Patterns for Collaborative Virtual Reality

Fabien Costantini[#], Christian Toinard[#], Nicolas Chevassus[+]
[#]CEDRIC – Centre D'Etudes et de Recherche en Informatique CNAM
292 rue Saint-Martin 75141 Paris Cedex 03 France
costanti@cnam.fr, toinard@cnam.fr
[+]EADS Corporate Research Laboratory
12 rue Pasteur BP 76 92152 Suresnes Cedex France
nicolas.chevassus@eads-nv.com

Abstract

The paper considers two design patterns enabling to extend automatically a virtual prototyping application into a collaborative application. The proposed patterns deal with distributed designation and automatic aggregation. They provide adequate properties in term of cooperation and mobility while guarantying good usability over the standard Internet. Thus, distant participants can share and cooperate to define common agreed design of a virtual prototype while alternating between disconnected and meeting works. These two patterns take part of a more complete framework including several other patterns that address distributed protection, meeting schedule, real time membership, real time awareness, progression, parallel work, refreshment, security, address allocation and multicast connectivity. First pattern provides a distributed designation of a virtual scene tree that supports mobility. Unique names are maintained in a fully distributed way among disconnected workspaces and meeting replicas. Second pattern addresses the aggregation of different disconnected workspaces into a global scene tree. That aggregation mechanism relies on a peer-to-peer communication model. It enables to aggregate the scene while working on it. Different patterns of our framework rely on existing patterns that we compound. They capture an efficient style of distribution through reuse of existing architectural patterns. The whole framework runs on Unix and Windows, can be deployed easily on the standard Internet, is fully distributed and supports efficiently mobility and conciliation meetings. It is used in a virtual prototyping application at EADS.

MOTIVATION

Designing a collaborative virtual reality (VR) application is a challenging work to support both an efficient distribution among mobile workers and real-time cooperation where a consistent progression does not limit the ability of parallel working. First, adapted metaphors are really missing to support easy and intensive collaboration where participants can set-up and examine quickly different design alternatives. Second, GroupWare toolkits do not identify elementary patterns that can be integrated to any standalone application. Moreover, GroupWare systems do not manage a virtual 3D scene. Third, distribution patterns are missing for the collaboration activity of mobile workers. At last, distributed virtual reality [HLA 97][HAGS 96] comes from the battlefield simulation and do not propose adapted solutions for the collaborative design.

Industrial proposals [IBM 2000] propose a limited style of collaboration. They consider mainly the design review where participants navigate through the scene. [HP 99] proposes a real-time collaboration that cannot be used easily in the context of mobile and distributed works.

GroupWare tools [GREE 98][YALE 99] focuses on room metaphors and windows sharing. They do not consider virtual scene sharing and mobile workers.

Distribution patterns [BERC 96][BERC 2000][BROW 99][SCHM 95][LALA 98] address different elementary problems like message recognition within a flow, management of versions of a collaborative workspace, scheduling of method invocations and shared memory communications. But, patterns for shared virtual spaces and mobile workers are still poorly studied.

A general framework [TOIN 99] is proposed for real-time cooperation among an extended team with support for mobility, parallel working and consistency. Protocols and network architecture [COST 2000] have been defined to deploy the framework securely over the Internet. In this paper, two patterns that compose the framework are described. These patterns record validated ways to structure collaborative virtual design and to solve difficult distribution problems. Each pattern is presented in a canonical form with example, context, problem, solution, structure and consequences.

These patterns are implemented and integrated in a standalone virtual prototyping application. A novel collaborative virtual early application is built in a fully distributed way without any server. The application is deployed securely over the Internet guarantying authentication and confidentiality.

A PATTERN FOR DISTRIBUTED DESIGNATION

Name

Hierarchical Designation Manager

Example

Consider the context of a 3D collaborative virtual prototyping design simulation. Distant designers can set-up in a collaborative way a virtual prototype and attach simulation modules to the different sub-systems. Typically, designers cooperate to allocate space for different sub-systems and connect these sub-systems to show their relationships. During the process, designers can interact with the virtual objects hierarchy and provide real-time inputs to the simulators onto a virtual shared scene.
Figure 1 shows a scene tree related to a ventilation system. That scene tree includes an electrical sub-system and hydraulic sub-system. Figure 1 presents also the 3D rendering of that scene. Suppose we want these sub-system objects to be managed in a fully distributed and efficient way while keeping consistency of work.

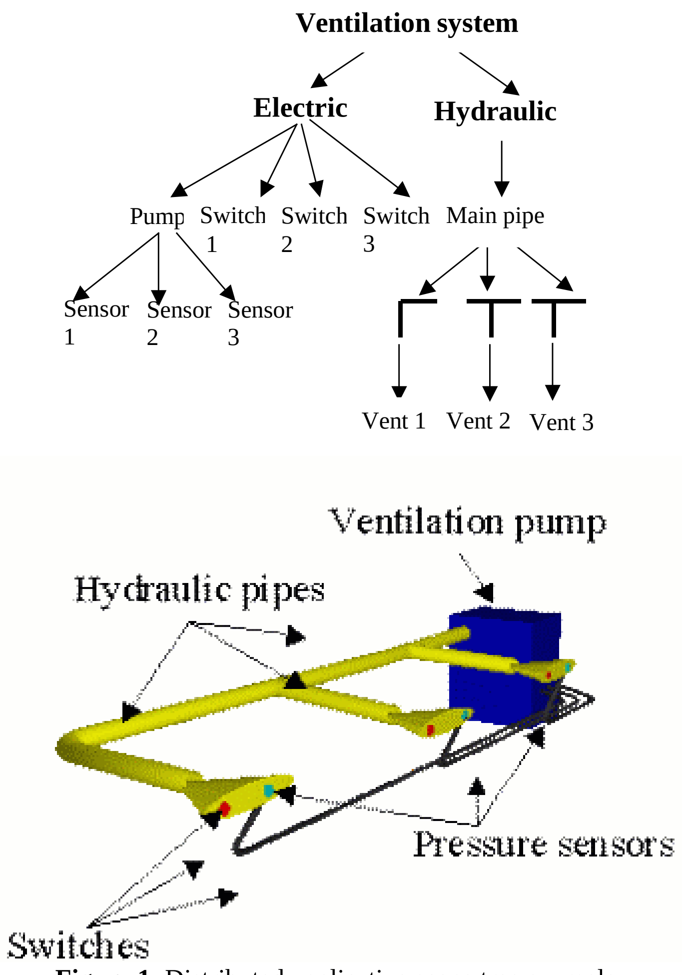


Figure 1: Distributed applicative scene tree example

Context

In a collaborative framework, workers need to share objects on the network. These objects have owners and may depend on other objects. Then, we need a component that manages a distributed applicative scene tree that allows these objects to be handled hierarchically in a fully distributed way.

Problem

The nodes of the global scene must keep unique names in time and space among the different private spaces and replicas.
The solution must guaranty several properties:

- Unique names should be easy to manage and use.
- Two private spaces or replicas cannot contain the same object with two distinct references. For example, during a disconnected phase, the pump can be present in two different private spaces and still be designated with the same name. During a meeting, two replicas use a same name to access the pump object (local or distant).
- If a name designates two distinct nodes, then uniquely one node is active and the other is a pending node that cannot be processed. The pending node must disappear and be replaced by the active one. The system must guaranty that a pending node exists only within a private space. In contrast, two versions of the same node have the same name.
- The distributed scene tree may not reflect the applicative scene tree: objects can be private and remain in the local private space or be public in the global shared space.
- A user creating a node must be able to use a dynamic Internet Protocol address. Despite a dynamic address allocation, the name must remains unique in time and space.

Solution

A peer (i.e. an entity processing a replica or a private space) computes locally a unique name when a user creates a distributed new node (i.e. a Network Decorator Leaf1 or Network Decorator Composite as defined in the sequel) within the scene tree. For example, a unique name contains the Internet Protocol (*IP*) address (*@IP*) of the creation machine, a local timestamp and the position within the tree (e.g. *@IPA,stampA,1.3* corresponds to the first child and third grandson of the node *@IPA,stampA*). A timestamp includes a date, corresponding to the local creation date, plus a random number. Moreover, a node maintains locally the child names. For example, node *@IPB,stampB,1* stores locally the list of its children (*@IPB,stampB,1.1*, *@IPB,stampB,1.2*, *@IPB,stampB,1.3*). To create or to delete a name, a user must be the owner of the father. For example, to create name *@IPB stampB,1.3* the user must own node *@IPB,stampB,1*.
Thus, a node (e.g. *@IPA,stampA*) can be created at the first level on a disconnected basis without being attached to any particular shared scene. It enables a user to prepare some work without knowing in advance in which scene he will enter his work. For example, one designer prepares an electric sub-system and another designer prepares an hydraulic sub-system. Afterwards, they reach a meeting to design a ventilation system and bring their works. Figure 2 gives the automatic merging of the corresponding scene tree. The root of tree corresponds to the name of meeting. It is seen that the unique names do not correspond directly to the graphical position of the objects (e.g. the first vent is named *@IPB,stampB,1,3* while the second vent is named *@IPB,stampB,1,1*).

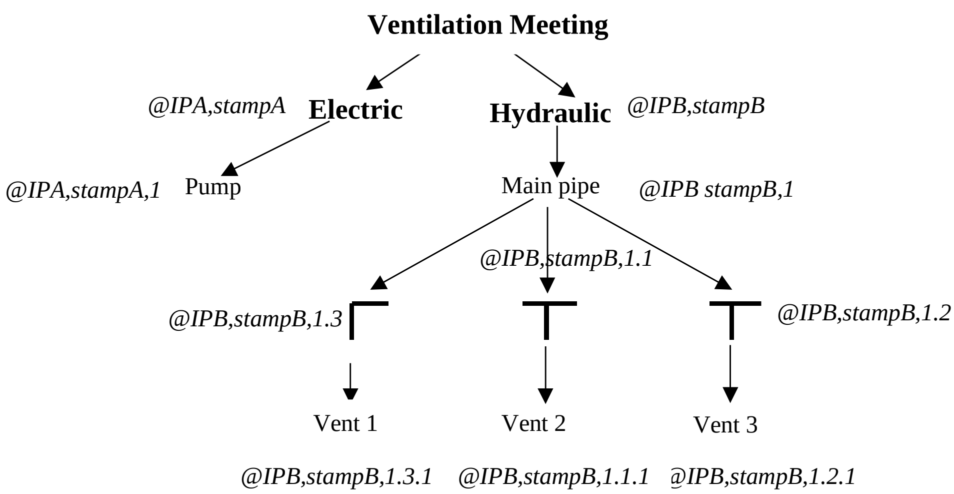
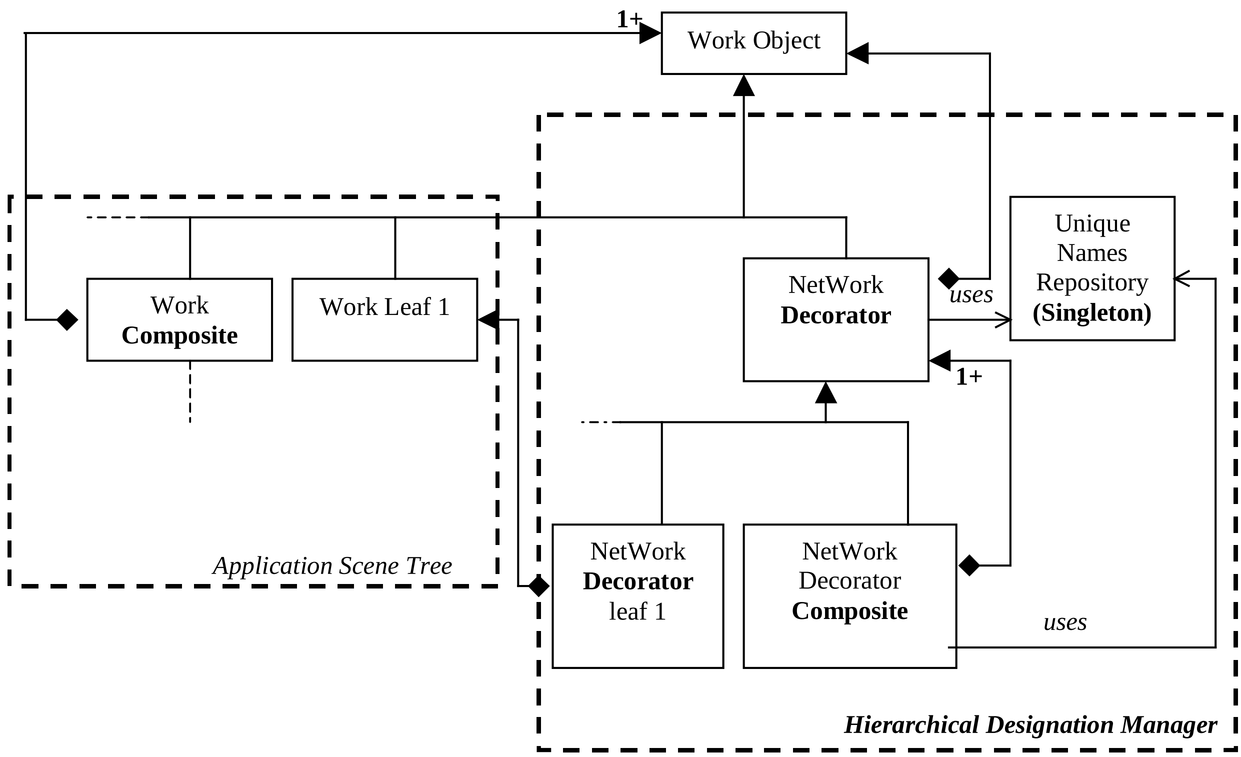


Figure 2: Distributed scene tree with unique names

Structure



Participants

Applicative part

- Work object:
  - Abstract work object class that defines all the generic attributes and methods that a VR work object may have (position, orientation, color, …).
- Work Leaf1 (Ventil, Sensor, Switch):
  - An concrete applicative object that may be wrapped by a Network Object Decorator on demand
- Work Composite inherited objects (Electric, Hydraulic, Pump, Main Pipe):
  - Concrete groups of applicative objects.

Distributed part

- Network Decorator:
  - Abstract class reflecting the generic network mechanisms of the Work Object abstract class. For example, it can implement at this level the distribution of abstract Work Object methods like moveObject(), resizeObject(),…
- Network Decorator Leaf1 (VentilDecorator):
  - This object inherits from an abstract Work Decorator and wraps a concrete applicative object (Ventil) from the scene tree.
  - It furthers additional network control to abstract Work Object methods in order to update the replicates of the under-laying object on the network.
- Network Decorator Composite:
  - This group forms, with the Network Decorator, the composite part of the distributed object tree implementation.
- Unique Names Repository:

- This Singleton implements the local Unique Naming access methods (CreateUniqueName(), RemoveUniqueName(), IsNameDistant(), …) and is used by the Network Decorator objects. It also handles the dynamic construction of the global name list by storing the distant unique names received from the remote participants.

Collaboration

When entering a meeting, Network Decorator objects substitute local Application Scene Tree objects. On the others participants sides, Network Decorator are constructed and then create replicates of the distant objects. Note that the same network classes manage the local objects and their replicates. So there are as many concrete Network Decorators implementations as there are concrete applicative implementations.

Consequences

Network properties

- The first property is satisfied. The unique names are easy to create since the Unique Names Repository creates each name locally. Thus, a unique name can be created both during disconnected work and during meeting. A unique name is independent of the geographical position within the 3D scene. So, objects can move without changing their names. As an example, A Network Decorator Object can update its underlying applicative concrete Work Object position but its name at the network level remains the same. A unique name defines the position of the node within the global scene tree. Thus, a name is unique, in time and space, but defines also a position within the distributed scene tree. It is useful since the application retrieves directly from a name which node is the father. Moreover, an application can decide to accept unordered messages notifications and stack them until it has resolved all the dependencies between them. For instance, a Network Decorator Composite Object node notification may results in several concrete Network Decorator object notifications. As hierarchical information is attached to the notification, we can implement a best effort algorithm that wait for all dependent nodes notifications before it updates locally the replicates.
- The second property is satisfied since the object name is defined at creation time and remains unchanged all along the life of that object.
- Third property is guarantied because to create a name (*@IPB,stampB,1.3*) a user must be owner of the parent node (*@IPB,stampB,1*). So, the parent (a Network Decorator Composite object) must be present to create a child node. Using the local list of children, the parent guaranties that an active name cannot be reused. To be reused, the owner of the active name must be present and must delete the name. If the same name exists for another object, it can be only for a user that is neither present nor owner. So, it can only be a private space that contains the pending name. Moreover, the pending name will disappear when the user will reach a new meeting.
- Latest property enables a communication peer to change its IP address without any difficulty. Already created names remains unique because the probability that two machines use the same address to produce the same name (same local time and same random number) is close to zero.
- Moreover, existing works can be reused through first level nodes that do not contain any reference to a specific shared scene.

Structure considerations

- Network implementation is well decoupled from objects implementation: the Network Decorators does the network job, so almost no specific and complex network code obscures the existing code and therefore simplify the developer task.
- The existing applicative object structure must be stable: adding new applicative objects types means adding new Network Decorators. However, note that many generic features may be already implemented in the abstract Network Decorator class that reflects the Work Object generic methods augmented with network constraints. So even then, the effort of creating a new concrete Network Decorator class should be minimized.
- In some situations, the Network Decorators may have to know about the private internal attributes of their underlying applicative objects in order to be able to dispatch object updates or to construct the replicates. This problem can be solved in most cases by using the Memento Pattern [GAMM 94] which will enables to interface the applicative object with his decorator by furthering only a subset of the real object state to the decorator without violating encapsulation of this underlying object.
- Maintaining two different trees synchronized (a distribution one and an applicative one) is not easy. However, this structure allows to choose clearly what should be distributed and what should remains in the private local space.

- Direct object distribution may not be acceptable for performance reasons. In some situations, it is better to implement a Distributed Facade [BROW 99] instead of sending several dissociated object notifications. In our example, the distributed objects are lightweight objects that need to send few messages to their replicates. Moreover, for more abstract command notifications that doesn't apply specifically to an object structure, it can be necessary to implement a Command oriented pattern which can be expressed as a combination of the Command pattern and the Decorator Pattern (see Compounding Command [VLIS 99]). The decorator part will correspond to the network transmission of the wrapped command.

See Also
Decorator, Singleton, Composite, Memento, Command [GAMM 94], Compounding Command [VLIS 99], Distributed Facade [BROW 99]

A PATTERN FOR AUTOMATIC SCENE AGGREGATION

Name
Scene Aggregation Helper

Example
Collaboration requires moving easily between disconnected works and meeting works. When virtual prototyping, each participant improves during a disconnected phase different nodes of the global virtual scene. Thus, a disconnected workspace includes nodes from the previous meeting but also new nodes created during the disconnected phase. During a further meeting, each participant brings the disconnected nodes into the shared scene tree. Thus, different entering nodes reform automatically a shared scene tree.

Context
This service is present typically when mobile workers collaborate at a virtual prototyping. Merging of disconnected works must be supported while satisfying a minimal "consensus". It is obvious that disconnected works cannot provide a global specification that satisfies all of the prototype requirements. For example, two disconnected designers can produce two conflicting improvements. During a meeting, the system can compute the shared scene to check for requirement respect. Detected conflicts must be resolved by interactions between the designers and direct modifications of the shared scene. End-users do not want the system to resolve automatically the conflict since they want to negotiate and conciliate the different proposals. Despite it is not feasible to guaranty a global consistency for the different entering proposals, the system must at least guaranty that the different proposals are associated with correct functional locations to guaranty a minimal consensus on the shared work.

Problem
A participant must be able to bring pieces of the scene puzzle and gets a copy of the global scene. The different pieces must go to the right place of the puzzle. A right place is not only a 3D position but also a correct functional location within the scene tree. For each arrival, the distant participants must observe the added pieces. Moreover, each peer must recover from transmission errors and gets a fresh state for each node of the global scene.

Solution
An application peer creates a Network List Object (i.e. an object including all of the entering nodes, see further details in the structure section) to define the nodes entered by the local user. At the creation time, the Network List Object multicasts several list messages to the distant peers for creating distant replicas. Each list message can be viewed as a part of the whole Network List Object. When a peer received the total number of parts, it has a complete copy of the Network List Object. Several messages are required since UDP (User Datagram Protocol) limits the size of a multicast message.
A list message is defined as follows [*LIST*: V_L, *TotParts*, *NPart*, $(G_1,V_1),\ldots,(G_N,V_N)$] where V_L is the version number of the list, *TotParts* is the total number of parts forming that list, *NPart* is the part number and each couple (G_X,V_X) defines the unique name and the version number for an object X. Using *TotParts* parts, a peer announces the objects it enters into the meeting. Afterwards, the announcing peer multicasts an object state [*State*: V_L, G_X, V_X, T_X, S_X] for each object X of the list where T_X is the type of X and S_X is the state associated with

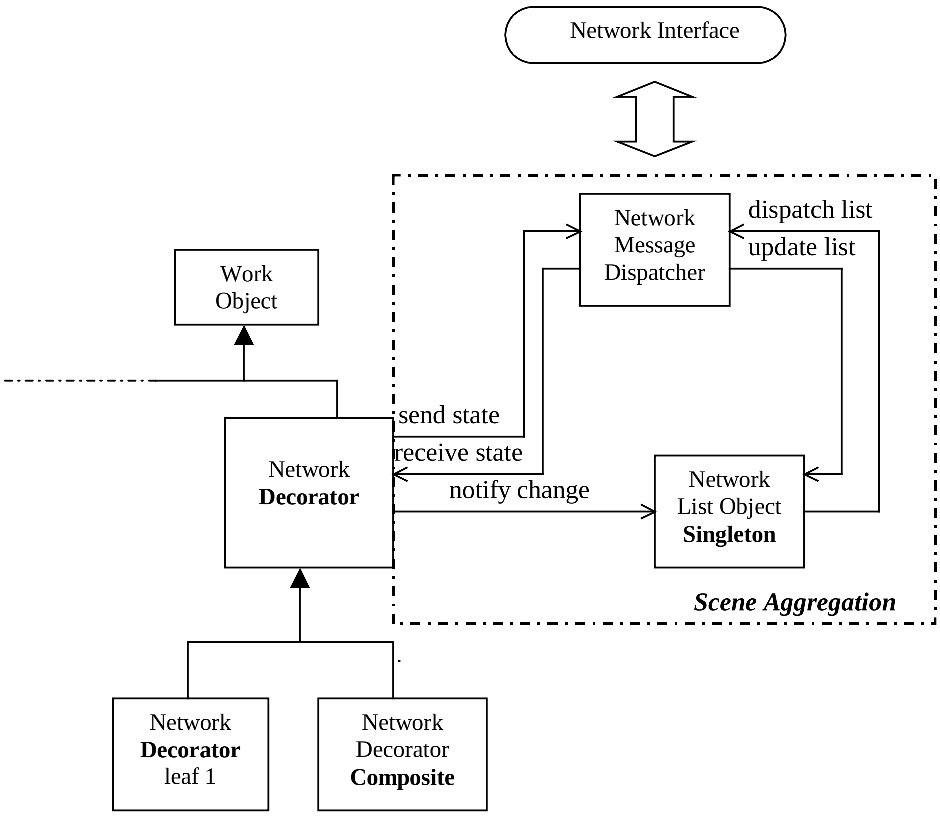
V_X. Thus, a distant peer recovers all of the announced objects. At the end of the scene aggregation, each peer has a copy of the shared scene.

Since a distant peer has the list of the objects, it can ask the retransmission of a missing object *X* by requesting a version number equal or higher to V_X. Thus, a producing peer has to retransmit uniquely the current version V'_X of *X* ($V'_X \geq V_X$).

Since each state [*State*: V_L, G_X, V_X, T_X, S_X] transports a list version V_L, a receiving peer that misses the corresponding list message can ask a retransmission by requesting the list part for the object G_X. Generally, the producing peer will retransmit the corresponding list part [*LIST*: V_L, *TotParts*, *NPart*, (G_1, V_1), …, (G_X, V_X), …, (G_N, V_N)]. When the producer added or deleted an object, the requested version V_L is no more available since the list changed. In that latter case, the producer retransmits all the new parts [*LIST*: V'_L, …] with $V'_L \geq V_L$. After receiving the new list, the requester has the responsibility of asking the retransmission of missing states based upon the received list.

It should be noted that a selective list [*SLIST*: V_L, T_L, *TotParts*, *NPart*, (G_1, V_1),…, (G_N, V_N)] transmits only object references of type T_L. A selective list is a subset of the list V_L.

Structure



Participants

- Network List Object:
  - There is as many list objects as meeting participants
  - It manages several attributes like (G_1, V_1),…, (G_N, V_N) and list version V_L
  - On demands, it furthers additional network control to update the replicates of the Network List Object on the network

The other participants have been described for the previous pattern.

Collaboration

Typically, a GUI uses the Scene Aggregation when the local participant decides to enter prepared works. Thus, a new Network List Object is created and transmitted to update the distant copies. Each Network Decorator (Leafs and Composite) uses the Network List Object each time a modification is processed on the corresponding objects (Leafs and Composite). Thus, each Network Decorator object modifies Network List Object of the local participant. On demands (from a local user or distant request), an update of the Network List Object is transmitted to the distant peers. For example, if a replica wishes to re-synchronize with fresh values then it sends a request for the Network List Object to check freshness of distant objects. When a Network List Object is newly created for a distant participant, the creation method uses the local Network List Object to update the entering participant with the local Network List Object. Since a communication peer receives the version number of the Network List Object but also of the different Work Objects, it can ask retransmission of missing list messages or object updates.

Consequences

Network properties

- Each entering peer brings pieces of the scene puzzle using the Network List Object. A couple (G_X, V_X) transmits a piece *X*. The entering participant gets a reply from distant participants. Thus, he completes the global scene as assembly of the different responses. At the end of the scene aggregation each participant a copy of the shared scene.
- Each piece *X* goes to the right place since the name G_X contains the location within the scene tree.
- Each time a participant enters, the distant copies of the global scene are updated using the list messages. Moreover, the responses give an opportunity for the copies to resynchronize and to recover fresher states.
- When a receiving peer detects a transmission error (missing announced objects or missing list parts), he asks for newer versions. Thus, retransmission only gives the fresh version of the missing object or missing list. Retransmissions of out-of-date states are thus avoided.
- Since the List Object is transmitted only on demands to the distant peer, the system does not retransmit redundant information i.e. the state of the Decorator Object that caused the List Object update.
- List Object gives information about the global progression of the corresponding participant. It enables a receiving peer to quickly check if the current state of a participant is fresh or if states are missing. It provides a basic support to further send negative acknowledgement about missing values.
- It shall be noted that a List Object can evolve while the local peer processes its Decorated Objects. The List Object evolves consistently with the current state of the peer. There is no need to locking the List Object while a peer processes its Decorated Object or vice versa.

Structure considerations

- Network implementation is well decoupled from any application processing: the cooperation between the Network List Object and the other Network Decorators does the network job. So, application code only have to create a List Object on user request to get a copy of the shared scene.
- As the List Object is processed either through local or distant request a Compounding Command [VLIS 99] can be used. The decorator part enables to wrap and transmit the command to a distant peer.
- The communication between the Network Decorator and the Network List Object can be implemented efficiently with Observer [GAMM 94]. In effect, the Decorator will be the subject and Network List Object will observes the notify updates.

See Also

Compounding Command [VLIS 99], Observer, Singleton [GAMM 94]

References

[ABDE 97] Abdel-Wahab,H., Favereau,J., Kim, O., Kabore, P. 1997. *An Internet Collaborative Environment for Sharing Java Applications*. IEEE Computer Society Workshop on Future Trends of Distributed Systems (FTDCS'97), Tunis, Tunisia.

[BARR 96] Barrus,J.W., Waters,C., Anderson,D.B. 1996. *Locales: Supporting Large Multiuser Virtual Environments*, IEEE Computer Graphics and Application. November, Vol.16, No.6, pp.50-57.

[BUSC 96] Buschmann F., Meunier R., Rohnert H., Sommerlad P., Stal M. *Pattern-oriented software architecture – A system of patterns*. Wiley, 1996.

[BROW 99] Brown, K., Eskelin P., Pryce N. *A mini-pattern language for Distributed Component Design.*
6th. Pattern Languages of Programs Conference. Urbana, Illinois. August 1999.

[BERC 96] Berczuk, S. *Organizational Multiplexing: Patterns for Processing Satellite Telemetry with Distributed Teams*. Book Chapter. Pattern Languages of Program Design 2. Vlissides, Coplien, Kerth Eds. Addison-Wesley. 1996. ISBN 0-201-89527-7.

[COST 2000] Costantini F., Sgambato A., Toinard C., Chevassus N., Gaillard F., *An Internet Based Architecture Satisfying the Distributed Building Site Metaphor*. IRMA2000 Multimedia Computing Track, Anchorage, Alaska, 21-24 May, conference proceedings, pp.151-155, published by IDEA Group Publishing ISBN 1-878289-84-5, 2000.

[BERC 2000] Berczuk S., Appleton B., Cabrera R. *Getting Ready to Work: Patterns for a Developer's Workspace*. 7th. Pattern Languages of Programs Conference. Monticello, Illinois. August 2000.

[GAMM 94] Gamma et.al., *Design Patterns: Elements of reusable Object-Oriented Design*, Addison Wesley, 1994.

[GREE 98] Greenberg S., Roseman M., *Using a Room Metaphor to Ease Transitions in Groupware*. Research report 98/611/02, University of Calgary, 1998.

[HAGS 96] Hagsand,O. 1996. *Interactive Multiusers VEs in the DIVE system*, *IEEE Multimedia*, Vol.3, No.1, pp.30-39.

[HLA 97] Defense Modeling and Simulation Office. 1997. *HLA Data Distribution Management Design Document Version 0.5,* February, U.S. Department of Defense, Washington D.C., http://www.dmso.mil/project/hla.

[HAND 97] Handley,M., Crowcroft,J. 1997. *Network Text Editor (NTE) A scalable shared text editor for the MBone*. Proceedings of ACM Sigcomm 97, Canne, France.

[HP 99] Hewlett-Packard. *CoCreate OneSpace: the Award-winning, Web enabled, CAD-Independent, real-time Collaboration Solution*. 1999.

[IBM 2000] IBM/Dassault Systems. June 2000. *Enovia Solutions*. http://www-3.ibm.com/solutions/engineering/esenovia.nsf/public/enovia_overview

[JSDK 99] Sun Microsystems. *Java Shared Data Toolkit*. http://www.sun.com/software/jsdt/index.html, 1999.

[LALA 98] Lalanda P. Shared Repository Pattern. Pattern Languages of Programs Conference. Monticello, Illinois. August 1998.

[SCHM 95] Schmidt D.C., Stephenson P. *Experience Using Design Patterns to evolve Communication Software Across Diverse OS Platform*. 9th European Conference on Object-Oriented Programming, Aarhus, Denmark, August 1995.

[TOIN 99] Toinard C., Chevassus N., Sgambato A., *Collaborative Virtual Reality and the Distributed Building Site Metaphor*, WSCG'99, The 7-th International Conference in Central Europe on Computer Graphics, Visualization and Interactive Digital Media'99, conference proceedings, pp. SP/111-117, 1999.

[VLIS 99] Vlissides J., Helm R. *Compounding command*. C++ Report, April 1999.

[YALE 99] Yales, C. *NetMeeting Knocking On Corporate Door*. August 1999. http://www.zdnet.com/eweek/stories/general/0,11011,2317429,00.html