Syndetic Models and Gestural Interaction

Giorgio P. Faconti, Angelo Fornari
CNR - CNUCE
Via S.Maria 36, I-56126, Pisa, Italy
e-mail: G.Faconti@cnuce.cnr.it

Abstract. This paper reports on a work aiming to investigate the potential of a new approach to model human-computer interaction, called syndetic modeling. The approach aims to solve one of the crucial problems of interactive systems by explicitly introducing the concept of usability in system design and development. Syndetic modeling combines a formal expression of system behaviour with an approximate representation of human cognitive resources in a unifying framework that allows reasoning about the flow and utilization of information in the combined system. The potential of the model has been tested against an abstraction of a mouse based gesture recognition system that is presented here as an example.

1 INTRODUCTION

Recent advances in user interface development has been mainly driven by technical innovation. Improvements in technology have made possible not only to increase the power of computing systems but also to develop new physical devices, as the data glove, novel logical devices, like the rack widget [Connor92] and the cone trees [Robertson91a], and to create new paradigms for *interacting with the information*, such as the perspective wall [Robertson91b] and the already familiar WWW [Berners92]. Although these developments have addressed significant problems and a great number of sophisticated systems have been made available in different application areas, an obvious question arises: will this technology be of effective use?

The issue is not new. In 1973, Martin [Martin73] already noticed that *computer system analysts are going to become increasingly involved with the psychology of terminal users* and a few years earlier Kubrik's movie *2001: A Space Odyssey* was intriguing enough to show that the best way for humans to communicate with a computer would be in their own language. Many years later, the desktop metaphor has become widely used, direct manipulation has been claimed to be *intuitive* and graphical user interfaces have been said to be *user friendly.* Currently, the challenge is to build intuitive structures and human-centered paradigms that will liberate from dependence on the fixed concept of computer and will lead to new ways of interacting with information; ways that will be intuitive, personal, and usable by everyone.

Works [Paterno94][Bastide95] have taken place investigating models and techniques for analyzing and designing interactionally rich systems, based for example on technology such as speech and gesture. The aim has been to understand how human aspects of novel interfaces can be modelled from a variety of disciplinary perspectives, how those disparate insights can be integrated for the purpose of system development, and the applicability of the techniques in practice. Formal methods have been one of a number of approaches; others include cognitive user models, design space representations, and software architecture models.

Applications (including industrial ones) for formal methods are well known, see for example Bowen and Hinchey [Bowen95], and Gaudel [Gaudel94]. However, none of the cited applications uses formal methods to examine the user interface. One reason is that the interface is often seen as too concrete for (useful) formal description, and the concept of usability is too vague to permit useful analysis or reasoning. In fact, established use of formal methods assesses an *extended concept of usability* which doesn't take into consideration human's capabilities and limitations. It focuses on the *what* rather than on *how* something can be obtained. Moreover, there is a temptation to minimize the number of un-predictable components (as a human being is) within a system by reinforcing its safety properties. Here, the contradiction is that established use of formal methods tends to describe systems that are interactionally impoverished.

In human terms, a usable system facilitates effective communication between human and computer by matching the needs and capabilities of the user with the interface. The factors that affect usability depend on psychological and social properties of cognition and work, rather than on abstract mathematical models of programming semantics. For this reasons described above, claims made through formal methods about the usability of a system must be grounded in some psychological or social theory.

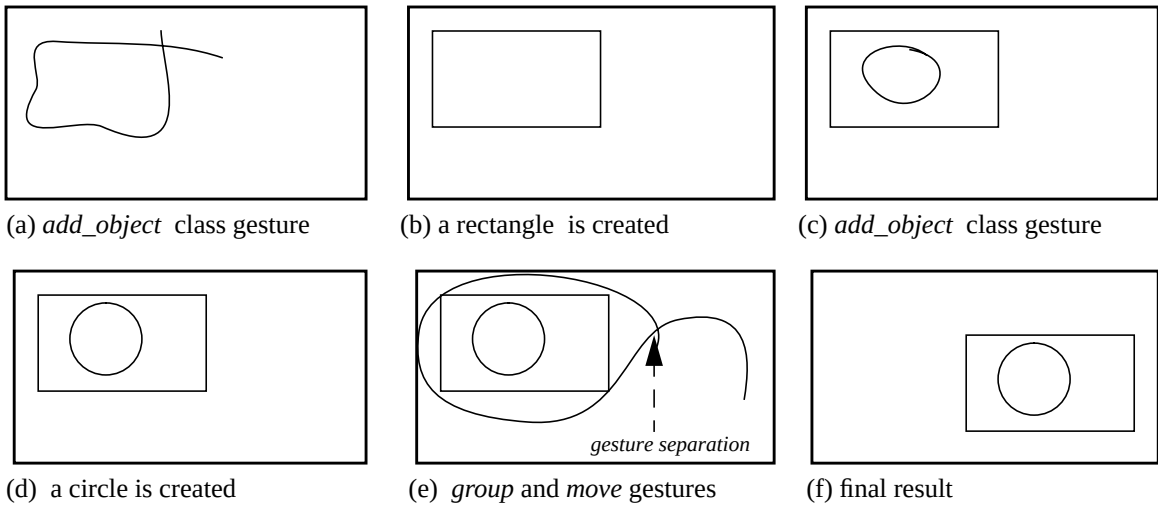
This paper builds on previous works carried within the AMODEUS project [Duke94a][Duke95] and demonstrates how a new approach to human-computer interaction, called *syndetic modeling*, can be used to gain insight into user-oriented properties of interactive systems. It uses the framework of ICS (Interactive Cognitive Subsystems) [Barnard93] to describe and analyze the human resources and the information processing required to interact with a given system. This will be shown by addressing a simple exemplar, described in section 2, that allows to enter graphical objects into the system and to manipulate them by means of gestures. Section 3 gives a short introduction and describes a formal model of ICS. In section 4 a syndetic model is developed that combines formal models of users and systems within a common framework. Finally, the syndetic model is analyzed in section 5.

2 EXEMPLAR DESCRIPTION

The example, used throughout the paper, is based on the work on gesture recognition by Rubine [Rubine91]. Rubine points out that a problem with most gesture-based systems is that an entire gesture must be entered and the interaction completed before the system responds. Such systems make it awkward to use gestures for operations that require continuous feedback. The *eager recognition* method, proposed by Rubine, supports a two phase interaction technique in which a gesture collection phase is immediately followed by a manipulation phase. This method is claimed to result in a smooth and natural interaction by allowing views to respond to gestures. We will examine the cognitive load of users when using such an interaction technique and provide insights into its strengths and possible limitations.

The example is simple enough and of a size suitable to illustrate the syndetic approach in the space available. The user specifies commands by simple drawings, typically made with a pen or mouse. The gesture is indicated before it is classified and command execution commences. During this phase, called *collection*, the points of the gesture are collected. Then the gesture is indicated, it is classified, and the *manipulation* phase is entered. The classification of the gesture determines the operation to be performed. Figure 1 show one of the possible scenarios, where two objects are created and

subsequently grouped and moved together.



(a) *add_object* class gesture (b) a rectangle is created (c) *add_object* class gesture

(d) a circle is created (e) *group* and *move* gestures (f) final result

Figure 1 Creating, grouping and moving objects

The eager recognition method allows for the gestures to be performed continuously by the user, as shown with the group and move commands mentioned in the figure. In fact, as soon as a gesture can be classified the manipulation phase is entered without user intervention; that is gestures are separated automatically by the recognizer itself.

The MAL [Ryan91] notation is used to specify and reason about aspects of the system. This choice is motivated by that it allows to describe the relationship between the internal system state and its presentation while keeping the specification compact and easily understandable. As with any *state based* specification, we begin by introducing the types of objects that are of interest. These are primitive types that represent domain concepts:

type Posn - positions in a user's gesture
Gesture - recognized gestures, i.e. add_object, manipulate
Shape - possible classification of gestures
Obj - objects in the system

Type constructors allow new types to be defined as the function space ($D \rightarrow R$), the cartesian product ($S \times S$), the power set (**P**S) and the sequence (S*). Axioms, expressed in modal action logic, contain the usual connectivity and quantifiers ($\wedge$ for and, $\Rightarrow$ for implies, $\exists$ for exists, $\forall$ for for-all, etc.). For any action A and predicate Q, the logic includes a modal predicate [A] Q, meaning that Q is required to hold after performing action A. In the specification, annotations called *percepts*, such as **vis**, are used to represent attributes that are perceivable through some human modality.

2.1 Specification of the Functional Core

According to [Salber94], the functional core is that component of a system that implements concepts identified by task analysis as relevant to the user to accomplish the tasks in a specific domain. In our case the first concept to be introduced is the one of gesture, and we model it as an agent containing an internal state representing the information relevant to gestures and a set of actions that it can be engaged in.

agent: GESTURE - GS

attributes

gestures	:	Posn* → Gesture	- the gesture corresponding to a sequence of points
history	:	Posn*	- the sequences of points performed so far

actions

form	:	Posn*	- recognizes a sequence of points formed by the user
recognize	:	Gesture	- recognizes a complete gesture

axioms

1 $\forall$ p, P $\in$ Posn* • history = H $\wedge$

1.1 (H ^ p ^ P) $\in$ dom gestures $\Rightarrow$ [form(p)] history = H ^ p

1.2 (H ^ p ^ P) $\notin$ dom gestures $\Rightarrow$ [form(p)] history = p

2 history $\in$ dom gestures $\Rightarrow$ **obl**(recognise(gestures(history)))

3 [recognize(G)] history = $\varnothing$

According to the eager recognition method, axiom 1 describes the effect of the user continuously forming a sequence of points that is kept in the history of points. If (axiom 1.1) the new entered subsequence 'p' might extend the history to a legal gesture it is concatenated to the history (the operator ^ is the sequence concatenation). Otherwise (axiom 1.2) the point cannot be used to form a gesture, the sequence so far formed is discarded and a new sequence is started. As soon as the history of formed points can be interpreted as a valid gesture, the system is forced to recognize it (axiom 2) and the history is reset as result of the recognition (axiom 3).

Having described the basic mechanism governing the eager gesture recognition, we provide an application for it by linking recognized gestures to object shapes and locations. This is done by defining a new agent that includes the recognizer and extends the internal state of the system with information describing the semantics of gestures. This information consists of a set of objects that have a position and a shape associated with them. No further actions are defined for this agent since state changes are governed by the actions of the gesture agent.

agent: OBJECT_STORE - OS
GESTURE

attributes

objects	:	**P**Obj	- the set of objects
shapes	:	Obj → Shape	- the shape of an object
location	:	Obj → Posn	- the location of an object
abstract	:	Gesture → Shape	- the shape corresponding to a gesture
update	:	Obj × Gesture → Obj	- the updating of an object

axioms

1 objects = O $\wedge$ history = H $\Rightarrow$
[recognise(*add_object*)] objects = O $\oplus$ o $\wedge$
shapes(o) = abstract(gestures(H)) $\wedge$
location(o) = last(H)

2 history = H $\Rightarrow$
[recognise(*manipulate*)] shapes(o) = shapes(update(o,gestures(H))) $\wedge$
location(o) = location(update(o,gestures(H)))

Axiom 1 describes the effect on the state after the system has recognized a gesture of class *add_object*. A new object is added to the current set of gesture (the operator ⊕ specifies object addition), its shape is abstracted from the gesture formed by the user, and its location is set to the last point of the gesture. Axiom 2 describes the effect on the state after the system has recognized a gesture of class *manipulate*. One (or more) objects is selected and its shape and location are updated accordingly to the actual gesture.

2.2 Specification of the Interaction Devices

In order to interact with the functional core, interaction devices must also be specified. We will consider a direct manipulation style of interaction made through the use of a mouse and a display as commonly available with traditional workstations. Both devices are specified as *interactors*, that is agents that provides with a perceivable presentation of their internal state [Faconti90][Duke93].

Two more primitive types are introduced that are necessary to describe the state of the interactors:

type D_Posn - positions in the display coordinate system
M_Posn - positions in the mouse coordinate system

For our purposes, the state of the display is defined simply by a set of sequences of points in the display space and by a function mapping neutral positions to display positions. The scene attribute has an associated **vis** decoration indicating that it is perceivable through the human vision modality.

interactor: DISPLAY - DS
attributes
vis scene : P(D_Posn*) - the visible set of objects
present : Posn* → D_Posn* - transforms neutral space to display space
actions
render
axioms
1 scene = D ⇒ [render] scene = D ⊗ present(dp)
2 scene@**vis** in DISPLAY@**vis**

Axiom 1 requires that the effect of the render action causes an update of the visible scene due to the merging of the previous scene with a newly generated output (the operator ⊗ indicates the merging). Axiom 2 indicates that the scene is always part of the presentation of the display. The @**vis** annotations informs that percepts are considered instead of the actual values of the attributes.

The description of the mouse simply defines a mapping from mouse space to neutral space and the operate action that is relevant to the human perception of force and timing of skeletal musculatures for the control of the arm and the hand operating the device.

interactor: MOUSE - MS
attributes
in_mouse : M_Posn* → Posn* - transforms mouse space to neutral space
actions
lim/bs operate

In direct manipulation, mouse and display are operated as a unique logical device. This is captured by the following interactor specification that includes both devices and links them together. A cursor state information is introduced representing the current position of the mouse in the neutral space. Although an actual implementation may realize it differently, the cursor is specified that way so that both display and mouse devices can refer to it without introducing additional machinery that is not necessary at the level of abstraction of the specification.

interactor: LOGICAL_DEVICE - LD
MOUSE, DISPLAY
attributes
vis cursor : Posn - position of the mouse in neutral space
axioms
1 [operate] **obl**(render)
2 [render] cursor = last(in_mouse(mp))
3 present(cursor)@**vis** in DISPLAY@**vis**

Axioms 1 and 2 indicates that when the mouse is operated, the display is forced to perform a rendering while updating the cursor position with the last entered point. The output of the cursor is always part of the presentation of the display (axiom 3).

2.3 System Integration

The integration of the functional core and of the interaction devices is described by a further interactor that includes the specification developed so far. A new attribute is required that concretizes the objects stored in the system from their shape and location.

interactor: SYSTEM - SYS
OBJECT_STORE, LOGICAL_DEVICE
attributes
concretize : (Shape × Posn) → Posn*

axioms
1 [operate] **obl**(form(in_mouse(mp)).
2 scene = D ∧ history = H ∧
2.1 H ∉ dom gestures ⇒ [render] scene = D ⊗ present(H)
2.2 H ∈ dom gestures ⇒ [render] scene = (D ® present(H)) ⊗
present(concretize(shapes(o),
location(o)))

The axioms completely describe the resulting system. Axiom 1 links the actions occurring at the device with the ones occurring at the functional core by forcing the system to recognize the sequence of points formed by the user when operating the mouse. Axiom 2 describes the changing of the scene when a rendering occurs. If (axiom 2.1) the user is forming a gesture that cannot be recognized, the history points are merged to the scene. Otherwise (axiom 2.2) the history points are removed (the operator ® indicates removal) and the concretization of a new added object or the updating of a manipulated one are merged to the scene. It should be noted that both ⊗ and ® operators represent complex functionality of graphics systems. They are under-specified here since a detailed description of the underlying operations doesn't add significant information for

the purposes of this paper.

The formal model developed in this section founds its value in that it captures salient aspects of the system while abstracting from details that are unimportant within this context. However, the properties that can be verified are properties on the system model itself, and might be used either as high level requirements, or should be derivable as consequences from the system description. The point of departure from known design methods is when the requirement is that the system should be usable. This is not just a *mechanical* property of the system, but a statement that implicitly or explicitly must embody some claim or understanding about human capabilities and limitations. In other words, in addition to being a (formally) provable consequence of the specification, the property must also be psychologically valid.

3 A COGNITIVE MODEL OF THE USER

This section describes how the interactor approach can be addressed to describe the user based on a cognitive model, thus representing expressions of user and system in a unifying framework. The state and behaviour of an interactor describing the user are derived from a representation of the Interactive Cognitive Subsystems (ICS), a model of human information processing that involves building approximate descriptions of the cognitive activity underlying task performance in human-computer interactions. This approach does not aim to simulate exactly what is going on in the user's head, but to capture the salient features of their cognitive processing. In the remainder of this section a short introduction to ICS is given for completeness. The interested reader may find a more complete description of the model in [Barnard93][May93][Barnard94].

3.1 Interactive Cognitive Subsystems

ICS represents cognitive activity as a configuration, or flow of information through different mental representations. There are nine different forms of mental representation, each of which can be operated on a particular cognitive subsystem. Although specialized to deal with specific codes, all subsystems have a common architecture, shown in Figure 2.

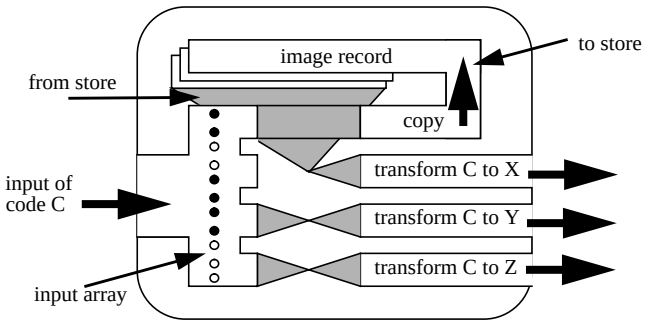


Figure 2 Common architecture of ICS subsystems

These subsystems can perform two kinds of operation upon the representations that they receive at an input array. They can copy the representation directly into the image record, which acts as a memory local to each subsystem, and they can transform the information into another mental representation and pass it through a data network to other subsystems. The transformation processes within each subsystem are independent and can work in parallel.

The representations that can be output by a subsystem are limited by the informational content of the representations that it operates upon; that is, a subsystem cannot produce output in every representation. Moreover, any one transformation process can only operate upon a single coherent data stream at one time. That is, it can only operate upon one representation, and can only produce one output representation.

If the incoming data is incomplete, a subsystem can augment it by accessing the image record. Coherent data streams may be blended at the input array of a subsystem, with the result that a process can transform data derived from multiple input sources in one step. This balances the output limitation.

The nine subsystems are further distinguished depending on their functionality as:

• Sensory subsystems

VIS visual: encodes dimensions of light such as wavelength, brightness, user visual space

AC acoustic: encodes dimensions of sound such as frequency, timbre, intensity

BS body-state: encodes dimensions such as skeletal muscle tension

• Structural subsystems

OBJ object: abstract structural description of entities and relations in visual space

MPL morpholexical: abstract structural description of entities and relations in sound space

• Meaning subsystems

PROP propositional: abstract description of entities and relations in semantic space

IMPLIC implicational: abstract description of human existential space abstracted over sensory and propositional input

• Effector subsystems

ART articulatory: encodes dimensions such as force, target and timing of articulatory musculatures

LIM limb: encodes dimensions such as force, target, position and timing of skeletal musculatures

The nine subsystems acts effectively as communicating processes running in parallel as shown in Figure 3.

The overall behaviour of the cognitive system is governed by a number of principles, most of which are out of the scope of this paper. Here, we will address only those *configurations* that are relevant to interact with the system described in the previous section. Configurations are the way in which ICS resources are deployed at a point in time to perform a cognitive task. Complex configurations can be constructed from elementary, partial ones, and if an information flow can be constructed, then it is a legal configuration, subject to three constraints. The first one is that no process can appear more than once in a configuration. The second constraints is that the order of cyclical flows within the configuration is not important. Finally, although any one of the sensors or effectors may be missing, if no sensors or effectors or both are missing in a configuration there must be a central flow. In other terms, input alone is meaningless and no output can be generated without either input or central activity.

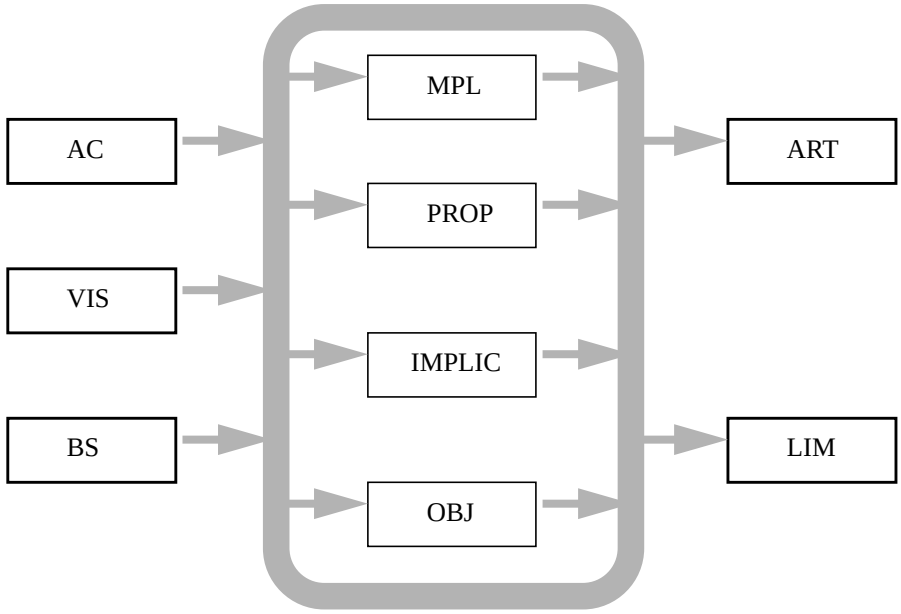
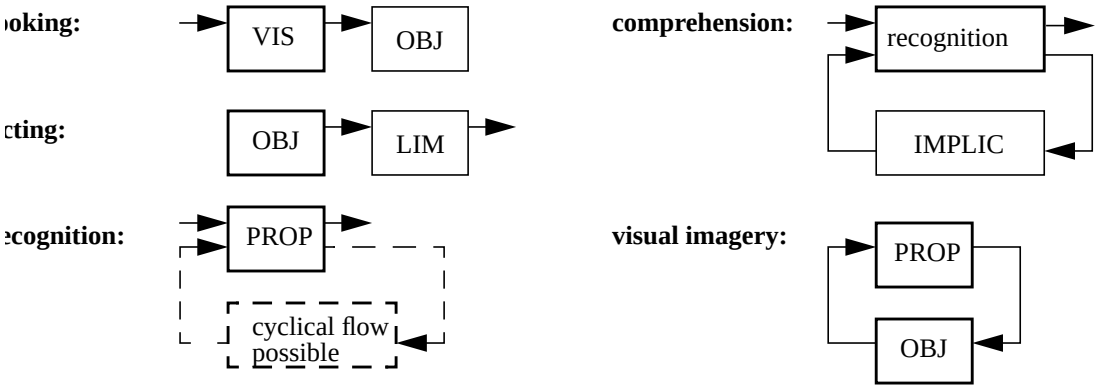


Figure 3 Overall architecture of ICS

Examples of configurations include:



Following [11], the concept of configuration can be represented formally using a recursive type definition:

Config ::=	sys-sys	- transformation from one code to another
|	Config::Config	- chaining of two configurations
|	Config_{R}	- reciprocal loop
|	Config_{BUF}	- buffered transformation
|	Config+Config	- dual processing

For example, the *looking* and *acting* while *thinking* configuration is expressed as

$$[\text{vis-obj::obj-lim}] + [\text{prop-implic::implic-prop}_{BUF}]_{R}$$

Moreover, we indicate that a configuration is part of another with

$$\_ \propto \_ : \text{Config} \times \text{Config} \rightarrow \text{bool}$$

as in

$$[\text{vis-obj::obj-lim}] \propto [\text{vis-obj::obj-lim}] + [\text{prop-implic::implic-prop}_{BUF}]_R$$

showing that the looking and acting configuration is part of the looking and acting while thinking configuration.

3.2 A Formal Representation of ICS

A simplified model of ICS is directly derived from an understanding of the framework discussed in the previous section. The notation used is the same used to describe the system component of an interactive system so that a unique specification can be subsequently built by combining the two components.

The ICS state is characterized by a configuration, a description of the currently buffered transformation in a flow, a description of the currently blended inputs, the availability of data representation at a subsystem, and the ability of a process to transform a data unit. Three actions are considered: 'transf' allows the propagation of information in a flow, 'buffer' allows the location of the buffered transformation to be transferred amongst subsystems in a configuration and 'blend' allows the combination of inputs at a subsystem.

interactor:ICS_MODEL - ICS

attributes

config	: Config	- configuration of transformation process
buffered	: sys x sys	- description of currently buffered transformation
_@_	: code x sys → bool	- data representation available at a subsystem
ability	: sys x sys x code → {high, low}	- ability of a process to transform a data unit
blended	: sys → **P**sys	- description of blending

actions

transf		- map data from one code to another
buffer	: sys x sys	- put a transformation into buffered mode
blend	: sys	- combine inputs by blending

axioms

1 p@*src* ∧ *src-dst* ∝ config ⇒ [transf] p@*dst*

2 *src-dst* ∝ config ∧ ability(*src*,*dst*,p) = low ⇒ **obl**(buffer(*src*,*dst*))

3 buffered = (*s*, *t*) ⇔ *s-t*$_{BUF}$ ∝ config

4 config = C ⇒ [buffer(*s*, *t*)] buffered = (*s*, *t*) ∧ ∀ *s*, *t* ∈ sys • *s-t*$_{BUF}$ ∝ config ⇔ *s-t*$_{BUF}$ ∝ C

5 **per**(blend(*t*)) ⇒ ∃ *s*, *t*, *u* ∈ sys • *s-t* ∝ config ∧ *u-t* ∝ config

6 ∃ *s*, *t*, *u* ∈ sys • *s-t* ∝ config ∧ *u-t* ∝ config ⇒[blend(*t*)] {*s*, *u*} ⊆ blended(*t*)

7 ∀ *s*, *t*, *u* ∈ sys • *s-t* ∝ config ∧ *u-t* ∝ config ⇒

7.1 {*s*, *u*} ⊆ blended(*t*)

∨

7.2 ∃ d_1,d_2 ∈ sys •

$d_1 \neq d_2 \Rightarrow [s\text{-}t::t\text{-}d_1] + [u\text{-}t::t\text{-}d_2] \propto$ config

∨

7.3 $d_1 = d_2 = d \Rightarrow [s\text{-}t::t\text{-}d] + [u\text{-}t::t\text{-}d_{BUF}] \propto$ config

$$\vee$$

$$[s\text{-}t::t\text{-}d_{\mathrm{BUF}}] + [u\text{-}t::t\text{-}d] \propto \text{config}$$

8 $\forall\, t \in \text{sys} \bullet$

$$t\text{-prop} \propto \text{config} \wedge \text{p@}t \wedge \neg\, \text{p@prop} \Rightarrow$$

$$[\text{transf}]\ \text{p@prop} \wedge (\text{prop-implic::implic-prop}_{\mathrm{BUF}})_{\mathrm{R}} \propto \text{config}$$

Axiom 1 indicates that the effect of the ’transf’ action is that the representation of ’p’ is transformed and transferred from source to destination. Axioms 2 to 4 refers to buffering. A transformation is forced to enter buffered mode if it is part of a configuration and it is unable to operate on the current representation, the buffered mode is linked to the value of the configuration and the location of the buffer is permitted to change while living the structure of the configuration unchanged. Axiom 5 indicates that information blending of incoming data streams to the same subsystem may take place, while axiom 6 requires that after the ’blend’ action the involved source subsystems are part of the blending description of the destination subsystem. Axiom 7 refers to the principle that a transformation process can operate only on one coherent data stream at any one time. Consequently, two data streams input to a subsystem are either blended (axiom 7.1), or refers to different transformation processes (axiom 7.2), or one of the two is buffered (axiom 7.3) determining an unstable configuration that will oscillate causing disruptive effects. Finally, axiom 8 states that any transformation in the propositional code requires that the comprehension loop becomes part of the configuration if neither description nor relation in semantic space are available for that representation at the propositional subsystem.

4 SYNDETIC MODELLING

A syndetic model of an interactive system extends the formal model of the device or interface with the model of the cognitive resources needed to interact with the device. The key feature of syndesis (as compared for example to other models of the user) is the use of a cognitive model of human information processing as its basis.

The system presentation is mapped onto the user sensors that receive and transform percepts for processing by the structural and meaning subsystems. Conversely, user actions expressed through effectors whose behaviour is driven by the internal subsystems are mapped onto the system devices for further processing by the functional core. To complete this mapping, we will indicate with ’lim-hand’ the transformation into the final user output: this will be represented by motor commands to the ’hand’ operating the mouse. Figure 4 shows this view.

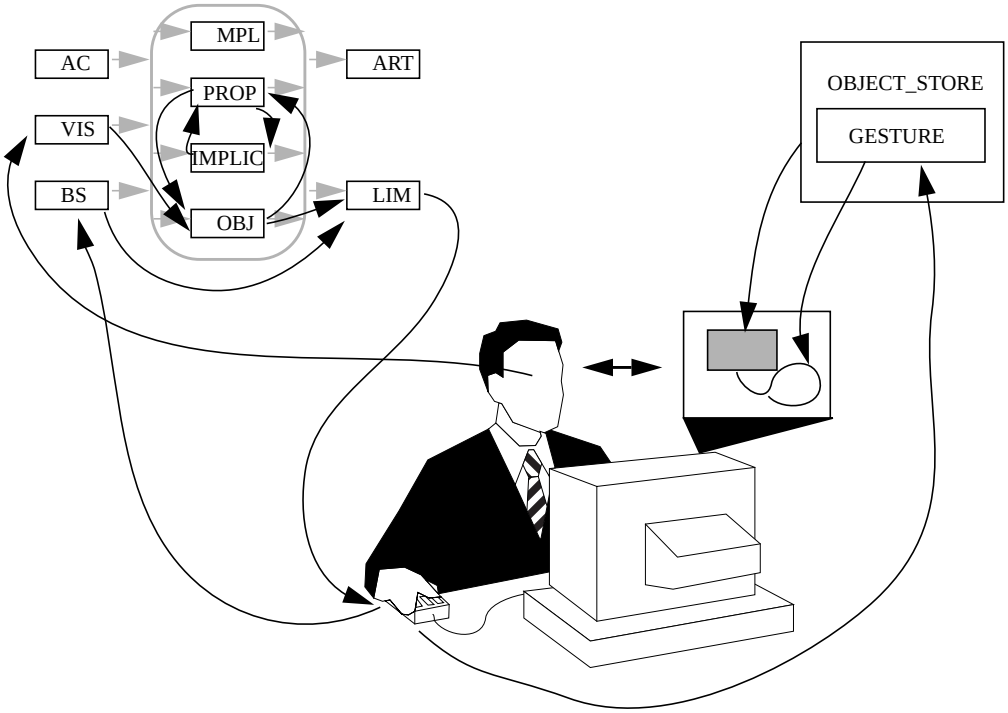


Figure 4 Syndetic modeling: user and system combined together

4.1 A Syndetic Model of the Example

The formal model of the system provides few insights into the usability of its interface as well as the formal model of ICS support general claims about the user's cognitive processes but not about the effective use of cognitive resources in a given context. By combining both of them in a syndetic model we can reason about how cognitive resources are mapped onto the functionality of the system.

The SYSTEM and the ICS_MODEL interactors developed so far are included in the definition of a new interactor and the axioms that relate the conjoint behaviour of the two components are defined. These axioms provide the generic ICS model with a concrete instance of the context in which the user is operating. The user task is to add a new object or to manipulate an already existing object in the system. The task is achieved by executing gestures formed by sequences of positions of the hand operating the mouse device. We assume that the user has knowledge of the relations occurring between objects and gestures and adopt a highly simplified representation of the user's goals in the terms of the sequence of the required positions and of the newly defined scene. Also, the user is able to interpret the trajectory followed by the hand operating the mouse from the history and the cursor position in the display space, and similarly to perceive the position of the hand in the mouse space.

interactor: SYNDETIC - SYN
SYSTEM, ICS_MODEL
attributes
goals : Posn* x Scene - the goals of the user are to form a sequence of positions in a scene
view : **P**(Posn*) → Posn* x Scene - interpretation of trajectory within a

scene
direct : Posn* → Posn x Posn* - interpretation of direction of movement
feel : Posn → Posn - interpretation of hand position

actions

exec - begin forming a sequence

axioms

1 **per**(operate) ⇒
∃ P ∈ Posn*, S ∈ Scene, p ∈ Posn •
(P,S)@prop ∧ p@bs ∧
(prop-obj::obj-lim::lim-hand)+(bs-lim::lim-hand) ∝ config

2 **per**(exec) ⇒ scene@vis ∧ vis-obj::obj-prop ∝ config

3 goals = (p ∧ P,S) ⇒ [exec] (p,S)@prop ∧ goals = (P,S)

4 ∀ o ∈ objects • o@vis ⇒ present(concretize(shapes(o), location(o)) ∈ scene

5 ∃ P ∈ Posn* • P@vis ⇒
present(P) ∈ scene ∧ history = P ∧ cursor = last(P)

6 ∃ P ∈ Posn* • P@vis ∧ scene@vis ∧ vis-obj::obj-prop ∝ config ⇒
[transf] view(scene)@prop ∧ first(view(scene))@prop = P@vis

7 ∃ P ∈ Posn* • P@vis ∧ vis-obj::obj-lim ∝ config ⇒
[transf] direct(P)@lim ∧ first(direct(P))@lim = last(P)@vis

8 p@bs ⇒ operate@**bs** in MOUSE@**bs**

9 p@bs ∧ bs-lim ∝ config ⇒ [transf] feel(p)@lim

Axioms 1 and 2 refer to the required ICS configuration. The 'operate' action, defined in the mouse interactor, is driven by the limb subsystem that receives a proprioceptive feedback from the hand as transformed by the body-state subsystem. In order to consciously operate the mouse, the representation of the trajectory to follow in a scene must be available at the propositional subsystem and the configuration be set so that it is transformed into musculature control. To start the sequence of positions the user must have a representation of the scene at the visual subsystem to be transformed into propositional code. The forming of a sequence is controlled at the propositional subsystem by the user's goals (axiom 3). The presentation of shape and position of all the objects stored in the system must be made perceivable at the visual subsystem (axiom 4) as well as the presentation of history and cursor (axiom 5). Both objects, history and cursor representation is available in propositional code after a 'transf' action (axiom 5 and 6). Similarly, history and cursor representation is also represented in limb code (axiom 7). Axioms 8 and 9 require that the body-state subsystem has a perception of the hand operating the mouse that is interpreted by the limb subsystem.

5 ANALYSIS OF THE SYNDETIC MODEL

The syndetic model given in the previous section permits to reason about interaction by considering the conjoint behaviour of user and system and to derive facts on the usability of the system in terms of the requirements for cognitive resources.

In conducting our analysis we refer to axioms with <agent_abbreviation><axiom_number>; for example, GS1 will refer to axiom 1 of the gesture interactor.

From SYN1 and SYN2 we build the *standard* gesture configuration of ICS subsystems:

$$[\text{vis-obj} :: (\text{obj-prop} :: \text{prop-obj})_R :: \text{obj-lim} :: \text{lim-hand}] + [\text{bs-lim} :: \text{lim-hand}]$$

The overall configuration satisfies all three principles stated in section 3.1. It is formed

by a primary flow from the visual to the limb subsystem including a reciprocal loop between object and propositional subsystems and from a secondary flow from the body-state to the limb subsystems.

5.1 Standard configuration

By analysis of the configuration we first derive the operating mode of the transformation processes.

- prop-obj$_{BUF}$
  Buffering is required at the propositional subsystem in order to initially construct and subsequently control the progressing the user's goal from the scene and the intended gesture representations (consciousness of user's actions).

- blended(obj)
  The object subsystem receives two input streams from the propositional and the visual subsystems. ICS5 and ICS6 indicates that blending is permitted and if a blend(obj) action occurs then {prop, vis} $\subseteq$ blended(obj). Blending at the object subsystem is highly desirable since the separation of the two flows will require buffering (ICS7.3) and its effect could be highly disruptive as described in [12]. For the blending to take place the representations derived from the user's view of the scene (SYN4) and of the history and cursor (SYN5, SYN6) must be coherent with the propositional knowledge representation about the intended hand position within the visual space perceived in terms of the cursor (LD2, LD3). This is a requirement imposed on the model by LD1 and SYS2.

- blended(lim)
  Two data streams are also present at the limb subsystem. Incoming data from the object subsystem defines the intended direction of movement, such as the next position to be acquired in a trajectory, while incoming data from the body-state system represents proprioceptive information about the current status of the musculature driving the hand movement; this has been described as the interpretation of the hand position in mouse space. The condition for the blending to take place can be expressed as

  $$p@bs \wedge P@obj \wedge \{bs,obj\} \subseteq blended(lim) \Leftrightarrow first(direct(P)@lim = feel(p)@lim$$

  A factor that inhibits the blending is derived from ICS2 for users that have a distorted perception of their hand derived from bs-lim. From experience, this will be always the case for multi-part gestures as usually it happens in hand writing. In the case that blending is inhibited at the limb subsystem, the lim-hand transformation will be necessarily buffered. Since we know from ICS3 that lim-hand and prop-obj cannot be buffered concurrently, the overall configuration will oscillate causing a sense of confusion due to the continuous transfer of the focal awareness about the representation that is being operated in buffered mode.

In conclusion we may say that in the case of a single continuous gesture performed by an expert user, the claims of Rubine about its system to result in a smooth and natural interaction are verified. The resulting configuration is:

$$[vis\text{-}obj :: (obj\text{-}prop :: prop\text{-}obj_{BUF})_R :: obj\text{-}lim :: lim\text{-}hand] + [bs\text{-}lim :: lim\text{-}hand]$$
$$\wedge$$
$$\{vis, prop\} \subseteq blended(obj) \wedge \{obj, bs\} \subseteq blended(lim)$$

In the case of a novice user or multi-part gestures it will cause confusion, user

performance and cognitive load can be improved by acquiring familiarity with the input device. Until then, the configuration will oscillate between

$$[\text{vis-obj} :: (\text{obj-prop} :: \text{prop-obj}_{BUF})_R :: \text{obj-lim} :: \text{lim-hand}] + [\text{bs-lim} :: \text{lim-hand}]$$
$$\wedge$$
$$\{\text{vis}, \text{prop}\} \subseteq \text{blended(obj)} \ \wedge \{\text{obj}, \text{bs}\} \subseteq \text{blended(lim)}$$

and

$$[\text{vis-obj} :: (\text{obj-prop} :: \text{prop-obj})_R :: \text{obj-lim} :: \text{lim-hand}_{BUF}] + [\text{bs-lim} :: \text{lim-hand}_{BUF}]$$
$$\wedge$$
$$\{\text{vis}, \text{prop}\} \subseteq \text{blended(obj)}$$

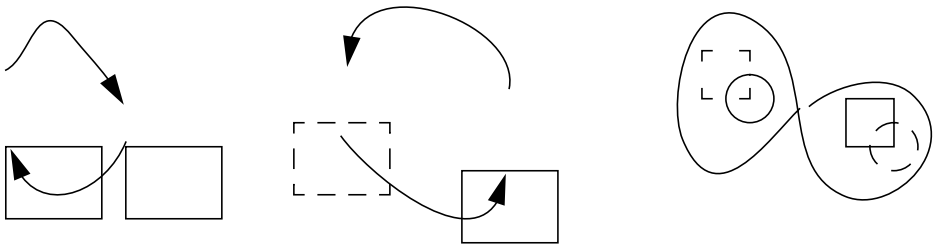
5.2 Potential problems

A potential problem is found in reaching the user's goal, if the system delays the feedback of the recognition of the gesture. We know from SYS2.1 and SYS2.2 that when a gesture is recognized the feedback of history is replaced by the result of the interpretation of the gesture so that the scene may change. In this case the user's goal will be (∅,S), and since there are no more positions to perform the configuration will change into :

(vis-obj :: obj-prop)

denoting a passive user looking at the display. A long time of waiting can be annoying.

A more serious problem arises when, due to similarity of gestures or in the case of gestures having in common a sub-trajectory, the system misinterprets the user intention. A possible scenario, derived from a gestural editor, is shown in Figure 5 where three possible gestures are shown. The 'move' and 'copy' commands are made by performing arcs, respectively clockwise and counterclockwise, while the 'swap' command is made by enclosing the objects to be swapped in an 8-shaped figure. The solid figures represent the objects in the scene after the gesture is performed and the dotted ones represent the corresponding locations before the gesture.



(a) copy (b) move (c) swap

Figure 5 Gestures sharing a common sub-trajectory

Due to the eager recognizer, a gesture is recognized as soon as there is a sufficient number of points in the history (GS2). When this happen, the history is reset (GS3) and its perceivable representation (SYS2.1) is replaced by the representation of the interpretation of the gesture (SYS2.2). Let's assume that the user intention was to swap two objects in the current scene and the corresponding goal were (p^P, SwapedScene). After the performance of 'p' the goal is (P, SwapedScene) and a propositional representation exists (p, SwapedScene)@prop from SYN3. Similarly, the representations

of the trajectory already performed and of the cursor are available at the visual subsystem (history,cursor)@vis from DS2, LD2, LD3, SYS2.1 and SYN5, and consequently it is propagated at the propositional subsystem (SYN6). This information is also blended at the object subsystem. However, the system may recognize 'p' as either a 'move' or a 'copy' gesture, depending on the direction of the movement followed in performing the 'swap'. In this case, by applying SYS2.2, the history is removed from the scene and an object appear as manipulated in a new scene after rendering. The new scene, when acquired by the visual and interpreted by the object subsystems will disable the blending at object level that will enter buffering mode causing an internal re-configuration. When the representation of the new scene is available at propositional level, there is a mismatch with the driving of the user's goal. From ICS7, there exists a possibility of starting the central engine configuration with highly disruptive effects.

6 CONCLUSIONS

With the advent of a new technology, commonly addressed as multimedia/multimodal, the challenge is to demonstrate their *usability* rather than that systems can be built out of it. The role of human cognitive abilities is becoming increasingly important in this respect. In particular, there exists a need to be able to reason about usability long before an actual system is built. Established methods in software engineering and formal methods do not offer today the necessary support for this reasoning. The approach outlined in this paper shows that syndetic modeling can be one possible way to explore to bridge this gap. The underlying combined representation of user and system adds to existing established methodologies in software development a theoretical background of cognition that can provide a basis for arguing the *why* an interface is successful, to discover potential problems and to suggest solutions.

ACKNOWLEDGEMENTS

This work was carried out as part of the ESPRIT BRA 7040 - AMODEUS project and of the HCM Interactionally Rich Systems Network both funded by the European Union. We would like to thanks Phil Barnard and Jon May whose papers and seminars on the Interactive Cognitive Subsystems have been of invaluable help. We would like also to express our gratitude to David Duke and David Duce for having spent much time in discussions and for having originally developed the idea of syndetic models.

REFERENCES

[Barnard93] Barnard P.J., May J., *Cognitive Modeling for User Requirements*, In Byerley P.F., Barnard P.J., May J. (eds.), Computers, Communication and Usability: Design Issues, Research and Methods for Integrated Services, pp. 101-145, Elsevier, 1993.

[Barnard94] Barnard P.J., May J., *Interaction with Advanced Graphical Interfaces and the Development of Latent Human Knowledge*, In Paterno' F. (ed.), Proc. Of The 1st Workshop On Design, Specification, Verification Of Interactive Systems, Springer Verlag Publisher, 1994.

[Bastide95] R.Bastide, P. Palanque (eds.), *Proc. Of The 2nd Workshop On Design, Specification, Verification Of Interactive Systems*, Springer Verlag Publisher, 1995.

[Berners92] Berners-Lee T.J., *Electronic publishing and visions of hypertext*, Phys.

world, 5 (6), 1992.

[Bowen95] Bowen J.P. , Hinchey M.J., *Applications of Formal Methods*, Prentice Hall International, 1995.

[Connor92] Connor D.B., Snibbe S.S., Herndon K.P., Robbins D.C., van Dam A., *Three dimensional widgets*, In Symp. on Interactive 3D Graphics, Special issue of ACM SIGGRAPH Computer Graphics Journal, pp. 183-188, 1992.

[Duke93] Duke, D.J., Harrison, M.D., *Abstract Interaction Objects*, Computer Graphics Forum, Vol 12(3), pp. C25 -C36, NCC/Blackwell, 1993.

[Duke94a] Duke D.J., Barnard P.J., Duce D.A., May J., *Syndetic models for human-computer interaction*, ESPRIT BRA 7040 - AMODEUS id_wp35, 1994.

[Duke94b] Duke D.J., Harrison M.D., *A Theory of Presentations*, In Proc. FME'94: Industrial Benefit of Formal Methods, Naftalin M., Denvir T., Bertran M. (eds.), Lecture Notes in Computer Science, Vol 873, pp. 271-290, Springer Verlag, 1994.

[Duke95] Duke D.J., *Reasoning About Gestural Interaction*, Computer Graphics Forum, Vol 14(3), NCC/Blackwell, 1995.

[Faconti90] Faconti G., Paterno' F., *An approach to the formal specification of the components of an interaction*, In Vandoni C., Duce D. (eds.) Proc. of Eurographics'90, pp. 481-494, North-Holland, 1990.

[Gaudel94] M.C. Gaudel, *A classification of formal methods*, In Proc. International Conference on Software Engineering, pp. 112-128, IEEE Computer Society Press, pp. 112--128, 1994.

[Martin73] Martin J., *Design of Man-Computer Dialogues*, Series in Automatic Computation, Prentice-Hall, 1973.

[May93] J., Barnard P.J., Tweedie L., *AnimICS Version 6.0B2: An animated tutorial on ICS*, Available on AMODEUS ftp server in usemod/AnimICS_6.BETA.hqx, 1993

[Paterno94] Paterno' F. (ed.), *Proc. Of The 1st Workshop On Design, Specification, Verification Of Interactive Systems*, Springer Verlag Publisher, 1994.

[Robertson91a] Robertson G.G., Mackinlay J.D., Card S.K., *Cone Trees: Animated 3D Visualizations of Hierarchical Information*, In Proc. of CHI'91: Reaching Through Technology, S.P. Robertson, G.M.Olson, J.S.Olson (eds.), pp. 189-194, 1991.

[Robertson91b] Robertson G.G., Mackinlay J.D., Card S.K., *The Perspective Wall: Detail and Context Smoothly Integrated*, In Proc. of CHI'91: Reaching Through Technology, S.P. Robertson, G.M.Olson, J.S.Olson (eds.), pp. 173-179, 1991.

[Rubine91] Rubine D., *The Automatic Recognition of Gestures*, PhD thesis, School of Computer Science, Carnegie Mellon University, 1991.

[Ryan91] Ryan M., Fiadeiro J., Maibaum T., *Sharing Actions and Attributes in Modal Action Logic*, In Ito T., Meyer A.R. (eds.), Theoretical Aspects of Comp. Software, Springer Verlag, 1991.

[Salber94] Salber D., Coutaz J., Nigay L., Faconti G., Paterno F., Duke D., Harrison M. *The System Modelling Glossary*, ESPRIT BRA 7040 - AMODEUS sm_wp26, 1994.