Intl. Conf. RIVF'04
February 2-5, Hanoi, Vietnam

UML tool for multi-user

Duy-Tai Nguyen, Tien-Dung Le

***Abstract*—As we know that "the Unified Modelling Language (UML) is a language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modelling and other non-software systems" and many enterprises are now using it to product their software. In the market, there are some well-known UML supported tools as IBM Rational, Together, and Poseidon…that are only used by a designer at a moment. So, in this paper, we would like to present an approach to build a UML tool (named CP) for multi-user. The first part introduces UML, a limitation of some well-known tools, and our solution whose details are presented in the second part. The next parts are the construction of CP tool and the conclusion.**

***Index Terms*—Distributed system, Multi-user, Tool, UML.**

I. INTRODUCTION

UML is a common language, used to describe the artifacts of projects to all users. It allows the designers present theirs ideas in diagrams that can be easily stored and retrieved. Recently, many well-known tools as IBM Rational (its previous version is Rational Rose) of the IBM Company, Together of Borland Software Corporation support it. Three are also some free tool written in java as Poseidon of the open source project ArgoUML. But a UML tool, which supports to design by many people, had been a demand for long time. We are now able to supply a version of this kind, called CP, which can be executed by many designers in different platforms. The CP tool is a product of a project at IFI (Institut de la Francophonie pour l'Informatique) and is written in java. In fact, the CP is a distributed system that has a server and a number of clients running on different host. When a client changes the design, all the others can see this change immediately and all the designers are able to communicate and discuss to have the best design. To solve the problem of using resources, we use "business actions" concept. The details of this idea are introduced in the next part. The third part will show the architecture of CP using the RMI (Remote Method Invocation) protocol that might reduce our programming efforts. And we use the PAC model to transfer messages among cooperating agents of a CP client.

II. TOWARD SIMPLE CP

All data, stored in CP server, are shared to all clients. Therefore the data synchronization problem arises because it is possible that more than one client request to change the design at the same time. We propose the solution based on "business action" to ensure that the design is always in a common state.

Duy-Tai Nguyen and Tien-Dung Le, Institut de la Francophonie pour l'Informatique, Hanoi, Vietnam. E-mail: ndtai@ifi.edu.vn, ltdung@ifi.edu.vn.

A. Concepts

State: A state of the current design in CP.

U : Set of clients.

S^A : Set of states at client A.

Common state: A state s that affects all states of clients.

S : Set of common states that affect all states of clients.

It is easy to see that $S = \bigcap_{u \in U} S^u$

X : Set of actions (might be performed by many clients).

X^A : Set of action at client A.

Business Action:

An action x is a business action if it satisfies $s \xrightarrow{x} s'$

For example: A business action performed by client A is to establish an association between the actor "customer" and the use case "register". We have $s_1 \xrightarrow{x} s_2^A \equiv s_2$.

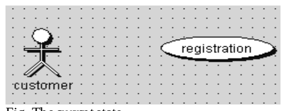


Fig. The current state

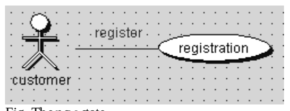


Fig. The new state

B. Concurrency Control

In a more complex case, client A wants to change the name of this actor to "user" as the same time as client B wants to delete the use case "registration". What will happen?

We assume the action x^A makes $s_2 \xrightarrow{x} s_3^A$; the action x^B makes $s_2 \xrightarrow{x} s_3^B$. It is clear that neither x^A nor x^B can be business actions because the s_3^A will not be able to change the state at client B and vice versa. Therefore, we define a set T of temporary states so we can present two actions as following: $s_2 \xrightarrow{x^A} s_3^A \equiv t_A$ and $s_2 \xrightarrow{x^B} s_3^B \equiv t_B$. The combined action $x = \{x^A, x^B\}$ is a business action if and only if it satisfies

$$(t_A \xrightarrow{x^B} t_{BA}) \wedge (t_B \xrightarrow{x^A} t_{AB}) \wedge (t_{BA} \equiv t_{AB} \equiv s_3).$$

I.e. the result state is independent of the order of actions. The action x in the example case is a business action.

In general, the combined action is a business action if every single action is performed on a separated part of the design. But to establish criterions of separated parts is very hard and to divide the design into separated parts requires a lot of time. A normal way is that a design can be considered as a graph $G = (V, E)$. The set V is a set of vertices that are actors, use cases, classes, objects … and the set E is a set of arcs that are associations of vertices. E.g. in the figure 2, V is {“customer”, “registration”} and E is {“register”}. And the problem becomes one of well-known graph connectivity issues.

Another simple way is that we permit only one client to change the state, and all others have to wait to get the permission. This solution is a specific case of the previous way when the graph G is treated as a connected graph. But it must allow all clients to see the change of the design. This way ensures that all the actions will be business actions.

However it requires managing the permissions the server give to clients and each part is associated with a semaphore. In the latter solution, there is only one semaphore associated with the hold design. We can use one of several methods in (1) to control the clients’ usage.

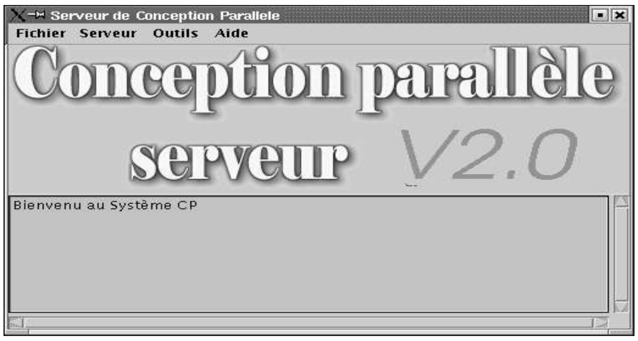
In brief, we have transferred the main CP problem to the graph issues and the resources management that are popular. This approach helps us to complete the CP version 2.0 presented in the next part.

III. CP VERSION 2.0 WITH JAVA – RMI

Our application is built for the simplest case: G is a connected graph. So, it’s enough to use a semaphore. Because G is a connected graph, at a moment, we permit only one client to change the state, and all others have to wait to get the permission. But it’s evident that we must allow all clients to see the change of the design. So, our application is constituted from two parties: client and server.

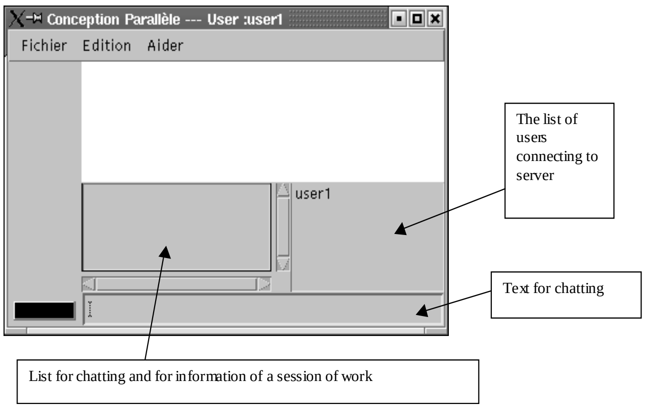
On the server, we save the current state of the conception. When a client, who has permission to change the state, has completed a business action, which is verified by client’s part of our application, the server will update the current state and send the change to all others.

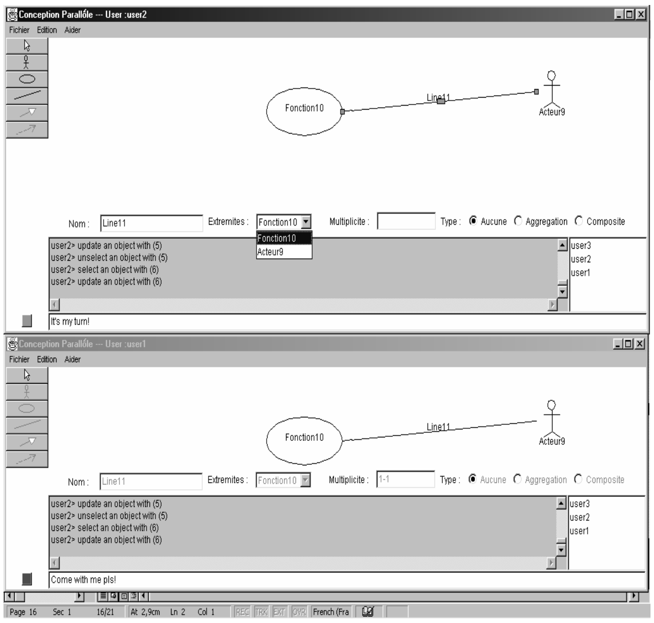
To manage the permission to change the state, we use a semaphore on the server. And to assure the functions of the server and the client, RMI (Remote Method Invocation) seems to be the best solution.



Bienvenu au Système CP

With some functions applied to the server, it can take back the permission of work from any client; can inform the client having the permission; can save and load a conception being designed; can print a conception….


Conception Parallèle --- User :user1
Fichier Edition Aider
user1
The list of users connecting to server
Text for chatting
List for chatting and for information of a session of work


Conception Parallôle --- User :user2
Fichier Edition Aider
Fonction10
Line11
Acteur9
Nom :
Extremites :
Multiplicite :
Type :
Aucune
Aggregation
Composite
Conception Parallôle --- User :user1
Page 16 Sec 1 16/21 At 2,9cm Ln 2 Col 1 REC TRK EXT OVR French (Fra

Nom : Line11 Extremites : Fonction10 Multiplicite : Type : (●) Aucune ( ) Aggregation ( ) Composite

Fonction10
Acteur9

```
user2> update an object with (5)
user2> unselect an object with (5)
user2> select an object with (6)
user2> update an object with (6)
```

user3
user2
user1

It's my turn!

Nom : Line11 Extremites : Fonction10 Multiplicite : 1-1 Type : (●) Aucune ( ) Aggregation ( ) Composite

```
user2> update an object with (5)
user2> unselect an object with (5)
user2> select an object with (6)
user2> update an object with (6)
```

user3
user2
user1

Come with me pls!

As you can see, when the user2 has permission to work (the green button), all others have to wait to their turn (the red one), but they can see what the user2 do. And if there is no user taking permission, the button is black to show that at this moment everyone has a same right and opportunity to work.

When the user2 finish a business action, server will update the state of the conception on server and inform to all other users (update an object).

In version 2.0, we also allow clients to designer a sequence diagram.

IV. Conclusion

Base on the experience with the solution which allows only one client work at a moment, we state that it is possible to allow many clients work at the same time. The problem is how we solve the graph connectivity issue. If we have a graph with, for examples, k sub-graphs connected, so we permit k clients to work parallel. After client i has finished a business action, there are 3 cases:

+ G has (k-1) sub-graphs connected: in this case, client i will be deleted from the list of workers by the server. This solution is acceptable because he has finished his work.

+ G has also k sub-graphs connected: we continue to permit k clients to work parallel

+ G has (k+1) sub-graphs connected: we permit (k+1) clients to work parallel

We can see that the role of server becomes very important in the general solution. He decides the list of workers, decides also the speed and the condition of work. A business action changing numbers of sub-graphs connected is, for the server, more important than one who doesn't. If server verifies numbers of sub-graphs after any business action, it slows down the conception. Therefor, the clients don't feel comfortable. So, the classification of business actions is an important mission to be solved.

References

[1] Alfred J. Lupper, *An Overview of Projects on Distributed Operating Systems*. Available as http://www-vs.informatik.uni-ulm.de/DOSinWWW/DistribSys.html

[2] Frédéric Gardi, "On the Partition of an Interval Graph into Proper Interval Subgraphs", 01-2002.

[3] Projet72, *Conception Parallel*, IFI-2002.