Visual Dynamic Environment for Distributed Systems

Vito Di Gesù, Francesco Isgrò, Biagio Lenzitti, Domenico Tegolo
*Dipartimento di Matematica ed Applicazioni, University of Palermo, ITALY

Abstract. Algorithms, based on information fusion, are often embodied in visual perception systems. Distributed architectures have been recently proposed to perform integrated computation. The complexity of distributed systems regards both their design, and the software environment to develop applications. Visual and iconic programming style intends to provide expressive tools to implement, to debug, and to execute programs in distributed environment. Multi-layers graphs languages seem suitable to handle such complexity. This paper describes the design of a visual dynamic environment (VDE), which is based on a graph-grammar. A new class of dynamic visual interfaces is also introduced, and its properties are described. The proposed VDE has been implemented on the first emulated version of the machine M-VIF (Machine Vision based on Information Fusion).

1. Introduction

Humans interact with the real environment through their senses, they react and make decisions depending on the result of such interaction. This simple consideration has suggested most of the interactive computer systems based on virtual reality [1] and multi-media [2]. Vision play a relevant role among human senses, many efforts have been done in the last decade to improve the design of visual interfaces for computer systems (CAD/CAM, windows, X-windows).

The performance of a visual system lies on the ability to focus the areas of interest, by maximizing a given costs/benefit utility criterion [3,4]. The selection of interesting regions is relevant; in fact the ability to select salient features is the basic question of intelligence, both artificial and natural. Moreover, visual perceptual systems should be able to adapt their behaviour depending on the current goal and the nature of the input data. Such performance can be obtained in systems able to interact dynamically with the environment.

Information-fusion techniques, implemented on distributed systems (DS), are suitable to develop goals-oriented strategies [5]. In fact, the computation can be driven by complementary information sources, and it may evolve on the basis of adaptive internal models and environment transformations. Moreover, results of several processing elements can be integrated to find an optimal solution.

Distributed systems are characterized by an huge amount of states and parameters, that are distributed through several elementary sub-system units. Moreover, functional dependencies exist between states and parameters. Automatic control assessment and motion detection in risky environments are examples of distributed systems. In these cases, visual data are usually collected from multi-sensors, and their elaborations are carried out on local processing units, which are logically interconnected to share and interchange knowledge (models, data and algorithms).

Visual dynamic environment musts provide a synthetic view of a distributed system behaviour, and guides to understand local and/or global computation phases. In fact, the design and the implementation of algorithms on multi-processors machine depends on the distribution of data and processes among the processing units, therefore the dynamic control is relevant to optimize and to tune the execution of processes. Dynamic visual tools allow to realize such user/machine interaction in a natural and efficient way.

This paper describes the design of a VDE, which is based on a multi-layers graph-grammar. A new class of dynamic visual interfaces is introduced, and its properties are described. The proposed VDE has been implemented on the first emulated version of the machine M-VIF [6].

The concept of dynamic icon, as it has been introduced in DIVA (Dynamic Interface for Visual Applications) [6], will be used to extend visual interfaces on DSs.

In Section 2 the definition of VDE, and related properties are given. Section 3 shortly describes the M-VIF machine. Section 4 describes the implementation of VDE to program M-VIF. An example of distributed computation using the proposed VDE is shown in session 5. Concluding remarks are given in Section 6.


Informally, a VDE is an iconic system that allows both to build an user view of an underlying DS, and to allocate and to control the related co-operating processes. Bricks of a VDE are both conventional and Dynamic Icons (DI) [7]. The semantic values of a DI depends on the evolution of the distributed processes. In order to define more
formally VDE's, it is necessary to introduce some notations and definitions about distributed systems.

**Definition 1:** A DS is a 5-tuple <S,I,O,A,W>, where S is the set of states, IcS is the set of inputs, OcS is the set of output. Transitions among states of a DS are represented by the function Δ:S→S, where Δ:S→S is a set of parts of S. W is a mask function W:S→{0,1}.

On the other hand, visual computation, based on information fusion, can be formulated in terms of five functional modules: Observe, Process, World Model, Choose Next, and Action. Sensor data are provided to the Observe module that performs early vision tasks; the information flows to the Process module, processes of which (algorithms executed on proper hardware) are selected by the Choose-next module, the processes are also driven by the World-Model. The outputs of the computation are directed both to Action, that operates on the environment, and to Observe module, that drives further sensor-explorations. The World represents the environment on which a DS operates. Within the system, information flows in a continuous active feedback loop (see Figure 1). Now, the states of DS can be defined as:

\[ S = D \cup M \cup A \cup P \]

where: D represents input/output data, collected by sensors from the world or produced as results of a given computation, M represents models (for example relations between objects, sensors characteristics, environment), P is the set of distributed processes, and A is the set of actions that modify the world (open/close a door, activate an alert system,...). Moreover, IcD is the set of input states, OcDUA is the set of output states. Here, the term process has a wide meaning, it could be a single algorithm or a sequence of processes associated to a set of processing units (PU) connected in a given topology.

The transition function is defined as follows:

\[
\begin{align*}
\Delta_1 &: \mathcal{P}(D) \to \mathcal{P}(P) & \text{data / processes} \\
\Delta_2 &: \mathcal{P}(P) \to \mathcal{P}(D) & \text{processes / data} \\
\Delta_3 &: \mathcal{P}(M) \to \mathcal{P}(P) & \text{models / processes} \\
\Delta_4 &: \mathcal{P}(P) \to \mathcal{P}(M) & \text{processes / models} \\
\Delta_5 &: \mathcal{P}(P) \to \mathcal{P}(P) & \text{processes / processes}
\end{align*}
\]

Each function, \(\Delta_i\), defines logical (or physical) links among elements of S. The computation will evolve on the basis of both transitions and mask functions. A transition is active if all its input links have mask value equal 1. When a transition \(\mathcal{P}(X) \to \mathcal{P}(Y)\) is active then information (data, model and tasks) can flow from X to Y. The nature of the information flow depends on the sets X and Y. For example, if Xє \(\mathcal{P}(D)\) and Yє \(\mathcal{P}(P)\) data are sent from X to Y; if X,Yє \(\mathcal{P}(P)\) both data and tasks could be sent from X to Y.

**Figure 1. A vision system based on information integration.**

In Figure 2 a DS is sketched, note that the arcs are labelled, and the information flow is enabled only if the labels \(w_i\) are "on".

**Figure 2. Automata for a vision system**

**Definition 2:** A dynamic icon \(I\in I\) is a correspondence between a set of metaphors, M, and a set of icons, I; where, metaphors have a perceptive meaning. For example, they may represent visual patterns as well acoustic signals or a combination of both.

**Definition 3:** A VDE is a 6-tuple <DS, DI, M_m, M_p, M_A, M_W>, where:

a) \( M_m : M \to P \) (metaphor - process)

b) \( M_p : PXM \to DI \) ((process, metaphor) - icon)

c) \( M_A : DI \to \Delta \) ((icon,icon)-transition functions)

d) \( M_W : DBDI \to \{0,1\} \) (mask on the relational arcs)

\( M_m \) is a function that assigns a metaphor to a process, it is many to one because the corresponding process may assume different status conditions during the evolution of the DS; \( M_p \) is one to one in order to avoid
ambiguities, it is responsible for the assignment of a \( \gamma \) a pair \((P_i, m_k)\) determines the current value of the icon \( \gamma_i^{(k)} \). \( M_\Delta \) defines the relations between \( \gamma \)’s; this function has been introduced to handle the evolution of a DS at a visual level; the function \( M_w \) assigns a mask flag value to each relational arc. The mask flag is useful to visualize the information flow in the DS. Figure 3 shows the diagram of the functions above introduced.

The previous definition can be very useful to represent the evolution of a process in a DS. In these cases, a process, \( P_i \), will be defined as a sequence of virtual processes \((P_i^{(k)} \mid k=1,2,\ldots,N)\) corresponding to a sequence of dynamic icons \((\gamma_i^{(k)} \mid k=1,2,\ldots,N)\), where:

\[
\gamma_i^{(k)} = M_\Delta(P_i^{(k)}, m^{(k)})
\]

for \( k=1,2,\ldots,N \)

**Figure 3. Functions describing the relations between the sets DI, P, and M.**

DS’s can be represented by direct weighted graphs, on the other hand their complexity (number of nodes and arcs) makes difficult their design and programming.

Visually, VDE can be also represented by direct graphs, \( G \), nodes of which are \( \gamma \)’s and the labelled arcs are determined by the function \( M_\Delta \), the label values, \( \lambda \), are determined by the kind of \( \Delta \) function (\( \lambda = 1,2,3,4,5 \)), the mask value \( m \) is fixed by \( M_m \). Moreover, a dynamic icon \( \gamma \) may represent recursively a sub-set of the VDE, named VDE\( \gamma \). Such kind of dynamic icon is named compound.

Two compound, \( \gamma_1 \) and \( \gamma_2 \), are linked by a compound arc \((M_\Delta(\gamma_1, \gamma_2))\) iff a subset of arcs, with equal label, exist between the corresponding sub-graphs VDE\( \gamma_1 \) and VDE\( \gamma_2 \). The mask flag of a compound arc is set on the basis of AND-OR rules applied to the mask-flags of the corresponding arcs. Dynamic icons and arcs, which are not compound, are said primitive.

The introduction of compound and primitive dynamic icons makes possible to organize a VDE in a hierarchical way (Figure 4), and this approach is useful to handle a DS at different levels of refinement. The hierarchical structure of the VDE, allows to develop and update the visual design of a DS, and to focus the attention easily where errors and bugs occur. Moreover, a VDE controls the evolution of a DS at different levels of detail (from compound to primitive).

The evolution and the programming of a DS can be exploited via VDE by testing the syntactic correctness of visual graphs. Visual parsing is recursively applied to compound-elements, until primitive-elements are reached. The parsing phase is given by a graph-grammars [8].

A graph representing a VDE, must satisfy the following properties, that are used during the parsing phase:

a) The graph, \( G \), of a VDE is connected;
b) \( G \) has labelled input/output arcs;
c) valid sub-graphs of \( G \) have input/output arcs labelled with the same label;
d) direct paths exist from input to output nodes of \( G \) and its valid sub-graphs.

**Figure 4. Conceptual organization of a VDE.**

The automatic inspection of a VDE may detect syntactic errors during the visual definition of a DS, for example the proper direction of an arc can be easily tested; moreover, during the visual editing some semantic inconsistencies can be discovered, for example is not allowed to input or output processes, data and models that do not match correct prototypes.

The whole syntactic correctness of a DS is then tested during the parsing phase of the VDE. The parsing consists on a match-merge procedure applied to the graph, \( G \), that can be considered as a visual program. The match step tests the consistency of sub graphs \( G' \) of \( G \); for this purpose standard graph-matching algorithms can be used; their computational complexity is strongly reduced.
to \( O(L) \), where \( L \) is the number of arcs, because of the constrains imposed by the graph grammar. The merge step creates a super-node, input (output) arcs of which are the input (output) arcs of \( G' \). The parsing is successfully if a single super-node is obtained, at the end of this phase; the merge step depends on the number of nodes in \( G \). The consistency rule depends on the graph grammar, and how it has been defined in the construction of the visual program.

3. The M-VIF machine.

The first prototype of the VDE, above described, has been implemented on the M-VIF machine, as an example of DS oriented to vision problems. Its architecture is based on a Compound Node (CN), which is composed of 4 functional modules (see Figure 5):

- the C-module is the controller of the CN, it manages the evolution of the computation in M-VIF;
- the modules H's are dedicated to data processing;
- the modules IP provide for the input/output data management;
- the module LN (Link Network) is dedicated to the interconnection of CN's, in order to realise several reconfigurable network topologies.

The emulation of M-VIF has been carried out on the reconfigurable and heterogeneous architecture based on the HERMIA machine. It includes 16 general purpose PU's (INMOS T800) and a bank of 6 Digital Signal Processors (INMOS A110) .

The system operates in pipeline modality. For our purposes we use only one compound node, where \( H_1 \) and \( H_2 \) have four processing units and they are dedicated to the segmentation and matching phases respectively. \( H_3 \) performs the integration and decision phase and it requires only one processing unit.

The image I/O is handles by the controller, that loads the data in the share memory, a Broadcasting/Multiplexer Data Unit units directs the data and intermediate results to the appropriate processes in each \( H_i \).

The evolution of a distributed process is based on firing conditions, which must be verified at the input of each state \( x \). Two sets are introduced for each \( x \) to define firing rules:

\[
IN(x) = \{(s,x) \mid s \in S\}
\]

\[
OUT(x) = \{(x,s) \mid s \in S\}
\]

Moreover each input is partitioned:

\[
IN(x) = I_1(x) \cup I_2(x) \cup I_3(x) \cup I_4(x) = \phi\] for \( i \neq j \)

Each element of \( I_i(x) \) is univocally determined by an integer index \( i_j \) for \( j = 1, 2, ..., k_i \). To each \( I_i(x) \) a logic function, \( f_i(x) \), is associated:

\[
f_i(x) = \bigwedge_{j \in I_i(x)} w(i_j, x) \bigvee \bigwedge_{r \in I_i(x)} w(r_j, x)
\]

The introduction of the mask function \( f_i \) and the partition induced by \( IN(x) \) and \( OUT(x) \), allows to implement the \( \{ A_i \} \) functions holding in the DS.

The computation performed by M-VIF depends on the values of the mask function, \( w_0 \), at the starting time, \( t_0 \), i.e. for each \( x \), the values \( w(i_j, x) \) are set initially to "0" and "1". The updating of the mask values is determined by those elements of \( S \) that, firing at a given time, assign new mask values to the elements of \( OUT(x) \). Therefore the value \( w_{i+1} \) at time \( t_{i+1} \), depends on both the values \( w_i \) and the results at time \( t_i \). The computation end as soon as the condition of firing is false for all the elements of \( S \). Note that the evolution of the whole mask values is determined by the deterministic tasks running in each state \( x \).


In the following, the VDS design for the M-VIF machine is described. It is based on three set of visual elements: Metabase (Metaphor database), Icons, and Arcs.

- **Metabase.** Is a database of metaphors which contains visual, acoustic and text patterns. The dynamic evolution of a process is represented by three colours, default values are:
  - green for active
  - orange for wait

\[
\begin{align*}
\text{Figure 5. The architecture of the CN-node.}
\end{align*}
\]

Here \( w(i_j, x) \) is the mask value of the transition \( (i_j, x) \). From the previous definition follows that \( f_i(x) \) is "1" if and only if each transition \( j \in I_i(x) \) is "1" and each \( f_r(x) \) with \( r \neq i \) is "0". The firing rule can be now stated as follows:

\[
\text{fire } x \text{ iff } \exists I_i(x) \text{ such that } f_i(x) = 1
\]

The introduction of the mask function \( f_i \) and the partition induced by \( IN(x) \) and \( OUT(x) \), allows to implement the \( \{ A_i \} \) functions holding in the DS.

The computation performed by M-VIF depends on the values of the mask function, \( w_0 \), at the starting time, \( t_0 \), i.e. for each \( x \), the values \( w(i_j, x) \) are set initially to "0" and "1". The updating of the mask values is determined by those elements of \( S \) that, firing at a given time, assign new mask values to the elements of \( OUT(x) \). Therefore the value \( w_{i+1} \) at time \( t_{i+1} \), depends on both the values \( w_i \) and the results at time \( t_i \). The computation end as soon as the condition of firing is false for all the elements of \( S \). Note that the evolution of the whole mask values is determined by the deterministic tasks running in each state \( x \).
Icons. Dynamic Icons, representing processes, are defined by the pair (colour, pattern). All icons (both compound and primitive) have the same visual organization (see Figure 6). The body contains a visual pattern and an expansion button, the colour of this button indicates also the current status (active, wait, stopped) of the icon. Primitive icons have yellow background, compound icons blue one. Six Input/Output channels are foreseen to implement the IN, OUT lines of each node. The expansion of a compound icon is performed by pushing its button. The expansion of a primitive icon depends on the kind of the information connected. For example, if an icon represents a process it returns the source code in I-PICL language [9], if it represents a kernel its expansion returns the kernel-values. The distribution of the resources (processors, sensors, data knowledge) are handled by the user. This is obtained by creating a link between the appropriate dynamic icon and the corresponding resource.

A first prototype of VDS is under development under Windows3.1 and C++ object oriented language.

5. An example of distributed computation.

In the following, the implementation under VDS of an information fusion technique to retrieve pictorial data is described. It is based on the integration of different data type and co-operating segmentation algorithms. Two, distance functions (Euclidean, Hausdorff) are evaluated and used to reach the best matching.

In the first phase the input image is segmented by four segmentation algorithms (Hierarchical Single Link Clustering (HSLC), Hierarchical Histogram Partition Clustering (HHPC), Hierarchical ISODATA (HISO), Two Phases Clustering (TPC) [10,11], the results are analysed by the corresponding matching modules (MM₁,...,MM₄). Each module MMᵢ has two different input data: metrics (M₁,M₂), the segmented image, and the target image (P), and it provides an ordered list of the retrieved objects (candidates). The ordering is performed on an evaluation parameter (qᵢ), ranging in the interval [0,1], related to each candidate.

The last module (Integrated Decision) performs the best retrieval on the basis of the results of the modules MMᵢ's, which are combined by using several decision functions; in our case have used alternatively the mean value, the maximum value, and the vote technique.

Figure 7 shows the visual organization of the proposed retrieval system. Figure 7a shows the tools used to draw the iconic algorithm, and to distribute the resources. It also shows the upper level of the retrieval algorithm, sketched above. Three compound icons represent the main phases of the retrieval procedure (modules: Segm, MM's, and ID).

Figure 7b shows the expansion of the Segm icon. Figure 7c shows the expansion of the MM icon, the primitive icon Prot contains the prototype database, the compound icon Mod represent the distance functions, and criteria to be used in the matching phase. Figure 7d shows the expansion of the primitive icon HISO.

An example of resource distribution management

Figure 6. Dynamic Icons: a) Process; b) Model; c) Compound; d) Input data; e) Output data.
is given in Figure 8. In this example, the user selects the compound icon Segm and assigns the processes associated with this icon to one of the computing site (the HERMIA-machine).

The tools windows includes also the following buttons:
- the magnifying glass performs debug of an iconic program;
- the clock starts the execution of an iconic program;
- the hand allows to navigate into the working window;
- the scissors and the magic wand are used to perform icon cut and paste respectively;
- the button "T" allows to include comments into the icon body.


The paper shows the general definition of Visual Dynamic Environment to develop distributed applications on a DS. A preliminary version of VDS for the M-VIF machine is also described. Preliminary tests show that VDS may improve the quality of the implementation of distributed algorithms; substantial performance has been also observed in software productivity.

At the present status of its implementation, VDS allows to develop algorithms on an emulated version. Further developments foreseen the use of VDE in a DS, which includes different platforms (UNIX workstations, HERMIA machine) and different sensors (CCD camera, infrared and acoustic sensors) distributed on a local ethernet network.

REFERENCES


Figure 7. An example of distributed algorithm: (a) main algorithm level; (b) expansion of the Segm icon; (c) expansion of the MM icon; (d) I-PICL HISIO program.

Figure 8. An example of resource distribution management.