

# Issues in Design and Implementation of Multimedia Software Systems

A.Bianchi\*, P.Bottoni<sup>+</sup>, P.Mussio\*

\* *Dipartimento di Elettronica per l'Automazione, Università di Brescia*

<sup>+</sup> *Dipartimento di Scienze dell'Informazione, Università di Roma "La Sapienza"*  
*{bianchi, mussio}@bsing.ing.unibs.it, bottoni@dsi.uniroma1.it*

## Abstract

*This paper describes a user-centered approach to Multimedia Software System aimed at satisfying requirements of end-user computing. MSS design and development explicitly accounts for their capabilities of capturing and producing textual, pictorial, sound and haptic events and for the requirements of empowering users in the execution of their tasks and preventing them from getting lost in the multimedia virtual space and from the consequences of their incorrect operations.*

## 1. Introduction

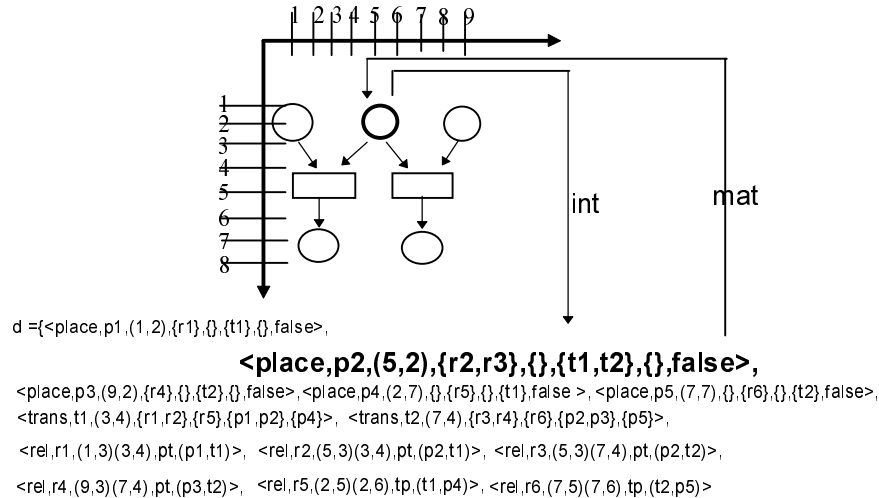
Multimedia Software Systems (MSSs) are characterized by their capabilities of interacting with their users by capturing and producing textual, pictorial, sound and haptic events. They are increasingly used by end-users experts in some fields other than computer science. End-users develop applications and produce results, of which they are responsible, to be shared with other users [1]. End-users need to control the interactive activity being performed and guarantee the quality of their results.

The paper explores some software engineering issues in the design and implementation of MSSs [2] satisfying these user requests. The discussion is based on an interaction model and a definition of visual language recently introduced by the Pictorial Computing Laboratory group [3, 4] and on experiences in the development of medical applications [5, 6]. The designer a) adopts as the core notation of the interaction a generalization and formalization of the experts notations used to communicate and reason on procedures, results and data in the traditional working environment; b) explicitly takes into account the constraints imposed by the current technologies; c) organizes the MSS output events so that they can be perceived and interpreted by the user; d) organizes the MSS input capabilities so as to correctly captures and interprets users gestures and sounds. The satisfaction of these design requirements is checked by evaluating the usability of the result of each design and implementation activity. Traditional models of software life cycles (e.g. the waterfall model), underestimate interaction and usability aspects. More recently, some models, aimed at improving usability of

interactive systems, stress MSS evaluation from the user point of view, e.g. the star model [7] but underestimate the influence of currently available technologies for interaction on MSS development. The WinWin Spiral Model [8] suggests that the use of the MSS is not the final stage of its life cycle, but rather is a source of suggestions for new applications and improvements of the MSS itself. Our approach takes into account these instances and the fact that MSS use suggests improvements of the interaction technologies. Due to lack of space, we only discuss the case of Visual Interactive System (VIS) in which interaction is based on visual messages displayed on a screen. Section 2 provides an overview of a task analysis technique and section 3 shows how its results are used to derive the visual languages used during the interaction. Section 4 presents an overview of the proposed life cycle of VISs. Section 5 summarizes the formal tools used to specify VISs. Sections 6 and 7 refine the life cycle of VISs. In section 8 conclusions are drawn.

## 2. User's Notations in User's Task

To understand how to support end-users in their performances, we start by analysing their activity before MSS introduction. In general, a community of experts develops specialised notations to describe, document and communicate the procedures they perform in accomplishing a task, as well as their intermediate and final results. By this notation they build documents in which texts, pictures and graphs are organised and assume a meaning according to the conventions adopted by the experts. The document appears to users as an image, i.e. a collection of *characteristic structures* (**cs**) which constitute functional or perceptual units for an expert reading or composing a document. In a text, characters as well as words are **css**. In sketches, graphs, and pictures, **css** are often specific to a discipline [5]. Experts name each type of **cs** they use and characterise it by a set of attributes, each attribute assuming its value in a well defined set of values. The name of the type and the set of values assumed by the attributes identify an instance structure of a given type and constitute its *description*.



**Fig. 1 A sketch of a Visual Sentence in the Petri Net VL: image part, interpretation and an element (pair) of the int and mat functions**

We call the association of a **cs** and its description a *characteristic pattern (cp)*.

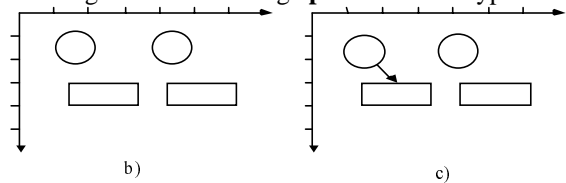
An expert composes a document by assembling simpler **css** into more complex ones, following the community conventions, so that the document itself becomes a complex **cs**, called image *i*. A different expert looking at the document sees a complex **cs**, which s/he can correctly interpret if s/he knows the set of **cps** and the conventions for their composition.

Fig.1 schematizes a Petri Net, designed by an expert to communicate a view on a system. The Petri Net is represented as a set of **cps** in the proposed formalism. The **css** are the circle, the box and the arrow; they give origin to **cps** when associated with their descriptions, which identify them as place, transition and relation. These descriptions are attributed symbols here represented by strings of the following structure:  $\langle \text{type identifier}, \text{instance identifier}, \text{coordinates of the } \text{cs} \text{ typical points}, \text{sets of } \text{cps} \text{ with which a relationship exists}, (\text{optionally}) \text{ data on state of the } \text{cp} \rangle$ .

A **cs** is related to its description by two functions: *int*, mapping the structure into the attributed symbol and *mat*, mapping the symbol into its structure. A **cp** is described by a triple  $\text{cp} = \langle \text{cs} \rangle, \langle \text{attributed symbol} \rangle, \langle \text{int}, \text{mat} \rangle$ . A document is an image *i* formed by many **css**, which must be interpreted to understand its meaning. The definition of visual sentence (**vs**) gives account of this fact: a **vs** is a triple  $\text{vs} = \langle i, d, \langle \text{INT}, \text{MAT} \rangle \rangle$  where *i* is an image, *d* is a description, i.e. a set of symbols, *INT* a function mapping the **css** of *i* into symbols in *d* describing their meaning and *MAT* a function mapping symbols in *d* into **css** in *i*. A Visual Language (**VL**) is a set of visual sentences.

For example Fig. 1 sketches a **vs** in which a diagram is associated with a description specifying its meaning as a Petri Net. A circle is characterized by one typical point, its center; the box by the center of the top segment and

the arrow by a pair: its initial and final points. For simplicity, the action of the *INT* and *MAT* functions is shown for only one **cp**. This **vs** belongs to the visual language **PN**, whose elements can be generated from the visual alphabet **PN\_K** consisting of: 1) a **cp** for places (of type place whose image part is a circle); 2) a **cp** for transitions (of type trans whose image part is a rectangle); and 3) a **cp** for relations (of type rel whose image part is an arrow) [4]. Experts describing systems as Petri Nets express their models as visual sentences in **PN**. They build a Petri Net in several steps, following a personal line of thought and instating **cps** of the three types.



**Fig.2. Two hand-drawn images in TVL representing the construction of a relation in modeling a system by PN.**

Fig.2 shows two images, produced in such a process. They give rise to **vss** when associating a meaning with each structure in them, i.e. each structure is the image part of a **cp** in **PN\_K**. These two **vss** are not Petri Nets, but represent two intermediate results in the construction of the complete Petri Net of Fig.1.

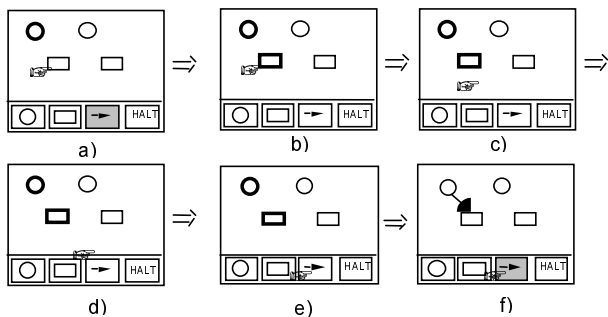
In performing an activity, experts also produce documents expressing intermediate results. The whole set of documents produced to accomplish a task constitute a **VL**, the *Task Visual Language (TVL)*. To implement a **MSS** satisfying the usability requirements of Section 1, we assume the **TVL** as a basis to develop the **MSS** surface.

### 3. Interaction Visual Language

In a VIS images on the screen are seen as electronic documents built in a generalized and formalized user notation as well as the medium through which interaction occurs. On the screen, the **css** of the notation are represented by sets of pixels and their descriptions are *attributed symbols* listing the type name and the values assumed by the attributes for a specific **cs**. A **cp** is now a triple:  $cp = (\langle \text{set of pixels} \rangle, \langle \text{attributed symbol} \rangle, \langle \text{int, mat} \rangle)$ . In the following, a set of pixels will be denoted by **cs** if it is a generic characteristic structure or **i**-for image-when the **cs** coincides with the image on the screen. A **vs** is now associated with the digital image **i**,  $vs = \langle i, d, \langle \text{INT, MAT} \rangle \rangle$ .

The screen must show to the expert the document under development -a **vs** in **TVL**- and the set of icons, words and widgets called *frame*, which surround or are superimposed to the document. The **vs** in **TVL**, together with the frame, recall the expert of the stage of the activity and make clear what s/he can do and how to do it. Moreover, the **vss** in **TVL** on the screen are augmented versions of the **vss** in **TVL** on paper: their **cps** are able to adapt their representation to the state of the computation -e.g. they change color or shape when selected- and are often associated to computational activities. The frame and the **TVL** document together form the image part of a new **vs**, the *active visual sentence*. The set of all the active **vss** which can be generated using a given VIS constitutes the *Interaction Visual Language*, **IVL**.

Fig.3 shows a sequence of image part of **vss** in **IVL** observed in establishing the same relation as in Fig.2.

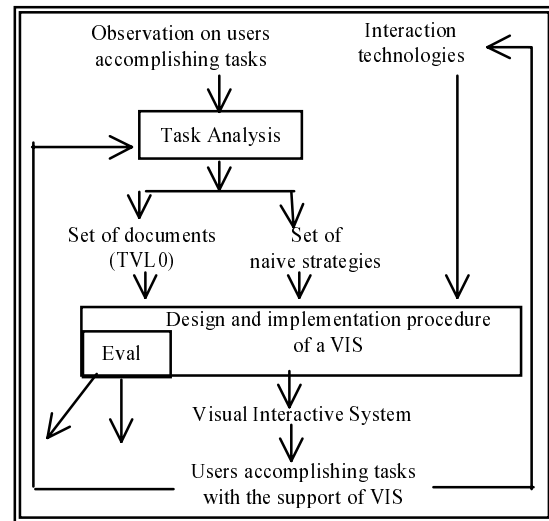


**Fig.3. A sequence in IVL describing the gestures required to generate an arrow**

Fig.3a is the echo check of the system dragging the pointer upon the transition structure as effect of the user moving the mouse; Fig.3b echoes the system highlighting the transition structure as effect of the user clicking the mouse; Figs.3c to 3e are samples of the echo check of the system dragging the pointer on the screen as effect of the user moving the mouse; Fig.3f is the echo check of the system generating an instance of relation and disabling the arrow button as effects of the user clicking the mouse.

### 4. An Overview of the VIS Design and Implementation cycle

The **VIS** design requires the definition of: 1) the **TVL** alphabet and composition rules; 2) the new features which augment the **TVL cps**; 3) the set of **cps** (image, corresponding computational meaning,  $\langle \text{int, mat} \rangle$ ) which a) help users not to get lost in the virtual space, b) determine which operation can be performed by the user; and 4) the rules for their composition. The designer must also define how the user actions and the feedback to them can be performed by the available interactive technology.



**Fig.4. Life cycle of a Visual Interactive System when designed from scratch**

Fig.4 outlines the life cycle of VIS, when designed from scratch. First, the designer identifies the **TVL** by the Task Analysis, which consists of the systematic observation of users accomplishing the task before the introduction of the new tools. It results in: a) the set of documents that experts use to document task execution, which constitutes the **VL TVL0**; b) the description of the set of sequences of actions an expert performs to execute a task (set of *naive strategies*). The two images of Fig.2 belong to a same naive strategy.

The results of Task Analysis, together with a set of descriptions of the available interaction technology, are the inputs to the VIS Design and Implementation activity, whose output is the Visual Interactive System. In user-centered design it is important that the results of each activity be evaluated with respect to their usability in the task performance context, to their internal consistency, their consistency with respect to previous established tools and procedures, their feasibility with the current technology [7]. The evaluation activity is especially signaled by the *Eval* box inside each activity box.

The feedback paths to be followed when an evaluation activity fails are not made explicit in the following figures, with two exceptions in Fig.4. These exceptions account for the facts that, a) using a VIS, experts modify their model of the task, identify new potential use of the tools and also new features they would like to have in the tools, and change their way of performing a task and diffuse the results; b) VIS use often suggests how to improve or modify the current technology and determines its acceptance or rejection -think for example to the mouse, track-ball, touch pad evolution.

## 5. Formal tools for VIS specification

In order to detail the design and implementation procedure, we need a deeper insight on VL formal definition [3, 9] and on the formalisms adopted to describe the interaction control. In [9], it was shown how an **IVL** can be specified by a family of rewriting systems [10], the *enabling visual conditional attributed rewriting system with groups* (*gevCARW*). A *gevCARW* is specified by giving an alphabet and a set of rules on it. Rules can be applied to a *vs1* to transform it in a new *vs2* and give account of the simultaneous evolution of the pictorial, description and the relational parts of a *vs*. A *gevCARW* is used to generate *vss* in an IVL starting from an axiom *vs0*. When the system is switched on, the image part of *vs0* appears on the screen. *vs0* identifies the initial state of the document in TVL and the set of actions which can be performed on it (*legal actions*).

A rule in a *gevCARW* describes how executing these actions transforms a *vs*. This requires that attributed symbols in the *vs* description admit functions as values of some attribute and that an interpreter capable of interpreting a description as a program be available. An **IVL** is thus specified by a pair  $\langle vs0, gevCARW \rangle$ . Note that also **TVL** can be described by such a pair. In both cases, this pair allows the generation of every legal sequence of *vss* in the language. Hence **TVL** and **IVL** result organised in sequences, each being the trace of an interaction strategy applied by the expert starting from *vs0*.

As to interaction control, we first need to define what MSS does when the user performs an action. To this end we introduce *Dialog Operations (DOs)* which are pairs  $\langle \text{user action}, \text{system reaction} \rangle$ . A **DO** can occur only on a suitable *vs*. For example, the creation of an arrow in a Petri Net by the VIS introduced in Fig.3 is the result of a  $DO = \langle \text{select the relation trigger}, \text{establish the relation} \rangle$  performed on a *vs* in which the potential arguments of the relation are selected and the pointer is on the correct button, i.e. the *vs* is in the set  $\{vs \mid \text{only one place and one transition are selected and button is pointed in } vs\}$ .

Both user action and system reaction can be defined at different levels of abstraction. User actions can be defined

at the low *gesture level*, describing the elementary user actions on a specific tool, for example *move\_mouse*, *strike\_key*, etc. *High-level actions* (or simply *actions*) describe sequences of gestures achieving a sub-goal in the task, for example *select\_object*, *type\_word*, etc. In a similar way, a system reaction can be described at a low level by specifying an algorithm which prescribes how to translate a user gesture into a system event, which function to compute in consequence, how to determine the output. At a more abstract level, system reaction is described as a set of post-conditions on the active *vs*. These descriptions are coupled into *dialog acts (das)*, pairs  $da = \langle \text{action description}, \text{reaction description} \rangle$ . A **DO** is an instance of **da** executed by the human performing the described action on a real input device and by the system interpreting the reaction description on the set of currently available data. An interaction is a sequence of **DOs**. Dialog acts are used to define *Interaction Control Automata (ICA)*, specifying dialogue control at different levels of abstraction. An **ICA** is a Conditioned Transition Network, an extension of Augmented Transition Networks [11] from which interpreters of interactions based on high- or low-level actions can be derived. A state in **ICA** reflects the current state of the interaction, for example recording which **cps** are currently selected, and which user actions are consequently legal. By performing an action, the user determines the execution of a computational activity and a transition to the state which records the results of that activity. The transition is therefore labelled by a **da**. A condition may be associated with a transition. In this case, activity execution and state transition occur only if the associated condition is satisfied. An activity results in: 1) the modification of the appearance and properties of **cps** in the current *vs*; or 2) the generation or deletion of new **cps**. To give visual feedback to the user, an operation of the first type must be associated with every transition. Hence it has the same effect as the application of a rule. In designing a VIS, three kinds of **ICAs** can be used to specify the desired system behaviours and to steer the VIS implementation:

- Action Control Automata (ACA), whose **das** couple high-level action and computation descriptions;
- Form of Interaction (FoI) Automata, describing how human and machine can perform a set of high level **do** through sequences of low-level dialog operations, i.e. how a human action is performed as a set of gestures on a given interaction tool;
- Gesture Control Automata (GCA), whose **das** couple gesture and computation descriptions.

**GCA** are the low-level **ICAs** which specify the sequence of gestures the users must follow with the available tools in order to realized the actions. **GCA** are obtained by **ACAs** gluing **FoI** on them.

## 6. The Design and Implementation of a VIS

Fig.5 details Design and Implementation of a VIS. In a first stage, the results of the task analysis and the descriptions of interaction technologies are translated into the formal specification of **TVL**, the set of **das** necessary to specify the user strategies, and the available interaction technologies. By the activity GENERALIZATION AND FORMALIZATION 1, the sets of documents and naive rules are used to generate a **gevCARW(TVL)** and an axiom **vs0'**. This pair defines a **TVL** which must be a superset of **TVL0** and in which every sequence should result in a meaningful execution of the task. This last condition can be checked by a usability evaluation procedure.

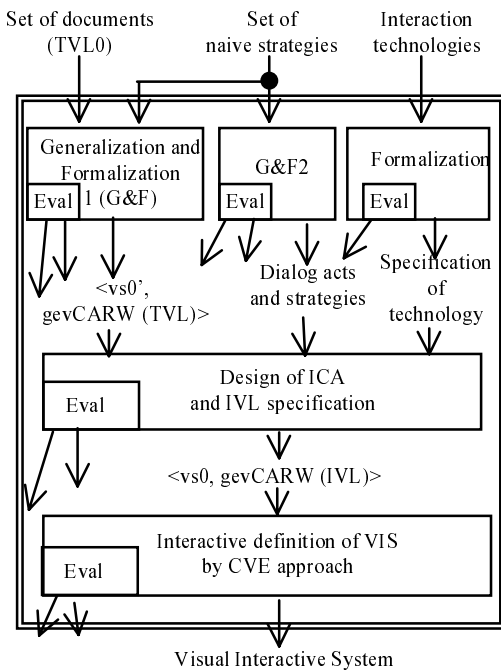


Fig.5. VIS design and implementation

In the activity GENERALIZATION AND FORMALIZATION 2, the set of naive strategies is studied and a *Set of Dialog Acts(SDA)* necessary to their execution is defined, formalised and generalised. This activity evaluation requires dialogue acts to be mimicked in realistic situations to verify that each sequence of **DOs** implementing the dialogue act is *natural* for the user.

In a third activity, SELECTION AND FORMALIZATION, technologies for interaction are selected according to the actions to be performed and to the types of data to be managed, and expressed as *Event-Based Specifications*. This activity evaluation consists of verifying that the obtained formalisation allows the user to perform all the actions needed to accomplish the task.

The results of these three activities are the input to the Interactive Specification of Control Automata and Interaction Visual Languages, commented on in the next

section. This phase produces the specification of the IVL, through the definition of **vs0**, and of a **gevCARW**. Next, given a library of programs performing the required computations, a VIS can be implemented following the CVE approach [5, 13] in the last step in fig.5.

## 7. ICA Design and IVL specification

In the ICA Design and IVL specification activity (Fig.6), the designer produces the formal specification of the Interaction Visual Language (**IVL**) as a couple **<vs0, gevCARW(IVL)>**.

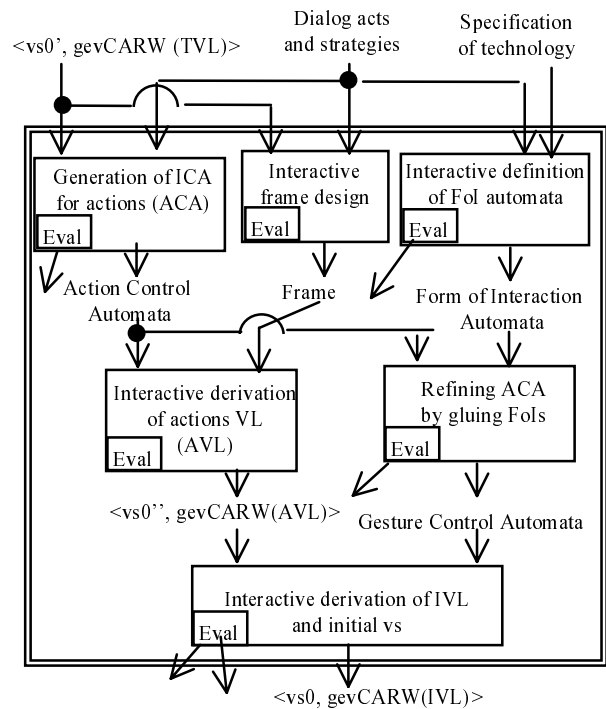
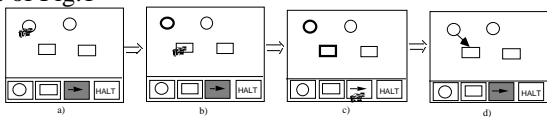


Fig.6. Interactive specification of ICA and IVL

The first activity is the definition of the high-level ACA which guarantees the complete control of the **VIS** by the user and prevents users from the consequences of their incorrect operations. ACA transitions are labeled by high-level dialog acts and its states identify situations in which high-level actions can be performed by the users. An algorithm for ACA derivation from a specific form of **gevCARW(TVL)** and a set of dialog acts is described in [4]. ACA usability is evaluated by paper mock-up.

The physical layout of the VIS is specified by the INTERACTIVE FRAME DESIGN activity. The frame, constituted by the set of **css** (icons, words and widgets), provides the user with the context in which a user action or a computer activity is performed. The designer establishes: a) **css** by which actions are denoted; b) **vs** layout, establishing the area where the **TVL css** are shown; c) frame layout; d) visual feedback to user gestures; e) visual feedback towards concurrent

computations (for example e-mail warning messages); f) visual cornerstones. Frame usability is evaluated by verifying that the frame always provides the context and the general tools of the application. Moreover, in this phase it is necessary that a feedback to users is foreseen for each user action in order not to get them disoriented [12]. The pair SDA and EBS allows the generation of the Form of Interaction (FoI) Automata described above. This activity usability evaluation consists in verifying that the obtained family of automata correctly describes the way in which users perform actions through the available interaction tools. The next activity INTERACTIVE DERIVATION OF ACTIONS VL has the purpose to produce a VL satisfying the requirement of prevention of illegal user actions, and whose vss have image parts fitting the established frame. ACA and the frame allow the derivation of the Action VL (AVL), formally expressed by a gevCARW, and by the axiom vs0'' from which all the vss in AVL can be produced. AVL is the visual language constituted by all the vss that can be generated during the interaction as effect of some DO, i.e. the vss resulting from the system reaction to a user action performed when the document is in a particular state. Each vs in AVL describes the system reactions to an action. Each sequence of vss in AVL describes a strategy to obtain a (intermediate) result in constructing a vs in TVL. In this sense, Fig.7 shows a sequence in the AVL for PN, describing three actions in the construction of a relation between a place and a transition in constructing the vs of Fig.1



**Fig.7 A sequence in AVL describing the construction of a relation as in fig. 2.**

The usability evaluation of this activity is carried out by using a mock-up to simulate user task execution as a sequence of high level actions and verifying that legal user actions are correctly managed, illegal user actions are trapped and do not produce situations out of user's control, and that users may always understand their working situation. In order to deal with user gestures the next activity has the purpose to satisfy the control requirements by considering the available interactions tools. To this end, ACA is refined so that a final *Gesture Control Automata*, GCA is obtained. GCA describes how to control the interaction at the abstraction level of gestures: it specifies the sequence of gestures the users must follow with the available tools in order to realized the actions.

Finally IVL, expressed as the pair <vs0, gevCARW>, is derived by the activity IVL AND VS0 DERIVATION. Some vss in IVL are also vss in AVL. These vss

correspond to situations created at the end of an action execution. For example Fig.3a describes the reaction to the last gesture for transition selection after place selection. Figs. 3b to 3f describe the reactions to the gestures in selecting the arrow button by moving (causing the movement of the pointer) and clicking the mouse (causing button highlighting and relation generation).

## 8. Conclusions

This paper introduces a view on the development of MSS, in the case that communication occurs through a screen. It is based on the use of gevCARWs to define image part and computational meaning of the messages exchanged in the interaction [9]. Its generalisation to other modes of interaction requires the formalization of the set of signs used in the communication and the precise description of their temporal and spatial properties.

## 9. References

- [1] Brancheau J.C., Brown C.V., The management of end-user computing, *ACM Computing Surveys*, 25, 437-482, 1993.
- [2] Hirakawa M., Do software engineers like multimedia?, *These Proceedings*, 1999.
- [3] Bottoni P., Costabile M.F., Levialdi S., Mussio P., Defining visual languages for interactive computing, *IEEE Trans. on System, Man and Cybernetics*, 27, 773-783.
- [4] Bottoni P., Costabile M.F., Levialdi S., Mussio P., Specifying dialog control in Visual Interactive Systems, *Journal of Visual Languages and Computing*, vol.9, n.5, 535-564, 1998.
- [5] Bianchi N., Bottoni P., Mussio P., Protti M., Cooperative Visual Environments for the Design of Effective Visual Systems, *JVLC*, 4, 357-382, 1993.
- [6] Mussio P., Pietrogrande M., Protti M., Simulation of hepatological models: a study in visual interactive exploration of scientific problems, *JVLC*, 2,75-95, 1991.
- [7] Hix D., Hartson H.R., *Developing User Interface*, John Wiley-New York., 1993.
- [8] Boehm B., Egyed A., Kwan J., Madachy R., Developing multimedia applications with the WinWin Spiral Model, in *Software Engineering Notes*, vol.22, n.6, M.Jazayeri, H.Schauer (eds.), Springer, 20-39, 1997
- [9] Bottoni P., Chang S.K., Costabile M.F., Levialdi S., Mussio P., On the Specification of Dynamic Visual Languages, *Proc. IEEE Symposium on Visual Languages'98*, 1998, 14-21.
- [10] Dassow J., Păun G., *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, 1989.
- [11] Green M., A survey of three dialogue models, *ACM Trans. on Graphics*, 5(3):244-275, 1986.
- [12] Preece J., Rogers Y., Sharp H., Benyon D., Holland S., Carey T., *Human-Computer Interaction*, Addison-Wesley, 1994
- [13] Bottoni P., Mussio P., Olivieri B., Protti M., A completely visual environment for agent-based computing, *Proc. AVI'98*, T.Catarci, M.F.Costabile, G.Santucci, L.Tarantino eds., ACM Press, 261-263, 1998.