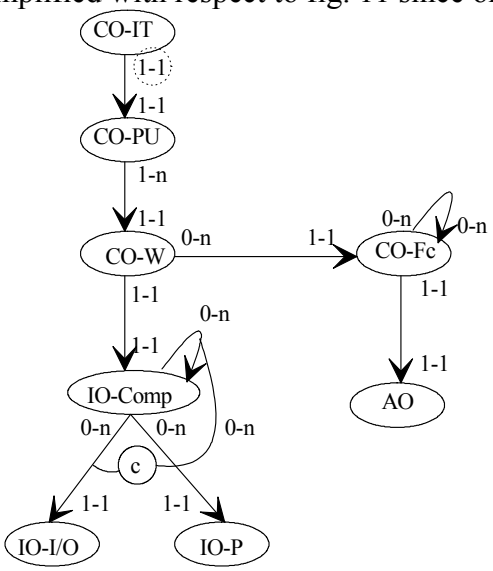

Fig. 15. Structure of Control Objects in MacIDA.

7.4. Impact of the restrictions on specifying and generating a dialogue

From first assumption, one can deduce that specifying a dialogue is largely simplified at the three levels defined in sub-section 6.2 :

1. inter-PU level dialogue is shrinked to a simple triggering of the only PU remaining. The behaviour of CO-IT therefore consists of only one rule for triggering the PU;
2. intra-PU level dialogue enables the user to browse freely between the PU's windows. The behaviour of the unique CO-PU is confined to one rule for creating all windows when initiating the interactive task and to the management rules for pull down menus required to activate windows;
3. intra-window level dialogue is reduced to an asynchronous conversation for the set of all CIOs in each window and for each action CIO. This total asynchronism results from the miss of links in MacIDA.

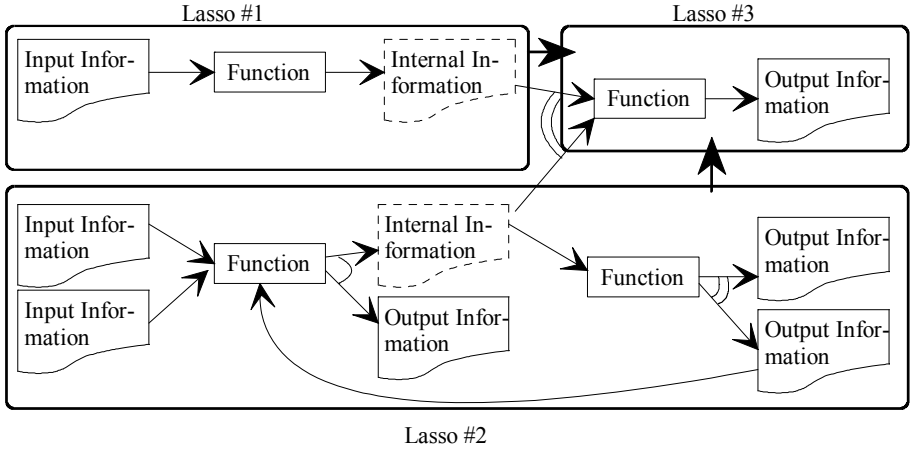

Fig. 16. Lasso technique.

From second assumption, one can deduce that intra-PU level (for menu items only) and intra-window level dialogues still allow some freedom constrainted by functional machine: these dialogue constraints are specified by ACG extensions that enables us to make the economy of state transition diagrams and specification rules.
The so-restricted CO behaviour can be entirely pre-programmed. We now show these extensions. Specifying a dialogue is a matter of enriching ACG with two techniques :

1. *defining the nature and scope of events for triggering functions*: this definition is achieved by a "lasso" metaphor grouping triggering of several functions into one event attached to a single action CIO (fig. 16). This metaphor decreases the amount of action CIOs by concentrating into one or many CIOs the set of all function triggerings in an interactive task. When functions are no longer triggered by their original action CIO, but by an action CIO resulting from a lasso, the function triggering mode becomes implicit.

2. *sequencing asynchronous inputs*: a sequence of input is forced (fig. 17). This sequence is specified by annotating the ACG with precedence links between information. These links are graphically represented by vertical bold arrows. A more directive conversation with a strict information ordering is gained.

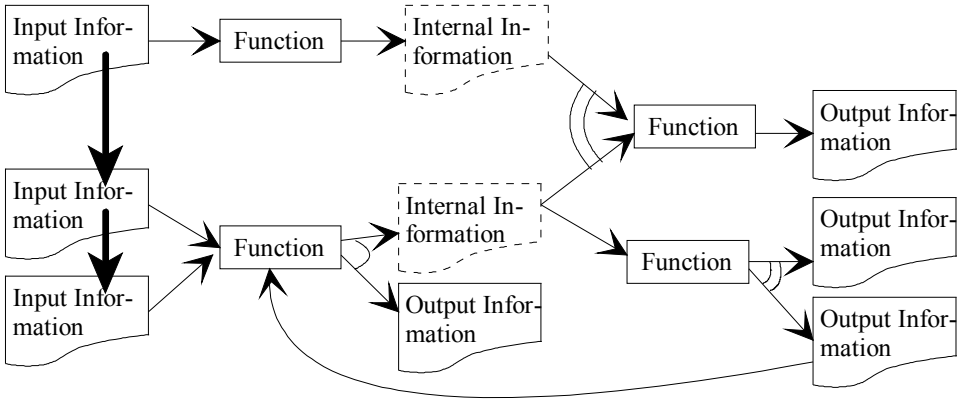

Fig. 17. Sequencing technique.

Extensions reported on the ACG should be consistent with the functional specifications. To insure this property, an extended ACG editor, called *dialogue editor*, will perform specific validations. For instance, it will check that a particular lasso surrounds all input information of a specific function.

7.5. UI dynamics

UI dynamics are realised with a small kernel which is independent of any particular interactive application, that is which is reusable across various ACGs. This kernel is built according to an electric circuit model recovering the ACG. Arcs are fired and disabled with respect to the node states (fig. 18): any information input or function triggering will fire the corresponding symbol in the circuit.

This firing proceeds in a parallel direction with the triggering of authorised functions. The role played by this kernel is double: on one hand, it insures a consistent information display with task's progression and, on the other hand, it restricts user's actions to the one which are functionally allowed. Two techniques ensure a correct execution :

1. the firing/disabling of action CIOs: action CIOs are activated whenever information input or function triggering requires it. This activation occurs when the pre-condition of related function, expressed in terms of input information, is verified. Similarly, action CIOs

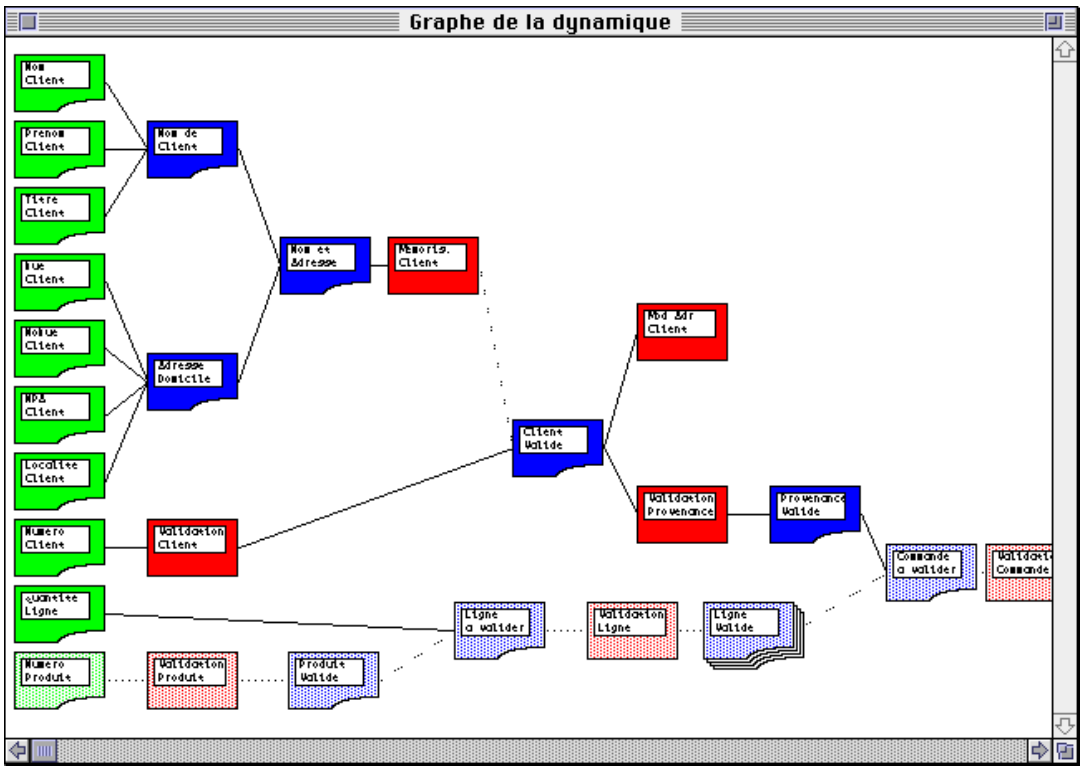

Fig. 18. The Activity Chaining Graph considered as an electric circuit.

which are related to functions whose triggering is not valid are deactivated. This deactivation occurs when the precondition of related function is not satisfied. For example, the input of a customer identification number provokes the activation of a push button for searching this customer;

2. the maintenance of consistency between UI and functional machine: this mechanism supports new information input by forgetting (deactivating) all information dependent of old values. Let us suppose that a customer id. has been provided and searched. The user becomes conscious that this customer is a wrong one. As early as the user provides a new id., all old information are forgotten to insure application consistency.

7.6. Original ergonomic contribution for an interactive application

MacIDA provides guidance and contextual help as a control panel with two windows :

- the first window of the control panel reflect graphically the state of the electric circuit (fig. 18). The user is then able to clearly visualise where (s)he is in the task progression, that is what (s)he has already done and what is remaining to do;
- the second window depicts the same information, but according to a natural language format with respect to the input context (fig. 19). In this example, the radio button highlighted on "+" for the product quantity means that the related information has already been provided and is correct ; conversely, the radio button highlighted on "?" for a product identification number indicates that this piece of information has already been provided, but not validated. Therefore, the current state is not yet correct.

These guidance features are automatically derived from the ACG. No extraneous specification effort is required from the human factors expert. Automated generation of such a control panel from the ACG as a contextual help and guidance for task's accomplishment is very worthwhile and can be expanded easily to build a new key activity: the automatic derivation of a help and guidance system from task analysis [21].

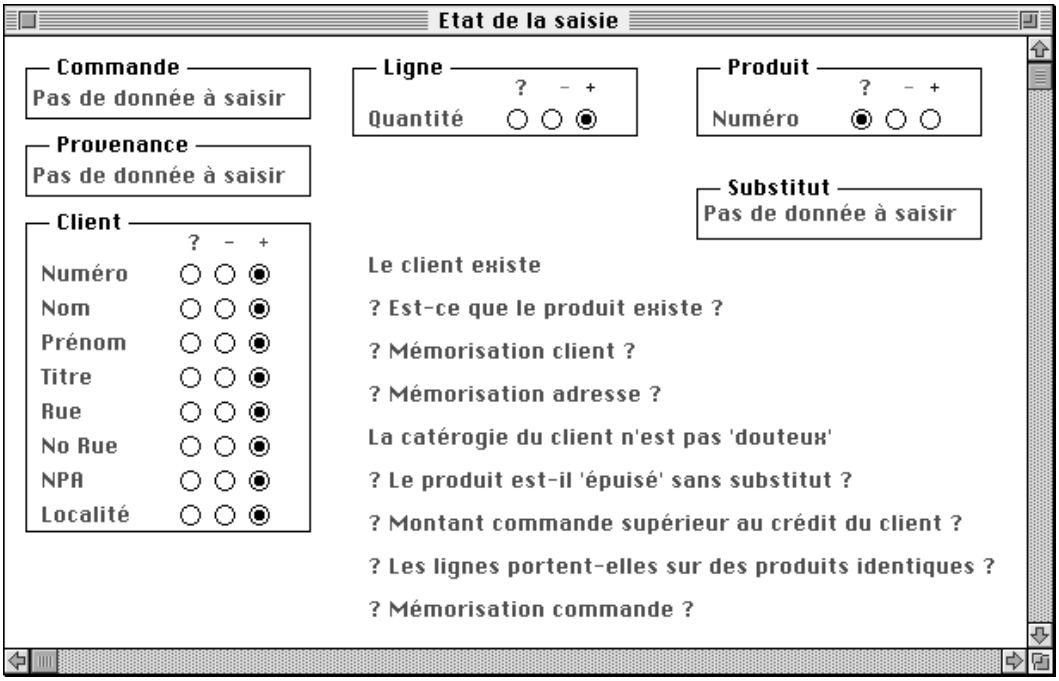

Fig. 19. Guidance Control Panel.

8. Conclusion

Five key activities have been examined and illustrated within a particular framework. Lessons learned show several benefits that have already been emphasised. A positive evaluation can be drawn for at least three issues:

- a better understanding of relationships between the different actors of the development life cycle of the interactive application;
- a highlighting of a precise set of design options to be defined for any particular design situation;
- an inducement to consider other key activities not only for business oriented applications but also for other domain areas.

8.1. Relationships between actors

Different actors in the complete development life cycle of an interactive application have been identified. We show here a supplemental advantage of the key activities: to recognise the relationships between those actors and to encourage them. Forming UI specification from the output of task analysis (activity 1 introduced in section 3) promotes a better communicability of the work done by task analyst, psychologist and usability engineer towards the software engineer.

Guiding presentation design by ergonomic rules (activity 2 introduced in section 4) helps the graphical artist who is responsible for the graphical appearance of the UI. Deriving an architecture from task analysis and presentation components (activity 3 introduced in section 5) is particularly well suited for the software engineer and/or the system engineer. The

proposed systematic approach provides guidance grounded on the work of task analyst without forgetting specific goals of the information system.

Specifying a high level conversation from task analysis (activity 4 introduced in section 6) should offer to the dialogue designer the faculty of conversation prototyping more rapidly and easily than with empirical manual methods. Specialising the methodological framework into a specification framework (activity 5 introduced in section 7) is more aimed at software engineers. This activity shows how important and affordable it could be to particularise a methodological framework into a specification framework.

8.2. Design options for a design situation

Each specification framework actually defines *a particular design situation*, characterised by a set of design options chosen among different alternatives. The clear identification of the five activities has collaborated to determining such options which are here summarised :

- a definition of activity domain: process control, office automation, information systems (e.g., accounting departments, selling, pharmaceutical area, medical area,...), information system level (operational, driving, decision);
- a task analysis method: by hierarchical decomposition, by constraints, by space-problems, by cognitive analysis,...
- a physical environment: availability of a screen editor, fourth generation language, library of custom widgets,...
- a selection of interaction style: command language, query language, natural language, questions & answers, menu selection, direct manipulation, form filling, function keys, iconic interaction, multi-windowing, multimedia interaction;
- a selection of dialogue attributes: dialogue control (i.e., internal, external, mixed), dialogue mode (i.e., sequential, asynchronous, mixed), function triggering mode (i.e., automatic, manual, implicit or explicit, displayed or undisplayed), metaphor (i.e., conversation based or universe based, navigation based);
- selection rules for choosing interaction objects: complete selection, customised selection, standardised selection (e.g., IBM CUA, Ms-Windows, Open Look, Humanoïd [21]);
- rules for placing interaction objects: complete placement, customised placement, standardised placement, column oriented (single, double, balanced double, multiple), form oriented, frequency oriented, conforming to the entity-relationship model's configuration, visual continuity,...

Other design options can also be investigated such as window selection type within a PU (maximal, minimal, functional, input/output, typed, grouped, or free [8]). Any design situation can be completely determined by choosing a particular value for all design options.

8.3. Completeness of key activities

Discovering and investigating the five activities has been confined in the beginning to business oriented applications. This should not mean that these activities are to be neglected when another domain area is concerned (e.g., graphical applications, multimedia services). They are likely to be revisited, but not forgotten. This should not also mean that these five activities are sufficient, whether for business oriented applications or not.

For instance, a desirable activity seems to be the systematic derivation of ergonomic criteria (e.g., compatibility, consistency, adaptability, work load) according to parameters of a task analysis. As a matter of fact, deciding that a particular criteria is more important than another

can only be achieved by considering the whole task structure. For a given task in a known context, explicit control might be appropriate. For the same task to be carried out in another context, with another user population, this criteria might become obsolete and should be changed by an implicit control.

Acknowledgements

The authors would like to greatly thank Gilbert Cockton for comments on earlier versions of this manuscript.

References

[1] Baecker, R.M. & Buxton, W.A.S., *Design Principles and Methodologies*, Chapter 11, in R.M. Baecker, W.A.S. Buxton, *Readings in Human-Computer Interaction - A Multidisciplinary Approach*, Morgan Kaufmann Publishers, Los Altos (Californie), 1987, pp. 483-491.

[2] Bodart, F. & Pigneur, Y., *Conception assistée des systèmes d'information - méthodes, modèls, outils*, 2nd ed., Masson, Paris, 1989.

[3] Bodart, F. & Provot, I., *Eléments de modélisation et d'architecture des IHM dans les SI - Une approche pragmatique*, Rencontres Lausannoises sur les Interfaces Homme-Machine, Troisième Cycle Romand d'Informatique, Lausanne, septembre 1989.

[4] Bodart, F., Hennebert, A.-M., Leheureux, J.-M., Sacré, I. & Vanderdonckt, J., *Architecture Elements for Highly-Interactive Business-Oriented Applications*, in Lecture Notes in Computer Science, Vol. 753, L. Bass, J. Gornostaev and C. Unger (Eds.), Springer-Verlag, Berlin, 1993, pp. 83-104.

[5] Bodart, F., Hennebert, A.-M., Leheureux, J.-M. & Vanderdonckt, J., *Towards a Dynamic Strategy for Computer-Aided Visual Placement*, in Proceedings of 2nd Workshop on Advanced Visual Interfaces AVI'94 (Bari, 1-4 June 1994), T. Catarci, M.F. Costabile, S. Levialdi, G. Santucci (Eds.), ACM Press, 1994, pp. 78-87.

[6] Bodart, F., Hennebert, A.-M., Leheureux, J.-M., Provot, I. & Vanderdonckt, J., *A Model-based Approach to Presentation: A Continuum from Task Analysis to Prototype*, in Proceedings of Eurographics Workshop on Design, Specification, Verification of Interactive Systems (Carrara, 8-10 Juin 1994), Eurographics Series, 1994, pp. 25-39.

[7] Bodart, F. & Vanderdonckt, J., *On the Problem of Selecting Interaction Objects*, in Proceedings of HCI'94 (Glasgow, 23-26 August 1994), G. Cockton, S.W. Draper, G.R.S. Weir (Eds.), Cambridge University Press, Cambridge, 1994, pp. 125-143.

[8] Bodart, F., Hennebert, A.-M., Leheureux, J.-M. & Vanderdonckt, J., *Computer-Aided Window Identification in TRIDENT*, in Proceedings of Fifth IFIP TC13 Conference on Human-Computer Interaction INTERACT'95 (Lillehammer, 27-29 June 1995), Elsevier Science Publishers, Amsterdam, 1995.

[9] Braudes, B., *Report of the Design Methodologies Subgroup*, SIGCHI Bulletin, Vol. 22, No. 2, October 1990, pp. 42-44.

[10] Carr, D.A., *Specification of Interface Interaction Objects*, in Proceedings of the Conference on Human Factors in Computing Systems CHI'94 (Boston, 24-28 April 1994), B. Adelson, S. Dumais, J. Olson (Eds.), ACM Press, New York, 1994, pp. 372-378.

[11] Cockton, G., *A New Model for Separable Interactive Systems*, in Proceedings of the Second IFIP TC13 Conference on Human-Computer Interaction INTERACT'87 (Stuttgart, 1-4 September 1987), H.-J. Bullinger, B. Shackel (Eds.), Elsevier Science Publishers, Amsterdam, 1987, pp. 1033-1038.

[12] Cockton, G., *The Architectural Bases of Design Re-use*, in User Interface Management and Design, D.A. Duce, M.R. Gomes, F.R.A. Hopgood and J.R. Lee (Eds.), Springer-Verlag, Berlin, 1991, pp. 15-34.

[13] Figarol, S. & Javaux, D., *From Flying Task Description to Safety Assessment,* Le Transpondeur, décembre 1993.

[14] Frese, M. & Zapf, D., *Action as the Core of Work Psychology: A German Approach*, in Handbook of Industrial and Organizational Psychology, H.C. Triandis, M.D. Dunnette, L.M. Hough (Eds.), Vol. 4, Consulting Psychologists Press, Palo Alto, 1994, pp. 271-340.

[15] Hennebert, A.-M., *La hiérarchie des Objets de Contrôle: Règles de construction*, Internal TRIDENT report, Institut d'Informatique, Namur, 6 April 1994.
[16] Johnson, P., *Human-Computer Interaction, Psychology, Task analysis and Software Engineering*, McGraw-Hill Book Company, London, 1991.
[17] MacLeod, M. & Bevan, N., *MUSiC Video Analysis and Context Tools for Usability Measurement*, in Proceedings of the Conference on Human Factors in Computing Systems INTERCHI'93 (Amsterdam, 24-29 Avril 1993), S. Ashlund, K. Mullet, A. Henderson, E. Hollnagel and T. White (Eds.), ACM Press, New York, 1993, p. 55.
[18] Marshall, C., Nelson, C. & Gardiner, M.M., *Design Guidelines*, in Applying Cognitive Psychology to User-Interface Design, M.M. Gardiner, B. Christie (Eds.), John Wiley & Sons, New York, 1987, pp. 221-278.
[19] Miyata, Y. & Norman, D.A., *Psychological Issues in Support of Multiple Activities*, chapter 13, in User Centered Design - New Persepectives on Human-Computer Interaction, D.A. Norman, S.W. Draper (Eds.), Lawrence Erlbaum Associates Publishers, Hillsdale, 1986, pp. 266-284.
[20] Moher, T., Dirda, V., Bastide, R. & Palanque, Ph., *A Bridging Framework for the Modeling of Devices, Users, and Interfaces*, Technical Report UIC-EECS-ICE-94-13, University of Illinois, 1994.
[21] Moriyon, R., Szekely, P. & Neches, R., *Automatic Generation of Help from Interface Design Models*, in Proceedings of CHI'94 (Boston, 24-28 avril 1994), B. Adelson, S. Dumais, J. Olson (Eds.), ACM Press, pp. 225-231.
[22] Nigay, L. & Coutaz, J., *A Design Space For Multimodal Systems: Concurrent Processing and Data Fusion*, in Proceedings of the Conference on Human Factors in Computing Systems INTERCHI'93 (Amsterdam, 24-29 April 1993), S. Ashlund, K. Mullet, A. Henderson, E. Hollnagel and T. White (Eds.), ACM Press, New York, 1993, pp. 172-178.
[23] Petoud, I. & Pigneur, Y., *Des specifications fonctionnelles à l'interface-utilisateur sans programmation*, in Actes du Colloque sur l'ingénierie des interfaces homme-machine, Sophia-Antipolis, May 1989.
[24] Petoud, I. & Pigneur, Y., *An automatic and Visual Approach for User Interface Design*, in Proc. of 4th IFIP Working Conference on User Interfaces, Nappa Valley, August 1989.
[25] Petoud, I., *Génération automatique de l'interface homme-machine d'une application de gestion hautement interactive*, PhD, Ecole des Hautes Etudes Commerciales, Université de Lausanne, Chabloz, Tolochenaz, 1990.
[26] Provot, I., *L'enregistrement d'une commande téléphonée*, Internal TRIDENT report, Institut d'Informatique, Namur, 17 December 1993.
[27] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. & Lorensen, W., *Object-Oriented Modeling and Design*, Prentice-Hall, Englewood Cliffs, 1991.
[28] Shepherd, A., *Analysis and Training in Information technology Tasks*, in Task Analysis for Human-Computer Interaction, D. Diaper (Ed.), Ellis Horwood, Chichester, 1989, pp. 15-55.
[29] Shneiderman, B., *Designing the User Interface, Strategies for Effective Human-Computer Interaction*, 2nd ed., Addison-Wesley, Reading, 1992.
[30] Siochi, A.C. & Hartson, H.R., *Task Oriented Representation of Asynchronous User Interfaces*, in Proceedings of the Conference on Human Aspects in Computing Systems CHI'89 (Austin), pp. 183-188.
[31] Sutcliffe, A., *Defining the Requirement for HCI Design Methods*, SIGCHI Bulletin, Vol. 26, No. 2, Avril 1994, pp. 21-23.
[32] Vanderdonckt, J. & Bodart, F., *Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection*, in Proceedings of the Conference on Human Factors in Computing Systems INTERCHI'93 "Bridges Between Worlds" (Amsterdam, 24-29 April 1993), S. Ashlund, K. Mullet, A. Henderson, E. Hollnagel and T. White (Eds.), ACM Press, New York, 1993, pp. 424-429.
[33] Vanderdonckt, J., *Guide Ergonomique des interfaces homme-machine*, Presses Universitaires de Namur, Namur, 1994, ISBN 2-87037-189-6.
[34] Wasserman, A.I., *Extending State Transition Diagrams for the Specification of Human-Computer Interaction*, IEEE Transactions on Software Engineering, vol. SE-11(8), August 1985, pp. 699-713.

[35] Zucchinetti, G., *Réalisation d'un éditeur d'interface homme-machine*, Mémoire de diplôme postgrade en informatique et organisation, Ecole des Hautes Etudes Commerciales, University of Lausanne, Lausanne, 1990.