

# On the Life Cycle 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"

## Abstract

*Multimedia Software Systems (MSS) are characterized by their surface capabilities of collecting and producing textual, pictorial, sound and haptic events. A crucial requirement on the MSS-user interaction is to prevent users from getting lost in the multimedia virtual space and from performing incorrect operations. To this end, MSS development requires the coordination of computational and output generation processes with human cognitive and gestural processes.*

*These requirements are becoming stricter because MSS are increasingly used by end-users -experts in some fields other than computer science- to develop their own applications and produce results to be shared with other users. Their satisfaction calls for the development of new techniques in MSS design, implementation, and evaluation. MSS design must explicitly take into account the dynamic of time-space organization of the surface as well as the constraints imposed by the current technologies. Implementation must explicitly take into account the management of the surface characteristics with respect to the human user profile. Evaluation is no longer limited to MSS algorithmic and computational aspects, but must also explicitly take into account its usability. MSS use cannot be considered as the final stage of its life cycle but rather as a source of suggestions for new applications and improvements of the interaction technologies and of the MSS itself.*

*This paper describes a user-centered approach to MSS design and development, aimed at allowing users to control the activities performed through the MSS. It also describes the life cycle of a MSS when implemented from scratch and a set of technologies for visual interaction and interactive computation are available.*

## 1. Introduction

Designers of Multimedia Software Systems (MSS) need to specify the coordination of the cognitive and gestural processes performed by humans with the computational and input/output processes performed by the System. To this end, designers must take into account the capabilities and the constraints of the available technology and organize the MSS output events in space and time so that they can be perceived and understood by the human user. Moreover, they must organize its input capabilities so that it a) correctly captures in space and time the gestures and the sounds produced by the users and b) properly interprets them.

A crucial requirement for MSS-user interaction is to prevent users from getting lost in the multimedia virtual space and from performing incorrect operations. This requires the MSS a) to give the humans some visual, textual or sound cornerstones so that they can continually maintain their bearings in the virtual space; and b) to trap illegal users' actions. A contrasting but equally crucial requirement is that MSS be tools of thought for the users and do not restrict their freedom in developing their strategies in solving problems. These requirements are becoming stricter because MSS are increasingly used by end-users -experts in some fields other than computer science- to develop their own applications and produce results to be shared with other users [1].

These requirements call for new techniques of MSS design, development, evaluation and use.

In this discussion we assume a precise position: the interactional and communicational aspects of the surface of a MSS are of paramount importance for the success of the system itself.

Two important features characterizing the design process of VIS with respect to the traditional software design derive from this position.

First, the MSS design has to start from the design of the surface aspects of the system, rather than from the computational ones. The design must explicitly take into account the dynamics of time-space organisation of the surface as well as the constraints imposed by the current technologies. The design of these surface aspects determines a set of constraints and functionalities to be satisfied by the computational engine, which thereafter influence the design and implementation of the engine itself.

Second, the implementation must explicitly take into account the management of the surface characteristics with respect to the user profile. Evaluation is no longer limited to the algorithmic and computational aspects of the MSS, but must also explicitly take into account its usability. The use of the MSS cannot be considered as the final stage of its life cycle, but rather as a source of suggestions for new applications and improvements of the interaction technologies and the MSS itself.

Traditional models of software life cycles (e.g. the *waterfall* model), being originally designed for the development of batch systems, underestimate interactional and usability aspects of the development process. More recently some models have been proposed which stress MSS evaluation of from the user point of view,

e.g. the *star* model [2]. These models are aimed to improve the usability of interactive systems, but underestimate the influence of currently available technologies for interaction and interactive computation on the MSS development and of MSS use on technology development.

Our approach to MSS design and development is user-centred and aimed at allowing users to control the activities performed through the MSS. To this end, MSS design starts from the generalisation and formalisation of the notations adopted by the experts to communicate and reason on ideas, procedures, results and data in their traditional working environment. We adopt these users' generalized and formalized notations as the core notation of the messages on which the human-computer interaction is based and by which the users' problems are expressed and resolved. The design and implementation then develop performing the concurrent verification and validation of the conceptual and physical tools being developed.

In the paper we introduce our approach to MSS development focusing on how to maintain consistency between the users' notations and the notation adopted for human-MSS interaction. We restrict the discussion to cases in which the interaction is based on visual messages displayed on a screen. We call a system which bases its interaction with humans on these visual messages a Visual Interactive System (VIS). We discuss how a VIS is implemented from scratch when a set of interaction and interactive computation technologies are available.

Moreover, due to lack of space, we restrict the discussion only to the case in which a VIS is designed and implemented from scratch. Often this is not the case but the design starts with some components already available. In this latter case, the life cycle must be reorganised according to the available elements and knowledge.

## 2. A view on visual interaction

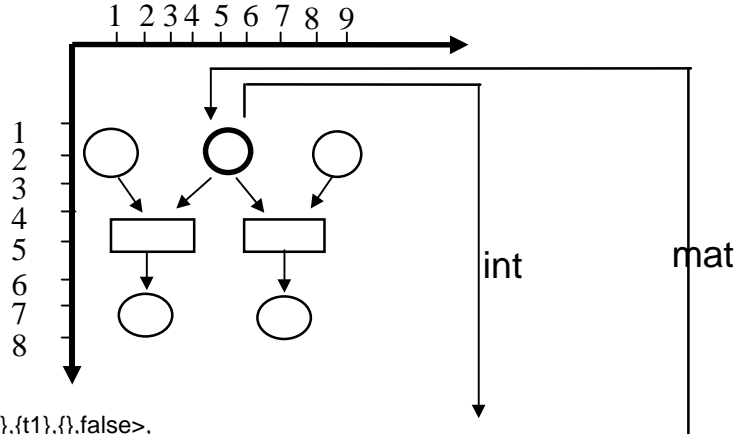
Our approach to MSS design and development focuses on supporting the MSS users, expert in some field but not necessarily in computer science, in performing some task of which they are responsible. To understand how to support users in their performances, we start by analysing their activity before MSS introduction.

In general, experts develop some specialised notation to describe and document the activities they have to perform in accomplishing a task, as well as the intermediate and final results of these activities. They use this notation to build documents in which texts, pictures and graphs are organised and assume a meaning according to the conventions adopted by the experts. The documents appear to the users as collections of *characteristic structures* (**cs**) which constitute functional or perceptual units for an expert reading or composing a document. In a text, characters of different fonts are **css**, as well as the words formed by them. In building sketches, graphs, and pictures, **css** are often specific to a discipline [3].

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*.

We call the association of a **cs** and its description a *characteristic pattern* (**cp**). An expert, having to communicate some concepts, composes the document by assembling simpler **cps** into more complex ones following the conventions established by experts so that the document itself becomes a complex **cp**. However, a different expert looking at the document sees a complex **cs**, which can be correctly interpreted by any reader knowing the set of characteristic patterns defined by the experts and the conventions to compose them.

When implementing a VIS to support a community of experts in the performance of their tasks, we assume the experts' notation as a basis to develop the language for human-MSS interaction. Here we restrict to the case in which commands are given through a video screen, and we focus on the VIS implementation. Images on the screen are therefore electronic documents built in a generalised and formalised users' notation.



$d = \langle \text{place}, p1, (1,2), \{r1\}, \{\}, \{t1\}, \{\}, \text{false} \rangle,$

**$\langle \text{place}, p2, (5,2), \{r2, r3\}, \{\}, \{t1, t2\}, \{\}, \text{false} \rangle,$**

$\langle \text{place}, p3, (9,2), \{r4\}, \{\}, \{t2\}, \{\}, \text{false} \rangle,$   $\langle \text{place}, p4, (2,7), \{\}, \{r5\}, \{\}, \{t1\}, \text{false} \rangle,$   $\langle \text{place}, p5, (7,7), \{\}, \{r6\}, \{\}, \{t2\}, \text{false} \rangle,$   
 $\langle \text{trans}, t1, (3,4), \{r1, r2\}, \{r5\}, \{p1, p2\}, \{p4\} \rangle,$   $\langle \text{trans}, t2, (7,4), \{r3, r4\}, \{r6\}, \{p2, p3\}, \{p5\} \rangle,$

$\langle \text{rel}, r1, (1,3)(3,4), \text{pt}, (p1, t1) \rangle,$   $\langle \text{rel}, r2, (5,3)(3,4), \text{pt}, (p2, t1) \rangle,$   $\langle \text{rel}, r3, (5,3)(7,4), \text{pt}, (p2, t2) \rangle,$

$\langle \text{rel}, r4, (9,3)(7,4), \text{pt}, (p3, t2) \rangle,$   $\langle \text{rel}, r5, (2,5)(2,6), \text{tp}, (t1, p4) \rangle,$   $\langle \text{rel}, r6, (7,5)(7,6), \text{tp}, (t2, p5) \rangle$

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

On the screen, the **css** of the notation are represented by sets of pixels. In our approach, the description of a **cs** is organised as an *attributed symbol* - a tuple which lists the type name and the values assumed by the attributes for the specific instance being described.

To allow the VIS to properly manage the electronic documents, the description of the relation between a **cs** and its description is made explicit by two functions: the first, **int**, mapping the structure into the attributed symbol and the second, **mat**, mapping the symbol into its structure.

On the whole, in a VIS a **cp** is described by a triple:  $cp = (\langle \text{set of pixels} \rangle, \langle \text{attributed symbol} \rangle, \langle \text{int}, \text{mat} \rangle)$ . 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.

The image *i* of a document is formed by many **css**, which must be interpreted to understand the image meaning. The definition of visual sentence (**vs**) gives account of this fact: a **vs** is a triple  $vs = \langle i, d, \langle \text{int}, \text{mat} \rangle \rangle$  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.

These definitions of **cp**, **vs** and **VL** can be used to formally characterize a generic set of documents and not only electronic documents used in VIS interaction. For example, Fig.1 schematizes a Petri Net as a **vs** in the proposed formalism. Here the **css** are the circle, the box and the arrow; they become **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:  $\langle \text{type identifier}, \text{instance identifier}, \text{coordinates of the } \text{cs} \text{ typical points}, \text{sets of } \text{cps} \text{ with which a relationship exists}, \text{and (optionally) state of the } \text{cp} \rangle$ . Note that 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. Due to lack of space, only a pair of **int** and **mat** functions are shown.

### 3.Task Visual Language and Interaction Visual Language

Two types of **VL** are of particular interest in the VIS life cycle: the Task Visual Language (**TVL**) and the Interaction Visual Language (**IVL**). **TVL** is the set of documents (**vss**) employed by the users to document their activities and communicate with each other. The **IVL** is the set of **vss** which allow user-VIS interaction. From a certain point of view, **TVL** can be considered a starting point of the design of a VIS and **IVL** the target of the design.

Let us for example observe an expert describing a system as a Petri Net. S/he expresses his/her models as visual sentences in the visual language **PN** in which the set of images of interest are representations of Petri Nets, the set of descriptions contains sets of attributed symbols describing the functional meaning of the net and of its components, and for each image a pair  $\langle \text{int}, \text{mat} \rangle$  is defined which establishes the correspondence between its

css and the attributed symbols in the description. Hence, a **vs** in **PN** is the association of a diagram representing a Petri Net with a description specifying its meaning as sketched in Fig.1.

Any visual sentence in **PN** can be generated from a 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).

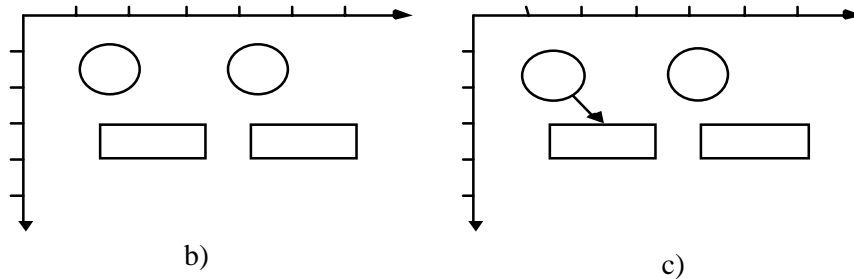


Fig.2. Two hand-drawn images representing the construction of the relation of a place and a transition in modeling of a system by PN. They are visual sentences in **TVL**.

An expert builds a Petri Net in several steps, following a personal line of thought and using the **cps** in **PN\_K**. Fig.2 shows two images, produced in this process, which can be seen as a **vss** associating a meaning with each structure in it, 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. When we speak of a set of documents produced by the expert in performing an activity, we also speak of the documents which express intermediate results or describe a line of reasoning which in a future stage of the process will be abandoned. They all belong to the Task Visual Language.

The **VIS** should allow users to build a final document following their line of thought and enjoying the new facilities and capabilities of the system. In the **PN** case, for example, users should be able to build progressively their models and check by simulation the behaviour of the proposed net with respect to the behaviour of the modeled system.

Let us now study the interaction of an expert with a **VIS** when the expert exploits the **VIS** to describe a system as a Petri Net.

The screen must show to the expert the document under development - a **vs** in **TVL**-and the set of icons, words and widgets which represent the set of legal actions which can be performed to build the document and understand the computation going on: typically add a new **cp** from the alphabet, delete a **cp** present in the document, scroll the document, enlarge a part of it etc. We call this set of icons, words and widgets a '*frame*'.

The **css** of the frame surround or are superimposed on the image components of the visual sentences of the **TVL**. The frame and the 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 screen displays observed during an interaction by which an expert uses a **VIS** to establish the same relation as in Fig.2. In each screen display the frame is composed by the icons of four buttons and by the hand-shaped pointer.

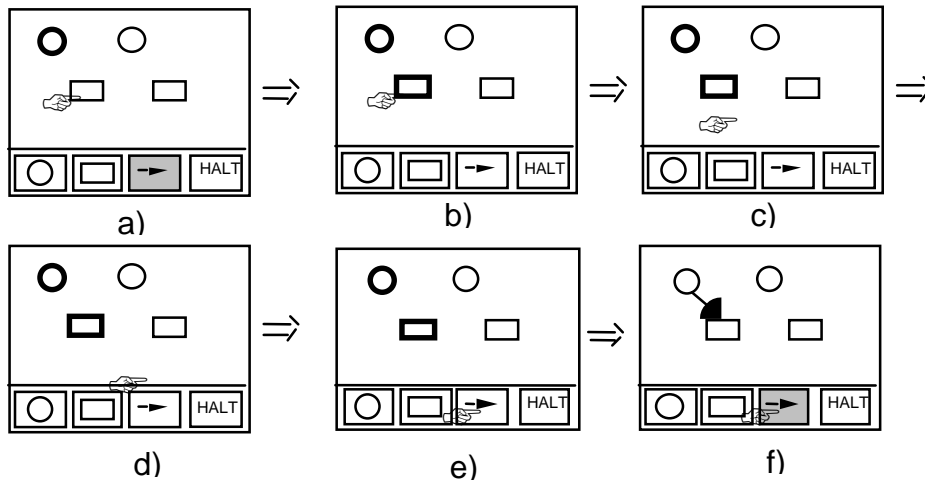


Fig.3. A sequence in IVL describing the gestures by which the action of selecting an arrow button after the selection of a place and a transition is realized

Each screen display is generated by a user gesture and consequent system reaction. 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 is the echo check of the system highlighting the transition structure as effect of the user clicking the mouse when the pointer is above it; 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 towards the arrow button; Fig.3f is the echo check of the system generating an instance of relation between the selected place and transition and disabling the arrow button as effects of the user clicking the mouse when the pointer is above the arrow button. They all represent the image part of *vss* in the *IVL* for Petri Net modeling.

To define a *VIS* we have to give account of the user operations which correspond to the creation of a *cs* in a document in *TVL* and of the gestures by which each operation is performed with the actual *VIS*.

We call *Dialog Operation (DO)* the triple <document state, user action, system reaction>.

A *DO* describes which actions a user has to perform in a given situation to obtain a desired result. For example, the tracing of an arrow in building a Petri Net is a particular instance of the *establish-relation* operation, described by a dialog operation of the type: <the potential arguments of the relation are selected, select the relation trigger, establish the relation>.

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

User actions can be defined at the low *gesture level* (simply *gesture* in the following), when one takes into account the elementary user actions on a specific tool, for example *move\_mouse*, *strike\_key*, etc. *High-level actions* (simply *actions* in the following) are synthesised from sequences of one or more gestures, which result in the achievement of a sub-goal in the execution of the task and are named, 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 step by step how to capture an input event, which function to compute in consequence, how to determine the output. At a more abstract level, the system reaction is described as a set of post conditions on the active *vs*. These descriptions are coupled into *dialog acts (da)* for each abstraction level.

A *Dialog Act (DA)* is a pair <action description, reaction description>.

A dialogue operation is an instance of dialogue act executed by the human interpreting the action description 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 dialogue operations.

The success in the use of a *VIS* depends on the properties of *IVL* and of the available set of dialog acts. *IVL* and this set determine a) the set of images which appear on the screen as the results of any user action or *VIS* computation; b) the set of executable user actions and *VIS* computation which can be performed at each step of the interaction; c) the set of gestures which the user performs to realize an action. The life cycle here proposed is aimed at designing and implementing *VISs* which support their users in performing a task, allowing them to follow their own familiar strategies. To reach this result, *VISs* allow their users to perform their strategies by sequences of gestures which result natural to them and do not distract them from the task execution.

#### 4. Formal tools for *VIS* specification

We have recently proposed visual formalisms [4, 9] which allow systems to offer this quality of interaction. A family of rewriting systems, the *Visual Conditional Attributed Rewriting System (vCARWs)*, were introduced to

specify a VL through a set of rules defined on an alphabet of **cps** so that the set of **vss** of IVL can be defined to specify the strategies to be performed as sequences of actions familiar to the users. Rules give account of the simultaneous definition of the pictorial, description and the relational parts of a **vs** in a **VL**. They can be used to define sets of active sentences -as required by the specification of **IVLs**- if attributed symbols in the **vs** description admit functions as values of some attribute and an interpreter capable of interpreting a description as a program is available. The rules in a vCARW specifying a diagrammatic language traditionally employed by the users to denote their concepts and constructs can be obtained in a systematic way from a definition in logical terms of the mutual constraints among **cps** [12].

However, vCARWs do not give account of how the interaction process organizes the **vss** of **IVL** into sequences. Actually interaction always starts from a same active **vs**: the *initial vs* whose image part appears on the screen when the system is switched on. Every other **vs** becomes active in reaction to some user interaction. The set of **vss** of an **IVL** can be organised into sequences, each being the trace of an interaction strategy applied by the expert starting from the initial **vs**. In every sequence, a **vs**(n) is generated by **vs**(n-1) by adding, deleting or transforming one or more structures to the image  $i(n-1)$ , the description of these structures to the description  $d$  of **vs**(n-1) and modifying **int** and **mat** accordingly. A pre-order is imposed on the **VL** by this grouping of the **vss** in sequences. We call dynamic visual language (**DVL**) a **VL** in which these sequences of **vss** can be identified [5]. The generation process of these sequences can be described by **gevCARW**, a family of rewriting systems which generalise the parallel pure rewriting approach [6] to the description of system dynamics, by introducing a notion of action to select the rule to be applied. A **DVL** is thus specified by a pair  $\langle \text{initial vs, gevCARW} \rangle$ .

Note that both **TVL** and **IVL** can be seen as **DVL** and described by such a pair. However this pair allows the generation of every legal sequence.

We need a tool specifying in every situation which are the legal actions, what happens when one legal action is chosen, what happens when the user tries to execute illegal actions, or when s/he decides to suspend or to conclude the interactive session. To this end a second formalism, that of *Interaction Control Automata* (ICA), is introduced to specify the system control which prevents the users from performing incorrect operations as well as the sequences of gestures by which the user performs each action. An ICA is a Conditioned Transition Network, an extension of Augmented Transition Networks [10]. ICAs are designed to control interactions and specify interpreters of interactions based on high level or low level actions.

To define ICA, let us observe that the set of actions  $U$  that the user can perform through the surface is finite. In a real interactive environment, when the current document is in a particular state  $s$  1) a finite set of user actions -denoted by a finite set of names  $U_s$ - are admitted; 2) a finite set of algorithms of legal system reactions -denoted by finite set of names  $C_s$ - are available; 3) user action names and algorithms are coupled into a finite set of dialogue acts, available for human-machine interaction interactions; 4) any action in  $U \cup U_s$ , if performed by the user, must be trapped and signaled. In order for the interaction to be correct, this coupling must be univocal for each state, i.e. when the system is in a given state, a given user action triggers a well specified system reaction.

A state in ICA reflects the current state of the interaction recording which **cps** are currently selected, and which user actions are consequently enabled when the user is interacting with the active **vs**. By performing a gesture, the user determines a system reaction and a transition to a new state. Hence, each active **vs** is generated by a user action and a consequent system reaction. For example Fig.3b is generated by a user click and the system reaction of highlighting the transition. Note that the two **vss** belong to a same sequence in IVL.

**Vss** in **IVL**, which are characterized by the same set of dialog acts, have a similar behavior when they are the active **vs**, the set of possible legal or illegal human action is the same and fires system reactions of the same type. IVL can therefore be partitioned into a finite set of subsets each one containing **vss** characterized by the same set of performable dialogue acts.

Each of these sets is associated with a state of ICA. In other words, each ICA state identifies a situation in which a finite set of dialogue acts can be performed. The situation identified by the state occurs when a **vs** is active in which all and only the dialogue acts in the set can be performed. A state  $s$  in an ICA identifies a set of **vss**, a class in the IVL partition, called  $VL(s)$ .

Each **vs** in  $VL(s)$  is characterized by:

- the set of available actions;
- an initial **vs** from which it can be generated through a finite sequence of dialog acts.

$TVL(s)$  denotes the set of task visual sentences which are sub-**vss** of the **vss** in  $VL(s)$ , i.e. each **vs** of  $TVL(s)$  is obtained from a **vs** of  $VL(s)$  by eliminating the frame.

An ICA is described by the 4-tuple  $ICA = \langle S, D_a, f, \eta \rangle$  where  $S$  is a finite set of states,  $D_a$  is a set of couples  $\langle \text{dialog act, condition} \rangle$ ,  $f: S \times D_a \rightarrow S$  is the next state function,  $\eta: D_a \rightarrow C_a$  is the output function.

The interpretation of ICA is the following: if ICA is in state  $s$  in  $S$ , a **vs**<sub>1</sub> in  $VL(s)$  is the active **vs**. Let the user make a legal action  $a$ , i.e. an action associated to dialogue act  $d_a$  in the set  $sda$  of that state.  $d_a$  indexes the couple  $\langle \langle a, al \rangle, co \rangle$ ,  $al$  being the name of an algorithm in  $C_a$  and  $co$  a predicate on some explicit variable in the description component  $d$  of the active **vs**. The system verifies if  $co$  holds: If  $co$  holds, ICA change states

moving to  $s1 = f(s, d_a)$  and the system executes  $al$ . The active  $vs$  becomes that  $vs1$  in  $VL(s1)$ , which is generated by  $vs1$  as effect of the execution of  $al$  on  $vs$ . If  $CO$  does not hold an error occurs.

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), that are the high-level ICA managing high level user actions;
- Form of Interaction (FoI) Automata, that is a family of medium-level ICA which describe how human and machine can perform the set of high level dialog operation through a sequence of low-level dialogue operations, i.e. when the human action is performed as a set of gestures on a given interaction tool;
- Gesture Control Automata (GCA), that are the low-level ICA which specify the sequence of gestures the users must follow with the available tools in order to realized the actions.

### 5. An overview of the VIS life cycle

This section offers an overview the life cycle of a VIS when it is implemented from scratch and a set of interaction and interactive computation technologies is available. A more detailed description of some of the activities to be performed will be introduced in the net sections.

The position we assume- that the interactional and communicational aspects of the surface of a VIS are of paramount importance for the success of the VIS- has two important consequences for the design process of VIS with respect to the traditional software design.

First the VIS design starts from the design of the surface aspects of the system, rather than from the computational ones. The design of the surface aspects determines a set of constraints and functionalities to be satisfied by the computational engine, which thereafter influence the design and implementation of the engine itself.

### LIFE CYCLE OF A VISUAL INTERACTIVE SYSTEM

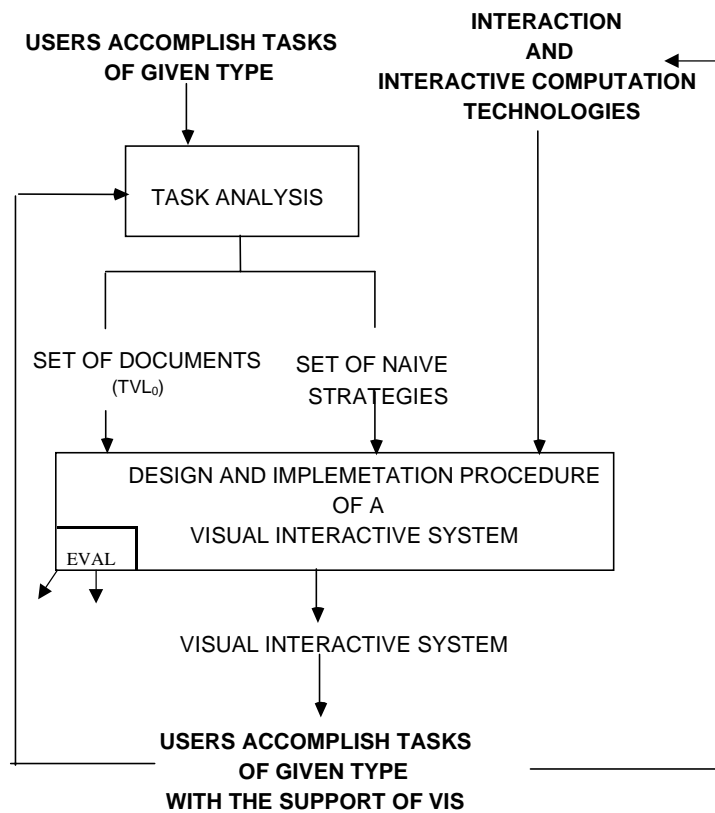


Fig.4. Life cycle of a Visual Interactive System when designed from scratch

Second, at the end of each activity its results are 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. The evaluation activity is especially signaled by the “Eval” box inside each box activity. Note that this is in some way a recognition of the centrality of the evaluation activity as emphasized in [2].

The feedback paths to be followed when an evaluation activity fails are not made explicit in the following figures, with the exception of the two, particularly notable and often underestimated, feedback paths in Fig.4.

The first one accounts for the common fact that 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 to perform a task and diffuse the results. For all these reasons, the life cycle restarts from an analysis of the new way of performing a task.

The second feedback accounts for a second process induced by the VIS use: experts' experience often suggests how to improve or modify the current technology and determines the acceptance or rejection of new technologies -think for example to the mouse, track-ball, touch pad evolution.

Fig.4 shows an overview of the VIS life cycle. The first step is the Task Analysis, which consists of the systematic observation of the users while they are accomplishing the task before the introduction of the new tools. Its goal is twofold:

- a. to collect the set of documents which expert uses to document the task execution and results and to communicate them;
- b. to describe the set of sequences of actions an expert performs to execute a task.

These sequences are called *naïve* because they are informally described and explained by the users.

Figs. 1 and 2 shows examples of documents observed in the PN case. The two images of Fig.2 belong to a same strategy and show the results of the expert action of establishing a relation between a place and a transition by drawing an arrow between the two. A naïve strategy derived from this observation states that a relation can be established if some places and transitions have been traced on the document (the document must be in a suitable state), a place and a transition are chosen and an arrow is traced linking the two structures.

The sets of documents and of naïve strategies, together with a set of descriptions of the available technologies for managing interaction and interactive computation, are the inputs to the VIS Design and Implementation activity whose output is the Visual Interactive System able to support an expert in performing a task.

## **6. The Design and Implementation of a VIS**

Fig.5 details the design and implementation activity of a VIS. In a first stage, the results of the task analysis and the descriptions of managing technologies are translated into a body of coherent formal specifications, which prescribe a) how the surface (the image on the screen) should appear before and after every user action; b) the VIS reaction to each user action; c) how each user action could be realized as a sequence of input gestures.

## DESIGN AND IMPLEMENTATION PROCEDURE OF A VISUAL INTERACTIVE SYSTEM

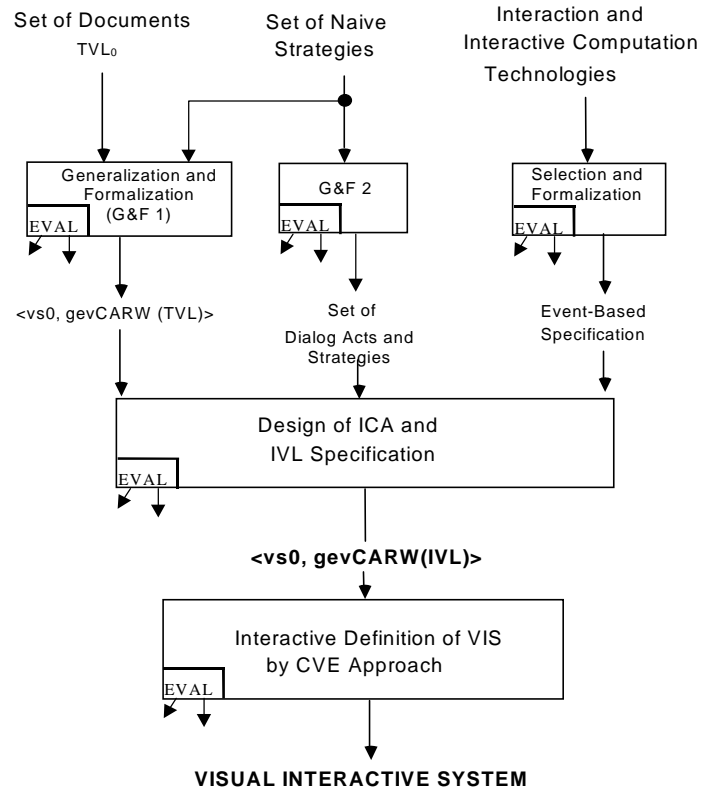


Fig.5. VIS design and implementation

By a first activity, called the GENERALIZATION AND FORMALIZATION 1, the sequences of documents and of naive rules by which documents are generated which are legal are defined by specifying a  $gevCARW(TVL)$  which defines the *Task Visual Language (TVL)*. The  $gevCARW(TVL)$  is obtained by generalization and formalization from the sets of observed documents and naive strategies. In this way the sequences of legal documents-i.e., images and their meanings- which describe the task execution can be univocally derived.

Usability evaluation is carried out by verifying a) that the set of the observed documents can be generated by the  $gevCARW(TVL)$  and b) that samples of documents which can be generated by the  $gevCARW(TVL)$  but were not observed in the real task execution, are meaningful.

In a second activity, called GENERALIZATION AND FORMALIZATION 2, the set of naive rules is formalised and generalised to a *Set of Dialog Acts(SDA)*, i.e. a set of pairs  $\langle$ user action description, system reaction description $\rangle$ . This activity usability evaluation requires that the dialogue acts are mimicked in realistic situations to verify that each sequence of dialogue operations implementing the dialogue act is *natural* for the user as well as the defined feedback.

In a third activity, called SELECTION AND FORMALIZATION, technologies for interaction and interactive computation 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 usability 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 inputs to the Interactive Specification of Control Automata and Interaction Visual Languages, which will be commented in the next section. This phase produces the pair:  $\langle$ initial visual sentence, formal expression of the Interaction Visual Language (via a  $vCARW$ ) $\rangle$

This pair completely specifies the visual interaction, both from the point of view of the surface evaluation and of the computation processes. Given a library of programs, which performs the required computation, a VIS can be implemented following the CVE approach [3] by the last step of the life cycle.

This step (last box INTERACTIVE DEFINITION BY CVE APPROACH in fig.5) consists in defining the visual interactive system. In our experience we carry out this activity by following the Cooperative Visual Environment (CVE) approach [11], in which tools for the definition of user customised visual languages are provided that allow users to define the elements of their own visual lexicons, as well as the rules for their

composition into visual programs. Users define the structures which represent the lexicon element on the display by associating an icon or an image to the intended computational meaning.

## 7 ICA Design and IVL specification

In this section we comment on the activity of ICA Design and IVL specification, sketched in Fig.6, highlighting the specific characteristics of a visual system design.

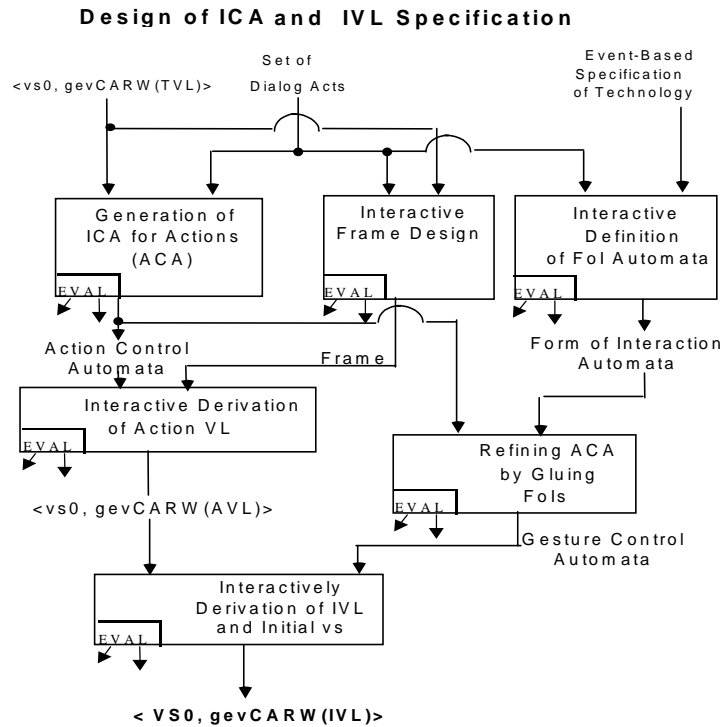


Fig.6. Interactive specification of ICA and IVL

By this activity, the designer produces the formal specification of the Interaction Visual Language (**IVL**) and of the initial visual sentence **vs0**. **IVL** is the set of all the active **vss** which can be generated during the interaction with the **VIS**. **vs0** is the visual sentence which becomes active when the **VIS** is switched on.

As a first activity the **ACTION CONTROL AUTOMATA (ACA)** is defined. **ACA** is the high-level interaction control automaton which manage the user high level actions. **ACA** should guarantee that the **VIS** will always run under the complete control of the user and will prevent users from performing incorrect operations. **ACA** transitions occur when the user performs an action -i.e. **ACA** specifies the control at a high level of abstraction, without taking into account by which gesture and through which technology the user performs the action. An algorithm for **ACA** derivation from a specific form of **gevCARW(TVL)** and a set of dialog acts is described in [9]. **ACA** usability is evaluated by paper mock-up.

The specification of a high level visual language by a **vs0** and a **gevCARW** describing the legal sequences of **vss** in the interaction, requires however the specification of the physical layout of the **vss** themselves.

To this end, the **INTERACTIVE FRAME DESIGN** activity leads to the definition of the frame, constituted by the set of **css** (icons, words and widgets), which provide the user with the context in which a user action or a computer activity is performed. The designer establishes: a) the **css** by which actions are denoted; b) the layout of the **vs**, establishing the area where the **TVL css** are shown ; c) the frame layout; d) visual feedback to user gestures; e) the visual feedback to concurrent computations (for example e-mail warning messages); f) the visual cornerstones.

Frame usability is evaluated by verifying with the users 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 the user is foreseen for each user action in order not to get him/her disoriented [13]

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 the user performs each action through the available interaction tools.

The next activity **INTERACTIVE DERIVATION OF ACTION** has the purpose to produce a **VL** which satisfies the requirement of prevention of illegal user actions, and whose **vss** have image parts which fit into the established

frame. ACA and the frame allow the derivation of the Action **VL (AVL)**, formally expressed by a *gevCARW*, and the first **vs** 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 dialog operation, 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 **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.

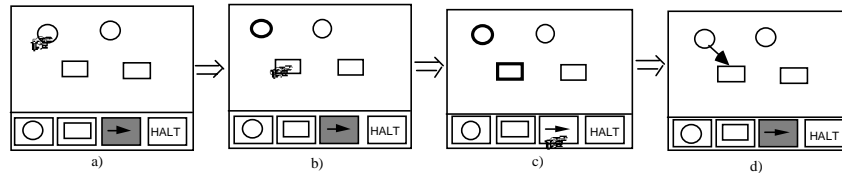


Fig.7 Four sentences in AVL describing the three user actions in the construction of the relation of a place and a transition as in fig. 2. Fig.3 shows the gestures required to implement the actions from b to d.

The usability evaluation of this activity is carried out by using a mock-up to simulate the user task execution as a sequence of high level actions and verifying that all the legal user actions are correctly managed, the illegal user actions are trapped and do not produce situations out of user's control and the users always understand the situation in which they are working.

In order to deal with the user gestures (low level actions) the next activity has the purpose to satisfy the control requirement by considering the available interactions tools. At this purpose the ACA is refined so that a final *Gesture Control Automata*, *GCA* is obtained. *GCA* describes the how to control the interaction at the abstraction level of low level gestures: it specifies the sequence of gestures the users must follow with the available tools in order to realized the actions. The refinement process is carried out by *gluing* ACA on FoIs. We say that an ICA is glued on another one by unifying some of its states with corresponding states in the second one. This is possible if each state which is glued on another state **S** does not restrict the **VL(S)**.

Finally **IVL**, expressed as the pair  $\langle vs_0, gevCARW \rangle$ , is derived by the activity **IVL AND INITIAL VS DERIVATION**. Some **vss** in **IVL** are also **vss** in **AVL**. These **vss** correspond to situation created at the end of an action execution. For example fig. 3a describes the reaction to the last gesture in the execution of the action of selection of a transition after the selection of a place. Figs. 3b to 3f describe the system reactions to the gesture executed to perform the action of selecting the arrow button by moving the mouse (and hence causing the movement of the pointer) and clicking the mouse (hence causing the button highlighting and the relation generation).

## 8. Conclusion

This paper introduces a view on the design of the interfaces of a MSS in the case that communication occurs through a screen and on the implementation of the VIS using it. The approach stems from experiences in the field of medical applications [3,7] and interactive image interpretation [8]. It is based on model of **VL** for interaction introduced in [4] and the use of *gevCARW* which allow the syntactic definition of the image part and of computational meaning of the messages exchanged in the interaction[5]. Its generalisation to other modes of interaction requires the adequate formalization of the set of sign used to form messages exchanged between humans and machines so that their temporal and spatial properties can be precisely described.

## References

- [1] Brancheau J.C., Brown C.V., The management of end-user computing: status and direction, *ACM Computing Surveys*, 25, 437-482, 1993.
- [2] Hix D., Hartson H.R., *Developing User Interface: Ensuring Usability Through Product and Process*, John Wiley - New York, 1993.
- [3] Bianchi N., Bottoni P., Mussio P., Protti M., Cooperative Visual Environments for the Design of Effective Visual Systems, *Journal of Visual Languages and Computing*, 4, 357-382, 1993.
- [4] Bottoni P., Costabile M.F., Leviardi S., Mussio P., Defining visual languages for interactive computing, *IEEE Trans. on System, Man and Cybernetics*, 27, 773-783.
- [5] Bottoni P., Chang S.K., Costabile M.F., Leviardi S., Mussio P., On the Specification of Dynamic Visual Languages, *Proc. IEEE Symposium on Visual Languages'98*, 1998, 14-21.
- [6] Dassow J., Păun G., *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, 1989.
- [7] Mussio P., Pietrogrande M., Protti M., Simulation of hepatological models: a study in visual interactive exploration of scientific problems, *Journal of Visual Languages and Computing*, 2,75-95, 1991.

- [8] Levialdi S., Mussio P., Bianchi N., Bottoni P., Computing with/on images, in *Artificial Vision - Image Description, Recognition and Communication*, V.Cantoni, S.Levialdi, V.Roberto (eds.), Academic Press, 239-265, 1997.
- [9] Bottoni P., Costabile M.F., Levialdi S., Mussio P., Specifying dialog control in Visual Interactive Systems, to appear on *Journal of Visual Languages and Computing*
- [10] Green M., A survey of three dialogue models, *ACM Trans. on Graphics*, **5**(3):244-275, 1986.
- [11] 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.
- [12] Bottoni P., Parisi Presicce F., Simeoni M., From formulae to rewriting systems, *Proc. TAGT'98*, 300-307.
- [13] Preece J., Rogers Y., Sharp H., Benyon D., Holland S., Carey T., *Human-Computer Interaction*, Addison-Wesley, 1994