# A User-Flow approach for multi-user applications with DMS (Distributed Modules System)

Sylvain Huet

Scol Technologies, Cryo-Networks
44-50 avenue du Capitaine Glarner
93585 Saint-Ouen Cedex

tel : +33 (0)1.49.48.69.04
fax : +33 (0)1.49.48.69.01
e-mail : s.huet@cryo-networks.fr
http://www.cryo-networks.com

**Abstract**

We have developped tools that bring to non-programmer people the power to design multi-user applications over internet. Multi-user application is not only 3d virtual world, it is more generally application in which people can interact with or through the same "object". Because they are addressing non-programmer people, these tools have to hide programming problems as well as distribution problems. This paper describes the DMS architecture on which both tools and applications are based.

# 1 Introduction

## 1.1 Starting point

In this paper, multi-user application stands for application in which users can interact together over the network. This interaction is organized around a server which is at least the meeting point of users, but which usually runs some parts of the application. Multi-user applications is most probably the next step in internet conquest, after email, forums, broadcasting informations (Html), videoconferencing.

Multi-user application is a more abstract kind of communication : when people speak about 'communication and internet', they usually think about text chat, audio chat, videoconference and file exchange. But, typically, how would you play chess with these communication tools ? Some recent developments restrict multi-user application to 3d-virtual worlds, but won't you prefer to play chess on a readable 2d board ? It seems that 3d environment is probably one of the most natural and impressive way to interact with other users, but this is not the Graal.

There are at least two major difficulties with multi-user application development : distribution and synchronization. In general, multi-user applications need to run both on user and server side. This means that a choice has to be made for each part of the application : will it run on user or server side ? This is the distribution question : of course only programmer people can understand and solve this question. The synchronization question is an outcome, and is critical only when low latency is required, which is not always the case with multi-user applications. This paper will address mostly the distribution question.

We wanted to develop an architecture that would allow non-programmer people to design multi-user applications, but large-public cannot understand and solve distribution question, so the architecture has to hide it. What can large-public understand ?

- Mouse control
- Lego-like systems : putting two bricks together.
- Probably some easy electricity problem , but with an angel-guard who prevents me from shortcuts
- Maybe some simple dataflow graph

In any case, large-public cannot and simply does not want to understand :

- Programming language, even simple script.
- Synchronization constraints : "this thing should happen before this one but after this one."
- …

So we concentrated on a modular approach : a module is like a lego brick or like an integrated circuit with some inputs and outputs. This architecture is called DMS

(Distributed Modules System). Basically, the designer only has to put modules together and link outputs to inputs. But the question is : what is the meaning of a module ? what is the meaning of these links ? how are modules organized ?

### 1.2 other concerns

There is a lot of non-scientific literature about virtual worlds, and there is usually an issue standing out : ubiquity. In usual virtual worlds, user is only in one place at a time. Why ?

## 2 Graph versus Tree

### 2.1 3D trees applied two non-3D issues

There are a few works based on 3D virtual worlds that deal with multi-user issues ([1], [2], [3]). Some of them are based on Vrml, some of them are not. However they are based on the 3d hierarchic tree. In these trees, the father-son relationship may mean :

- 3d hierarchic : the son coordinate system is related to the father's one
- life hierarchic : the son dies when the father dies, the son moves when the father moves accross the tree
- visibility hierachic : the son is inside the father, and if the father is closed, the son does not see his grand-father

Moreover, the tree is not always homogeneous : the father-son relationship meaning can be different from some part of the tree to another.

Nodes can be very different :

- 3d element (mesh, camera, light, bounding box, …)
- 2d interface
- script

We want to understand the way to simplify the relationship between nodes. Let us consider the following basic example :

"In the user interface, beside the 3d window there is a banner. There are two rooms A and B. In room A there is a 3d button. When a user clicks on the 3d button, a password is asked through a tiny 2d interface. If the password is correct, the remote banner editor, a 2d interface, is displayed on his screen. If the password is wrong, he moves into room B (the prison)."

Let us try to build the tree. We need five nodes, banner, room A, room B, 3d button, password. Then we have relationships between nodes :

- 1} visibility relationship between the 3d button and room A

- 2} no visibility relationship between room A and room B (room B is beside room A)
- 3} implication relationship between the 3d button and the password : by clicking on the 3d button we open password interface. Let us call it 'implication', we precise it later in this paper.
- 4} implication relationship between the password and the banner to display the remote editor
- 5} implication relationship between the password and room B

{1}means that 3d button is a son of room A
{3}means that password is a son of 3d button
{5}means that room B is a son of password
So that room B is a son of room A, which is impossible due to {2}

This implies that we cannot have one tree to describe all these relationships. We need a graph.
Then, there is a very simple solution with only four nodes : room A, room B, password and banner (we consider that the 3d button is part of room A so that it is not a different node). There only remains implication relationships :
- from room A to password
- from password to banner
- from password to room B

By the way, we are putting the focus on implication relationships, and considering other relationships as secondary.

### 2.2 Mobility versus Cloning

By removing tree structure, we are losing 'life-relationship', and moreover mobility. Within a tree, it is really convenient to move a whole subtree only by moving one node. By example, a '3d robot' node is the father of a script node that describes its behaviour : the behaviour remains linked to the robot as the robot moves inside the tree.

But we have to think about it : let us consider scalability issues and the multi-servers solution. Some parts of the tree are running on computer A, and some other parts are running on computer B. As the 3d robot is moving from computer A to computer B, the script node has to move too, so that we do not have mobile code but mobile processus, which is quite harder problem, particularly for security reasons ([4],[7]).

A solution to mobility is cloning : cloning means that the 3d robot is described by genes, and that only genes are moving from one point to another. There, the robot is re-built from the genes, and the initial one is simply destroyed. If not, there is ubiquity. There is some advantage for cloning versus mobility : the robot might move from a 3d room into a 2d room. If the 2d room can interpret the genes, it will build a 2d robot. Moreover, there is no security problem : gene is not code but seed. The 2d robot program is part of the 2d room program.

# 3 User-Flow Graph

## 3.1 Graph, Modules and Links

3.1.a definitions
In our model, each module is simply a brick with named inputs and outputs. And modules are organized in a graph with oriented links. Links represent implication relationships. From outside there is no structural difference between a 3droom module and a password module : they all are bricks with inputs and outputs. Basically there will be an output for each clickable 3d object. There will be an input for starting the password interface, and two outputs corresponding to a right and wrong password.

We assume that there is no dependency between modules : modules can be written by different people in different places in different time, we should not need anything else than the list of inputs and ouputs to use a module.

3.1.b distribution
The graph describes the application structure. We solve the distribution question with a copy system : the same graph runs both on server and client side, but only a part of the graph may run on the client side :

- some modules do not need computation on client side
- with the previous example, there is no need to run the client-side of roomB when the client is in roomA.

Thus, there is a dynamic activation mecanism :

- only the server-side module can order the activation of the client-side on a given user
- both server-side and client-side modules can desactivate the client-side module
- a module can never activate/desactivate a client-side module from another module

3.1.c graphic interface
There is a minor question that we have to mention now. The described structure is supposed to address applications with multimedia fonctionnalities. This means that modules display 3d, 2d, text, … in a graphical interface on both client-side and server-side. Thus, the graph comes with two documents: a server document and a client document. Each document is composed of windows (child or popup windows) and zones (geometrical subparts of windows). Each module may need different zones : therefore there is an application (not a bijection) between the needed zones and the actual zones of client and server document.

## 3.2 Users

We define two types of users : real user and virtual user. A real user is simply one of the connected clients : it means a processus that is connected to the server. There is exactly one real user for one client processus. A virtual user is a user that is not a client : it might be some robot and more generally some functionnality. Moreover, a virtual user can be global or local. Global users are managed by the server, local users are managed by clients, and server simply does not know them : it is usefull to create local and non distributed fonctionnalities.

Let us remember the previous example, and consider user Alice. We can say : Alice "is in" room A. She clicks on the 3d button, then she goes into password module, but remains present in room A (ubiquity). Then she goes into banner module through the 'remote editor' input. At each "move", there is a **message** running the graph from one module to another. This message encloses reference to Alice, and maybe additional information, but the main part of the message is the reference to Alice, the answer to the question "who is moving in the graph ?".

That is the reason why we call the graph a **User-Flow Graph**. Thus, messages are running through the graph, referencing one user, and sometimes enclosing parameters.

To increase performance, inputs and outputs can be located on client-side or server-side. Because there is not only one client, the system has to decide on which client a message going to a client-side input has to be fired. Basically we have three rules :

- 1} a message referencing a real user U and going to a client-side input goes to the client corresponding to user U.
- 2} a message referencing a virtual user and going to a client-side input stays on the server.
- 3} a message coming from client-side output of client C can only reference the real user corresponding to client C.

### 3.3 UserInstances, Location and UserClass

Unlike dataflow graphs ([5]), we have to deal with ubiquity : in a dataflow graph, each data has a precise location. Here, a user can be in several modules at the same time. We introduce the UserInstance concept.

A UserInstance is a couple (User,Module), where Module represents one Location of the User, so that from now we will talk about (User,Location) couple. We have the following rule :

for one User and one Location, there is at the most one UserInstance (User,Location).

Thus, we handle ubiquity : for one User, there might be several UserInstances in different locations.

We introduce some parameters to compose the complete genotype of the UserInstance :

- a User
- a class

- some ressources
- a security state

Each Location is managing a set of UserInstances. A Location can :
- create an UserInstance based on a User (known from input messages) and a Location (only itself)
- destroy an UserInstance
- change part of the genotype : class, parameters and security state

A Location can define another module as UserClass module for a given class. The UserClass module will manage a set of UserInstances with the given class and possibly different Locations :
- destroy an UserInstance
- call method of UserInstance

We now consider the security issue and then precise how the distribution question is solved.

# 4 Security

## 4.1 Initial concept

By security we do not mean implementation issues, nor mobile code issues. The security system defines wether an UserInstance is visible or not from another UserInstance. Moreover, we need a system that allows groups and super-users.

For each Location, there is a tree of groups of UserInstances. And each UserInstance defines a commutation flag. The visibility rule is the following.

UserInstances A can “see” UserInstance B if :
- B’s group is in the subtree of A’s group
- A’s group is in the subtree of B’s group AND B’s commutation flag is set.

## 4.2 link with distribution

Basically, UserInstances related to real or global Users are managed on server-side (creation, destruction, mutation) and synchronized on client-side Location module : UserInstances are created, destroyed and modified on client-side according to server orders.

The security system adds a filter to this synchronization : UserInstance A is synchronized on client C if :

- there is an UserInstance B related to client C on the same location than UserInstance A
- B can “see” A

Else the client C will simply not know the existence of UserInstance A.

# 5 Implementation

A first implementation of DMS (Distributed Modules System) has been developped based on Scol Technologies (Standard Cryo On-Line). A commercial version of the tool is available : SCS (Site Construction Set). And a simple wizard for very large public is also available : Cryonics. It is possible to try the system for free as a client ([6]).

## 5.1 Requirements

There are some major requirements :

- mobile code : client-side module means client-side program that has to be downloaded. The security problem needs to be addressed.
- dynamic environment : client-side modules are activated dynamically. The mobile code is compiled during run-time, precisely when the client-side module is activated
- multi-environment : modules might be developped by different people that have no knowledge about each other ; to avoid conflict, modules need to run in different environments which have nevertheless common parts
- strong IP communication capability

There are other minor requirements :

- same technologie on both client and server side
- good multimedia features : 3d, 2d, text, audio, video, …
- multi-platform technologie

## 5.2 Scol (Standard Cryo On-Line)

Cryo-Networks has developped a technology dedicated to on-line multi-user applications.

This technology is based on a virtual machine – the Scol machine –, that is either the client and the server. The virtual machine is managing dynamic environments : basically, we define an environment as a list of functions and variables. The Scol

machine can handle several environments at the same time, and two environments may have a common end of list, so that this common end of list is a shared environment.

The Scol machine memory management is automatic, and implements a Garbage Collector system.

The virtual machine uses a special programming language, called Scol language. Its main characteristics are :

- functionnal language
- polymorphic and static type-checking
- run-time compiler
- inter-machine communication model based on mutual suspicion
- communication constructors to deal with IP communication

Many high-level libraries are available : 3d, 2d, text, audio, video, file, sql, telnet, big numbers, …

There is free download of the complete documentation and the virtual machine (compiler included) on [6]

### 5.3 SCS

The SCS is a tool that implements DMS architecture. The tool presents the User-Flow graph in a graphical way and allows users to organize modules and draw links with the mouse. The tool itself is written in Scol language. Today, about 60 modules are available.

The development of a module requires at the most three components :

- server-side module code
- client-side module code (when needed)
- editor-side module code

The editor-side of a module allows to change inner module parameters : modules all look the same from outside (a brick with inputs and outputs), but of course there are some inner differences between a password module and a 3d space module ! By example, the 3d space module editor allows you to define the 3d scene, and the password module editor allows you to define the password and the prompt message.

The first release of SCS implements the first release of DMS architecture : complete implementation of modules graph, but primitive User management system. An alpha version of the second release of DMS is available on demand, with full User management system.

### 5.4 Applications

The SCS allows to build easily 3d virtual worlds, but also interactive internet radio, monitoring tools, video games, … The modular approach is very convenient for programmers : they need only to develop a few additional modules and then integrate the application graphically.

The SCS was beta-tested by 60 external non-programmer people, that realized different kinds of applications : shopping mall, virtual gallery, intranet offices, murder party, ski simulator, …

Cryo-Networks has developped a few "showcase" 3d sites, a paintball game, an interactive internet radio, and the new on-line game is totally based on SCS. This is today the main development tool of the company.

## 6 Future work

There are some fields that remain unexplored :

- multi-server extension : how to distribute the graph over different servers
- dynamic updating of modules : is it possible to shut down and restart a module independently ?
- typing of the parameter included into messages
- …

## References


[1] Mitra. *"Living Worlds" Concepts and Context*. Living Worlds. VRML consortium.1997

[2] Mitra. *The Architecture of Living Worlds*. Living Worlds. VRML consortium.1997

[3] M.Vellon, K.Marple, D.Mitchell, S.Drucker. *The Architecture of a Distributed Virtual Worlds System*. Virtual Worlds Group. Microsoft Research. 1998.

[4] Cédric Fournet,Georges Gonthier. *The reflexive chemical abstract machine and the join-calculus*. Para project. INRIA. POPL'96

[5] K. R. Traub, G. M. Papadopoulos, M. J. Beckerle, J. E. Hicks and J. Young. *Overview of the Monsoon Project*, January 1991, In *Proceedings of the 1991 IEEE International Conference on Computer Design*, Cambridge, MA, October 1991

[6] S.Huet. *Scol Language* . http://www.cryo-networks.com . 1998

[7] S.Huet. *Sécurité et processus mobiles* . INRIA. http://pauillac.inria.fr/~shuet/. 1996