Journal Pre-proofs


# MVC-3DC: Software Architecture Model for Designing Collaborative Augmented Reality and Virtual Reality Systems

Samir Benbelkacem, Nadia Zenati-Henda, Djamel Aouam, Yousra Izountar, Samir Otmane

PII: S1319-1578(19)30232-0
DOI: https://doi.org/10.1016/j.jksuci.2019.11.010
Reference: JKSUCI 706

To appear in: *Journal of King Saud University - Computer and Information Sciences*

Received Date: 16 February 2019
Revised Date: 4 November 2019
Accepted Date: 18 November 2019

Please cite this article as: Benbelkacem, S., Zenati-Henda, N., Aouam, D., Izountar, Y., Otmane, S., MVC-3DC: Software Architecture Model for Designing Collaborative Augmented Reality and Virtual Reality Systems, *Journal of King Saud University - Computer and Information Sciences* (2019), doi: https://doi.org/10.1016/j.jksuci.2019.11.010

# MVC-3DC: Software Architecture Model for Designing Collaborative Augmented Reality and Virtual Reality Systems

Samir Benbelkacem[a], Nadia Zenati-Henda[a], Djamel Aouam[a], Yousra Izountar[a,b], Samir Otmane[c]

[a]Division Robotique et Productique, Centre for Development of Advanced Technologies, Algiers, Algeria
[b]Déprtement d'informatique, University of Ferhat Abbas Sétif 1, Algeria
[c]Laboratoire IBISC, Univ. Evry, University of Paris-Saclay, Paris, France

sbenbelkacem@cdta.dz, nzenati@cdta.dz, daouam@cdta.dz, yousra.izountar@univ-setif.dz, samir.otmane@ibisc.univ-evry.fr

**Abstract**
In this paper, software architecture model "MVC-3DC" for Collaborative Augmented and Virtual Reality Systems design is proposed. This model is the results of merging several aspects: Human-Computer Interaction (HCI), distribution systems, computer-supported cooperative work (CSCW) and new technologies such as augmented reality and virtual reality. MVC-3DC integrates collaboration principles between remote users. MVC-3DC allows a low dependency between components such as the core functions, 3D graphics API and data distribution modes. The proposed architectural model integrates simulation models, SDKs and algorithms for different nodes involved in a collaborative session. This facilitates interoperability and capability to manage heterogeneity and relationship between different nodes participating in the collaborative session. Finally, our model makes it possible to integrate other toolkits without completely changing the structure of collaboration model. A simple adaptation could be made.

**Key-words**: software architectures; patterns; interoperability; augmented reality; virtual reality; computer-supported cooperative work.

## 1. Introduction

Both collaborative Augmented Reality (AR) and Virtual Reality (VR) systems provide a significant graphical potential for multiple users to interact with each other using 3D objects. For that purpose, these systems are increasingly being used to share and distribute 3D data between geographically separated and collocated users. It is, therefore, explicit, that systems which are intended to support collaborative AR/VR activities should be designed according to the tasks to be achieved and the intended users' social and cognitive characteristics. In the other hand, collaborative AR/VR systems design (Lukosch et al., 2015), (Huang et al., 2018), (Šašinka et al., 2019) becomes more complex since it must address heterogeneous 3D interaction software and devices as well as various network specifications (from high bandwidth on professional or experimental networks to low bandwidth on personal networks). Furthermore, researches only focus on technical issues of collaboration such as networking, synchronization, latency, to develop collaborative AR and VR systems. The lack of design models and interdisciplinary nature of AR and VR systems require design approaches providing a possibility to make changes in the design level before implementation level. Therefore, there is few suitable solutions for modelling, designing and then realizing distributed collaborative AR/VR systems.

These new generation of collaborative systems should be developed according to a design pattern that makes it possible to solve 3D interaction devices and network interoperability issues. Such a design pattern must be able to design software components supporting both hardware and software 3D graphics requirements. Existing solutions are, only, based on how to manage independence between components such as core functions and graphics API for classic (2D) collaborative systems.

For the time being, existing design models for collaborative AR/VR platforms deal neither with independence to 3D graphics API nor with efficient management of data distribution modes. In our case,

we need a software architecture model that is able to model objects independently from their distribution in the network. In addition, we should be able to model the virtual environment independently from the way it is represented to users. Finally, it is necessary to integrate heterogeneous software components (graphic, sound, etc.) connected to the hardware devices of the users.

Thus, our proposal consists in merging two main research fields, namely, augmented reality/virtual reality and collaborative work in order to propose an innovative solution, the *MVC-3DC* model, for providing a better approach to develop 3D collaborative systems. In our case, the proposed model should guarantee synchronization and independence of these systems from 3D graphics engines to make interoperability possible between such 3D engines.

In this paper, Section 2 presents a related work on software architecture models for collaborative systems. Section 3 presents MVC-3D software architectural model already proposed for 3D applications design. Section 4 presents our innovative collaborative software architectural model, *MVC-3DC*, and its implementation in the different distribution modes of a network. We explain how the proposed model offers interoperability between graphical libraries. Section 5 gives two examples illustrating how to implement our *MVC-3DC* model. Finally, Section 6 provides an evaluation of the effectiveness of the proposed distribution model.

## 2. Related work

### 2.1 Software architecture models for collaborative systems: groupware

A groupware is a software that allows a group of people that shares documents, remotely, in order to perform a collaborative work. Several architectural models were proposed for groupware design. Each of them proposed a method to deal with the group activity concept. The proposed models were used to develop classical and/or 3D user collaborative interfaces.

Patterson in (Patterson, 1994) proposed ZIPPER model. It is based on the shared states notion. For the author, a groupware can be decomposed into four states-levels that represent levels of abstraction: (1) Display state corresponds to the state of the input and output devices (monitor, mouse, etc.), (2) View state corresponds to the state of the data logical presentation, that is, the state of the user interface, (3) Model state corresponds to the functional kernel and domain objects, and (4) File state corresponds to the persistent representation of the model. These states can be instantiated into three modes: shared, synchronized and replicated.

Dewan (Dewan, 1999) introduced Dewan meta-model which is a generalization of both Arch model (Bass et al., 1992) and Zipper model (Patterson, 1994). According to this meta-model, a groupware is composed of a variable number of layers representing several levels of abstraction. The highest layer (level N) is semantic while the lowest layer (level 0) is hardware. Compared to Arch model, the highest layer corresponds to the Functional Core and the lowest layer corresponds to the Physical Interaction Component (Laurillau, 2002). The global architecture is composed of a root and several branches. The root is composed of shared layers (levels L+1 to N). The branches are composed of replicated layers (levels 0 to L), connected to the root at the layer *"L"*. In addition, the objects managed by the branches are private. Conversely, the objects managed by shared layers are public.

Abstraction-Link-View (ALV) model was proposed by Hill (Hill, 1992) which is based on a multi-agent approach. Here, the facets were defined as follows: (1) shared Abstraction (A), manages domain objects, shared by all users in the collaboration space, (2) replicated View (V) interprets a user's inputs and manages the outputs. Events generated by the interaction are processed at the View level by dedicated

functions that modify locally the data. (3) Link (L) has the task of linking the Abstraction facet with one View facet. Also, it ensures that local data of a *View* (V) are conform to the corresponding data's representation in Abstraction facet.

Salber (Salber, 1995) proposed an extension of PAC-Amodeus model (Nigay, 1994) called *CoPAC*. It is a PAC agent (Coutaz, 1987) enhanced by a communication component. It allows collaborative agents (users of the collaborative workspace) to communicate with each other directly. Control (C) facet plays the role of distributing messages from both Abstraction (A) and Presentation (P) facets to the Communication facet and vice-versa.

In (Tarpin-Bernard, 1997), AMF-Collaborative design pattern was proposed to design mixed reality collaborative systems. The proposed model is derived from AMF model (Multi-Faceted Agent) proposed in (Ouadou, 1994). AMF model, compared to PAC model, is not limited to the Abstraction (A), Presentation (P) and Control (C) facets. It adds new facets such as the facet dedicated to error processing. These facets interact with the Control (C) facet. AMF-C design pattern is composed of AMF agents distributed on different nodes according to two strategies: fragmentation of agents and replication of agents.

Furthermore, PAC* model (Calvary et al., 1997), (Laurillau, 2002) introduces a functional decomposition at the PAC agent using the three basic spaces of the 3C model (Ellis, 1994): communication, coordination and production. The three facets of a PAC agent are decomposed into three parts. Each part is dedicated to a dimension of the 3C model. For example, the Presentation facet is composed of three sub-facets dedicated to communication, coordination and production. Therefore, an agent covers a functional space, according to two orthogonal dimensions: the three facets of the PAC agent and the three dimensions of the 3C model.

Clover model (Laurillau and Nigay, 2002) combines the Dewan generic architecture metamodel for groupware (Dewan, 1999) with the Arch reference model (Bass et al., 1992). It defines six levels of abstraction. Here, the core is divided into two levels: Private Functional Clover and Public Functional Clover. The public functional clover manages all objects of the domain common to all users. The private functional clover manages all the objects of the domain specific to each user. The specific feature of this model is that its functional core encapsulates three subcomponents integrating the three facets of the 3C model: communication, coordination and production.

C4 model is an agent-based collaboration formalism proposed in (Khezami et al., 2005), dedicated to collaborative tele-operation through Internet. It is based on PAC* model. Three agents were designed according to the three spaces of the 3C model, to preserve the modularity of the system. Besides these three spaces, this model implements a fourth agent, a Collaborator Agent, that encapsulates the three precedent agents. The collaborator agent deals with the exchange of data between its counterparts. A collaborator agent provides two main functions. First, it allows a communication between the three dedicated agents inside the collaborative agent. Second, it establishes a direct interaction between each dedicated agent with its correspondent of another collaborator agent. This direct communication between the dedicated agents of the same collaborator agent, and with their correspondents of the other collaborator agents, makes it possible to increase the efficiency of the system. This avoids the bottlenecks effects due to centralization.

Fleury proposed PAC-C3D (Fleury, 2012) model based on PAC design pattern to develop collaborative virtual environments. This model integrates three distribution policies into PAC's Abstraction model to manage collaboration process in virtual collaborative environments. The role of each policy is: (1) referent distribution policy, manages the set messages and distributes updates to other Controls, (2) proxy distribution policy, transmits the set messages toward a referent Control, and manages the update

messages returned by a referent Control, and (3) duplicated distribution policy, combines the management of the set messages of the first distribution policy and the management of the update messages of the second distribution policy.

**2.2 Synthesis of groupware-based collaborative models**

Several works on architectural models' design for collaborative human-computer interaction (HCI) have been established. They, essentially, deals with separating functional core from graphical interface of interactive applications. These latter are, essentially, decomposed into three kinds of components (facets) that should be as independent as possible from each other.

Zipper model offers a global decomposition of collaborative systems. The model is rather vague on this point, since the management of the dialogue is established by Model (M), View (V) and the synchronization of different views. Here, no difference was made between single-user and multi-user functions. This is because the model reasoning is based on states not on functions. In the other hand, Zipper model does not consider functions related to production, communication and coordination.

Dewan architecture, which is a generalization of Arch model, is less accurate than Arch model since it does not detail the functional role of each layer. The model inherits from the Zipper model the notion of replicated and shared layers. Indeed, the meta-model defines a degree of collaboration that identifies the replicated layers implementing collaborative aspects. It, consequently, identifies the functions of collective actions. Nevertheless, the meta-model makes no distinction between functions related to production, communication and coordination. Dewan meta-model considers, as in Zipper model, that public components (i.e. shared layers) are allocated to centralized processes. But, these public components can, at the implementation level, be allocated to synchronized replicated processes. Moreover, the meta-model designs an architecture without public components, which is inconsistent for a conceptual architecture of a groupware. Without, at least, one public component, there is no possible collaboration.

ALV model is a variation of PAC model. But it does not advocate any rule to structure View and Abstraction facets. Also, this model does not differentiate components dedicated to individual actions from those dedicated to collective actions. ALV model clearly considers collective resources, managed by shared Abstraction, and individual resources, managed by View component. Nevertheless, this approach seems restrictive since shared Abstraction and global View are necessarily public and private components respectively. Also, this model confuses conceptual and implementation aspects: shared Abstraction is necessarily allocated to a centralized process (server) and Views are allocated to distributed processes (graphic terminals). For the observability of resources and actions, the model allows an exchange, only, between private components (Views) and the public component (shared Abstraction). There is no possible exchange between private components (Views). This model allows a user to have private data at View level only. This is a form of observability of individual resources.

Clock model is similar to ALV model since the components of the Functional Core are necessarily public, and the components composing the interface are private. Clock model does not propose a functional decomposition that separates functionalities of an individual action from those of a collective action. Also, components composing the root are clearly identified as public and components composing the branches as private. In the same way as ALV model, Clock model considers private components which composes the interface. Clock language offers a complete set of services to ensure interaction between public and private components. However, these services are programming-oriented. It is the same for the observability of context resources. The Clock language is used to specify which are public and private data. This property is not explicit, and the task is delegated to a programmer.

For CoPAC model, each PAC agent that populates the dialogue controller can be public or private. This approach, unlike, previously discussed models, allows all possible combinations to make components public or private. Using CoPAC model, the difference between a collective action and an individual action is explicit since the model dissociates single-user PAC agents from collaborative CoPAC agents. Moreover, collective action dedicated to communication is made explicit by the Communication facet. Collective actions dedicated to coordination is delegated to Control facet. CoPAC model considers, clearly, the notion of public (common) and private (decoupled) components. For the observability of actions and resources, the model proposes an exchange protocol between CoPAC agents through the Communication facet. The latter is responsible for broadcasting messages coming from Control facet and inversely.

AMF-C model provides a fine granularity functional decomposition than PAC model, since it defines new facets for an agent. It focuses on the development of distribution strategies and does not propose a functional decomposition to consider the group's action in the sense of 3C model. Also, AMF-C defines public and private components using a shared agent and a local agent respectively. Chalon in (Chalon, 2004) merges AMF-C with IRVO model to design collaborative mixed reality applications.

PAC* model refines CoPAC model by considering the three functional spaces of the 3C model: communication, coordination and production. In addition, PAC* model uses Dewan's model to define an exchange protocol between agents of the same hierarchy or between two instances of a hierarchy. PAC* is the only one that decomposes a collective action into communication, coordination and production. Comparing to PAC model which makes explicit individual resources, PAC* model makes explicit collective resources. The problem is that, PAC* does not propose an explicit solution to implement interaction coupling. Also, PAC* does not provide any solution to consider the observability of context resources.

Clover model offers a modular implementation that satisfies the properties of reusability and scalability. It, therefore, reduces the development cost. This model does not fix the number of layers that can exist. Other layers can be added to increase the separation of the functionalities and to satisfy the modularity of the code, which makes the software architecture open (Maeda et al., 1997).

C4 model introduces the concept of collaboration agent which ensures the relationship between collaborator agents of the distributed system. It decomposes a collaborative action according to 3C model. A major drawback of this agent is that it is a passive agent, pre-programmed to manage the internal and external agents of each collaborator agent of the system. The collaboration agent is not able to look for new resources to provide new behaviours in the system.

PAC-C3D model is an adaptation of PAC design pattern to develop collaborative virtual environments. This model addresses collaboration from development point of view. It focused on the implementation aspect of the collaboration rather than the architectural aspect. From architectural point view, the proposed model is driven from PAC model. So, it inherits the same problems of PAC model. In fact, Virtual reality features and collaboration concepts are not integrated in their architectural model, but only considered during the implementation of a collaborative application. Finally, PAC-C3D model is only implemented for collaborative virtual environments. No indications were given on the possibility to use this model to support collaborative augmented reality systems.

Through this review, we discovered that it is necessary to provide a model that integrates both augmented reality, virtual reality and collaboration concerns, early, in a software design method. Also, the proposed model would allow a best separation between functional core, processing (SDKs and libraries) and visualization (3D graphics) in a collaboration process. The proposed model should provide an explicit solution to integrate communication, coordination and production actions, as well as the data distribution modes in a collaborative platform.

## 3. MVC-3D for collaborative 3D applications

### 3.1 MVC-3D Design Pattern

The benefit of the classic MVC pattern is the separation of both interaction and visualization aspects of the user interface when designing an application. Also, it enables modular designs with which changes performed on a component doesn't affect other components. In practice, MVC components have a different behavior; View (V) defines the concrete syntax of the application, i.e. the input and output behavior of the application as perceived by the user. Model (M) corresponds to the semantics of the application; it implements the functions that the application is able to perform. Controller (C) handles the user actions and maintains the mapping and the consistency between the abstract entities (Model) and their corresponding concrete entities (View). Controller embodies the boundary between semantics and syntax and connects them. From, application, point of view, MVC pattern model has shown its effectiveness since it has been used to develop web applications, and could be adapted to develop tangible interfaces for VR systems (Benbelkacem et al., 2018).

In (Benbelkacem et al., 2018), we proposed and adaptive MVC-3D design pattern to develop both Augmented Reality and Virtual Reality systems. In fact, MVC's components has been extended to integrate AR/VR concerns: (1) we introduced "Library" component to the MVC model (see Fig.1) which encapsulates all processing including simulation models (e.g. Matlab/Simulink), AR/VR algorithms (e.g. tracking techniques, gestures, faces and speech recognition algorithms), and SDKs, to process the amount of data provided by the View (V) via the Controller (C). (2) We extended "View" component by introducing a sub-component that captures the surrounding real environment of the application and a sub-component that integrates the sensors module and manages the tangible objects (physical and digital). We obtained interactive View (iV) facet. Fig. 1 shows the structure of our MVC-3D model.

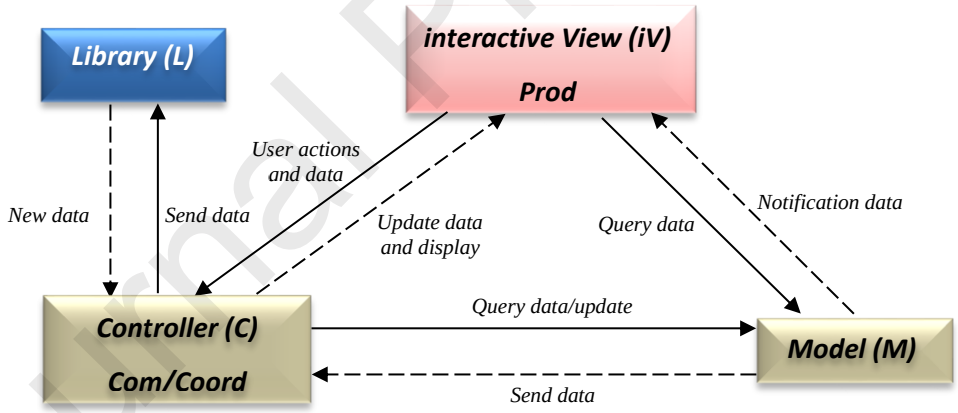


Fig. 1. MVC-3D Model.

### 3.2 Collaboration and the 3C model

In order to consider the principle of collaboration, our proposal is based on the *3C functional model* proposed by Ellis (Ellis, 1994), shown in Fig. 2. According to this functional model, a groupware system composes three domain specific functions: communication, coordination and production/cooperation.

Communication space allows actors exchanging information where the system acts as a messenger. Coordination space defines the actors and their social structure, as well as different tasks to be carried out in order to produce results. Production space represents the result of common tasks to be achieved from the activity of the group (ex: word document, paint, etc.).

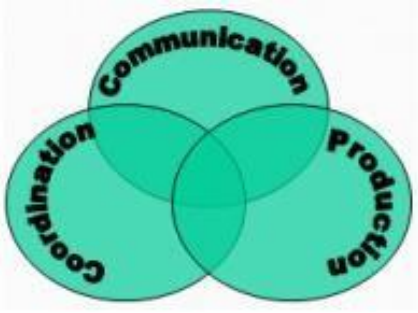


Fig. 2. 3C model by Ellis.

In this paper, we combine MVC-3D pattern with 3C model to propose a groupware for designing collaborative augmented/virtual reality systems. MVC-3D model (Benbelkacem et al., 2018) will be the foundation of our proposal. This model can be easily adapted to collaborative systems, because the aspects related to the collaboration can be integrated in the form of independent processes. Each process can manage the distribution of its data in the network. In addition, we believe that data distribution modes used should not affect the *interactive View* (*iV*) nor the *Model* (*M*). In the same way, the 3D graphics' details should be limited inside the *Interactive View (iV)* and the core concepts should remain in the *Model*. From another point of view, the programmer will not be forced to adapt the code describing the objects or their representations to the chosen data distribution mode.

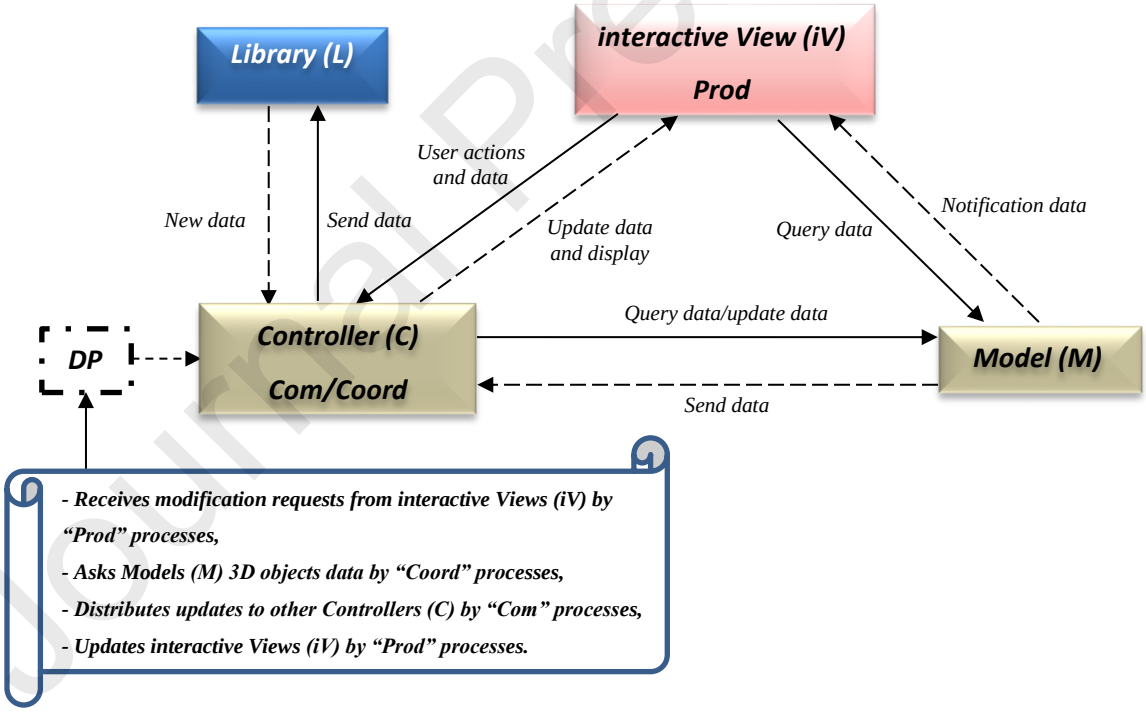


Fig. 3. MVC-3DC model.

Therefore, we introduce a new software architectural model, *MVC-3DC,* that extends MVC-3D pattern (Benbelkacem et al., 2018) to support 3D Collaborative Augmented and Virtual Environments. This model integrates both 3C model (Ellis, 1994), (Cheaib et al., 2012) and distribution policies (Fleury et al., 2010) by maintaining a low dependency between the functional cores, distribution modes, and 3D graphics API

used for visual representations. Our collaborative model provides the possibility to integrate heterogeneous 3D graphics API into different nodes involved in the same collaborative session. Interoperability between these 3D graphics API could be guaranteed.

MVC-3DC's architecture is presented in Fig. 3. Here, a **Distribution Policy (DP)** is integrated into Controller component. Its role (see Fig. 3) is to ensure distribution of 3D data and tasks within the network when users interact with their viewers.
We specify for each facet of MVC-3DC model the objects' functionalities of both real and virtual worlds. The role of each facet is defined as follows (see Fig. 4):
- Associate to Controller (C) facet two collaborative processes: *"Comm"* process for communication and *"Coord"* process for coordination. *"Comm"* process communicates information about modification requests of 3D objects' states, 3D scene or users' behaviour. *"Coord"* process interacts with *Model (M)* to receive object's data and perform modification on these objects. It also receives and sends update messages to other "Coord" processes of the network.
- Associate to interactive View (iV) *"Prod"* process for production of results. Its role is to adapt user's 3D interface to update messages sent by the *"Coord"* process.
- Model (M) and Library (L) do not include any process.

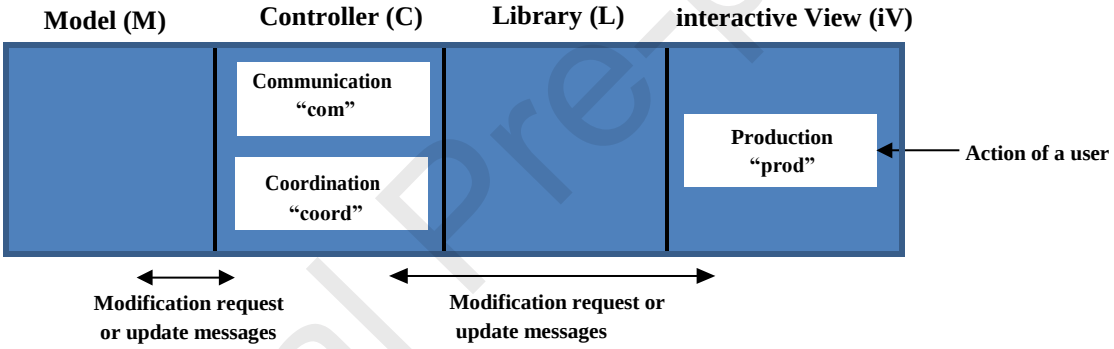


Fig. 4. Definition of facets for MVC-3DC model.

Each object of the virtual world is considered by these facets in different ways. This is described by as follows:
→ *Model (M)* stores an object's data on augmented reality/virtual reality worlds (ID, types, ID markers, position, orientation and 3D transformation matrix). *Model (M)* communicates with *Controller (C)* through *"Comm"* and *"Coord"* processes. If a user performs an action on an object, "*Comm*" process of corresponding user's node requests *"Coord"* process of the same node to report a modification request on the object. Here, a verification operation is performed to authorize the modification of the object's parameters. *"Coord"* process sends modification request to *Model (M) to obtain object's data*. Then, *"Coord"* process performs modification on the object and send update messages to *Model (M)*. In addition, *"Comm"* process sends then update messages to other *"Comm"* processes of *Controllers (C)* associated to different nodes of the network.
→ *Controller (C)* manages modification requests and object parameters of augmented/virtual worlds through *"Comm"* and *"Coord"* processes. In fact, *Controller (C)* component communicates with *Model (M)* in order to update the object subject to modification request. Also, it distributes these update messages to different nodes of the network.

→ *interactive View (iV)* offers a dedicated representation of augmented/virtual worlds. For instance, for a 3D world composed of 3D objects, its *interactive View (iV)* corresponds to the elements of the scene that graphically represent these objects. In order to obtain the 3D world consistency, the different interactive views integrate, via "*Prod*" processes, 3D transformation matrixes of objects making it possible to display several augmented/virtual views adapted to users' movements.

MVC-3DC makes it possible to design a collaborative augmented/virtual reality system with low dependency between the functional cores, distribution modes and 3D graphics API used. Our model provides an explicit communication between these components. This independence through MVC-3DC model is structured as follows: Model (M) encapsulates data, functionalities and modelling of augmented/virtual reality applications. Controller (C) handles the user actions and plays an intermediate role between interactive View (iV) and Model (M). Library (L) encapsulates all processing using specific and heterogenous AR/VR algorithms (e.g. tracking techniques, gestures, faces and speech recognition algorithms), complex and simulation models (e.g. Matlab/Simulink) and SDK/toolkits to process the amount of data provided from interactive Views (iV) via the Controller (C). The latter returns updates, to all interactive Views of the collaboration session. If we plan to change an algorithm or a toolkit, we just modify the Library's content without completely changing both Model and Controller's contents. interactive View (iV) represents the visual items of the augmented/virtual reality application. It, also, introduces a sub-component that captures the real environment model of the application and a sub-component that integrates the sensors module and manages the tangible objects (physical and digital).

MVC-3DC model ensures interoperability between multiple interactive Views. Any changes made in one viewer is reflected in others. The Controllers of the same collaborative session is responsible for sending the updates of the virtual environment to their own interactive Views described by heterogeneous 3D graphics API. Therefore, Model, Controller, Library and interactive View facets communicate in order to distribute and share 3D data and 3D tasks performed by one user of the collaborative session. For instance, when a user performs an action on its interactive View, his modification request is sent to Controller facet. The latter asks its Model for the corresponding 3D object's data. After receiving these data, Controller performs updates or interacts with Library to perform updates of the corresponding 3D object. The Controller transmits these new data to the Model. It also sends them to other Controllers of the collaborative session. Each Controller update locally its corresponding interactive View.
Finally, Library (L) allows all the objects of the virtual environment to be represented in a different way in each interactive View of the collaborative session. To do this, Library, integrates, in each node, suitable software components, adapted to the hardware devices that each user uses.

**4. Distribution modes using MVC-3DC**

The virtual environment's data storage and processing is an important issue to consider when designing collaborative AR/VR applications, since it is necessary to ensure consistency between users' visual renderings. The challenge is to determine which nodes (usually the users' computers) store these data, which nodes process them and how the synchronization of the distributed data is achieved. We can differentiate three data distribution modes to overcome this challenge: centralized mode, duplicated mode and hybrid mode. The idea of our approach is driven form approaches presented in (Fleury et al., 2010) and (Dewan, 1999).

We show, in this section, the ability of MVC-3DC facets to deal with the three distribution modes when implementing collaborative augmented reality/virtual reality systems. We consider that, when a user makes an action upon a shared object using his interface, this modification, from software architecture point of view, occurs on *interactive View (iV)* component. Based on this hypothesis, we describe the behaviour of MVC-3DC's components during collaboration process and the relationship between these components.

**4.1 MVC-3DC in centralized architecture**

In this mode, all virtual objects' data that compose the augmented/virtual environment are stored and processed in the server side. Modification requests of the objects are all sent from clients (nodes) to the server, which sends them updates.

For each shared virtual object with a typical centralized architecture, we have associated to each node the following facets: *Controller* and *Interactive View*. We associate to the server the facets: *Controller*, *Library* and *Model*. Thus, there is only one instance of *Model* on the server. There are as many instances of *interactive View* as there are visualization nodes embedding one or several representations of the shared augmented/virtual environment.

To formalize the collaboration process according to the 3C model in centralized architecture, we integrate:

- « CommN » et « CoordN » processes in *Controller* facet of a node,
- « ProdN » process in *interactive View* facet of a node.
- « CommS » et « CoordS » processes in *Controller* facet of the Server.

The exchange between the facets of MVC-3DC for centralized distribution will be as follow: When a user performs an action on an object, this means that the user performs an action on its *interactive View*. This facet communicates the user's modification request to *Controller* facet. *"CommN"* process (belonging to *Controller* facet of user's node) sends modification request to *"CommS"* of the server. This request contains the identifier *(ID)* of the augmented/virtual object and the modification action. Then, *"CoordS"* process asks its *ModelS* for object's data. After receiving these data,*"CoordS"* performs updates or interacts with *Library* to perform updates of the corresponding object. Then, *"CoordS"* transmits these new data to the *ModelS*. *"CommS"* sends new data, by an update message, to "*CommNi*" processes of different nodes (this sending should be asynchronous). *"CoordNi"* of nodes perform updates of the object locally. Then, *"CoordNi"*, sends these update messages to *"ProdNi"* processes. For each node, *"ProdNi"* asks its *interactive View* for an update in order to display to users the updated scenes. Fig. 5 describes a typical centralized architecture with one server and two clients.

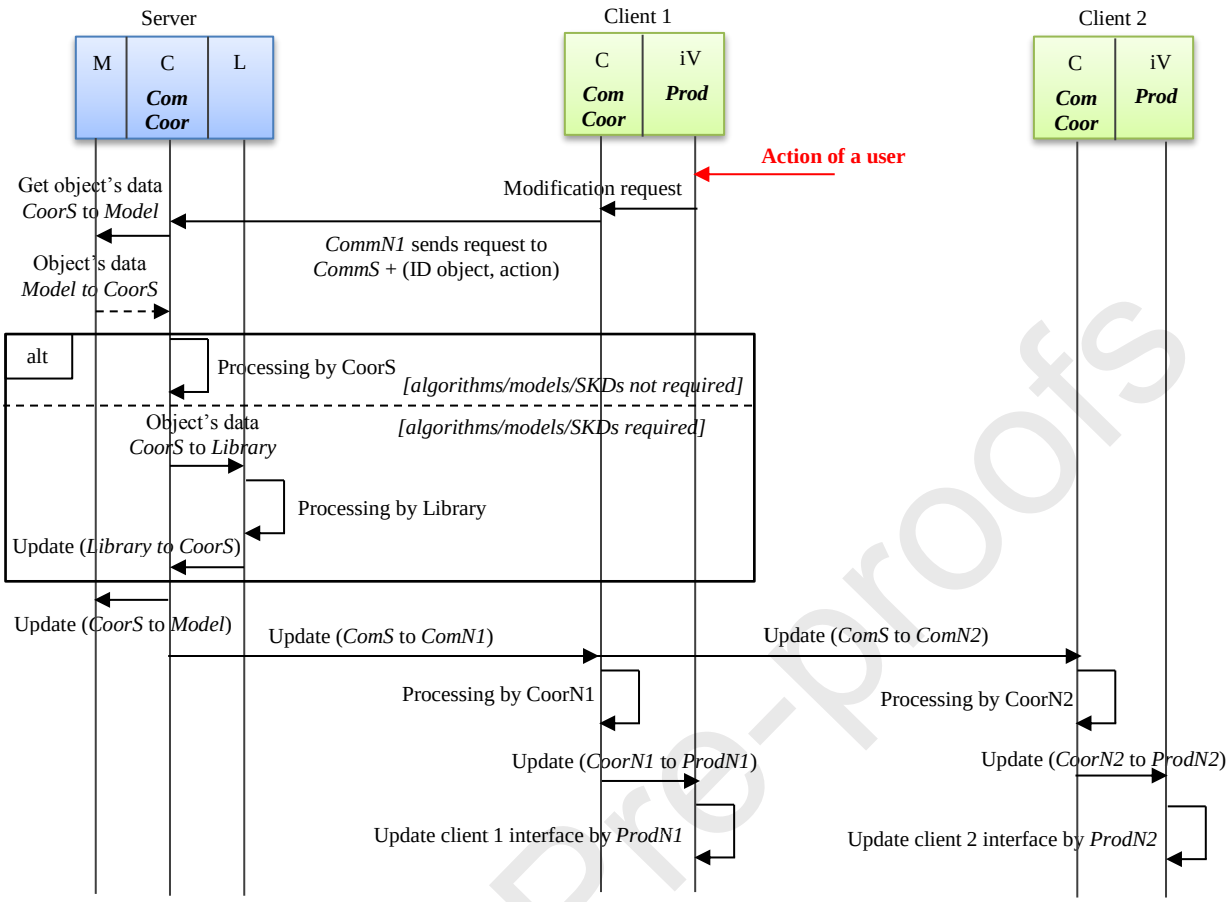


Fig. 5. Instantiation in centralized mode with MVC-3DC model.

The main benefit of this architecture is that the users may perceive the result of modification at the same time, but the problem concerns the feedback delay which can reach twice the network latency. Another interesting property of this distribution policy is that it is not necessary to ensure a strong synchronization between the nodes since the modification requests are executed on the server. Finally, it is easy to allow several users to interact at the same time with the same object. Here, the server centralizes all the modification request coming from all nodes, then transmits the update state of the modification to all the nodes, including the one which has asked for modification. This method ensures a strong consistency between the nodes participating in the distribution process, and makes it possible to avoid duplication of 3D data. However, this architecture presents two principal problems: (1) a network latency may appear between the action performed by a user on its proper node and the modifications operation of the virtual environment performed by the server, (2) when the number of users increases, a bottleneck may appear in the server.

### 4.2 MVC-3DC in duplicated architecture

In this mode, each node stores and processes objects locally. The node where the modification is performed (action of a user on augmented/virtual environment) is responsible for sending updates to other nodes of the network. To ensure the same update for all nodes, they must be synchronized to each other in order to obtain the same execution speed of the object's behaviour on each node. Fig. 6 presents a typical duplicated architecture with three nodes.

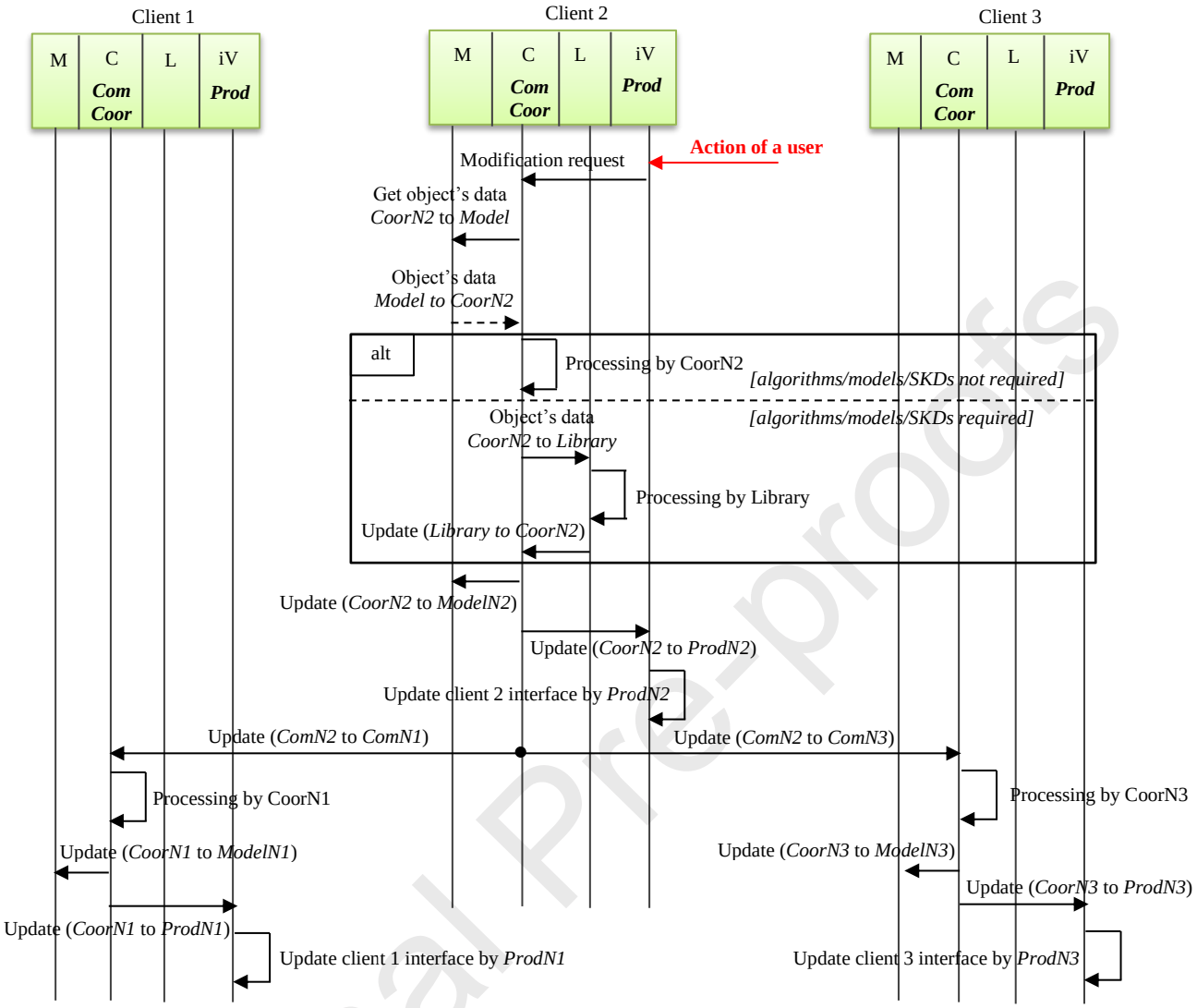


Fig. 6. Instantiation in the duplicated mode with MVC-3DC model.

When the user performs interaction with its *interactive View* (we take an example of client 2 corresponding to node 2), this facet communicates the user modification request to *Controller* facet. This request contains the object's *ID* and the modification action. By *Controller* facet, *"CoordN2"* process transmits the modification request to its *ModelN2* to receive the object's data. After receiving these data,*"CoordN2"* performs updates or interacts with *LibraryN2* to perform updates of the corresponding object. Then, *"CoordN2"* transmits these new data to its *ModelN2*. It, then, sends these update messages to *"ProdN2"* process that asks its *interactive View* for an update of user's scene. Furthermore,*"CommN2"* sends new data, by an update message, to "*CommN1*" and "*CommN3*" processes. *"CoordN1" and "CoordN3"* perform updates of the object locally. They send new object's data to corresponding *ModelN1* and *ModelN3*. After, *"CoordN1" and "CoordN3"* send update messages to their corresponding *"ProdN1" and "ProdN3"* processes. *"ProdN1" and "ProdN3" ask* their corresponding *interactive Views* for an update in order to display to users the updated scenes.

In this method, the number and the size of messages transiting in the network are lower since only request and update messages are exchanged. In addition, there is no latency when the user interacts with its virtual model (he obtains an immediate feedback) because the modification request is processed locally. Thus, the changes are performed, at first, on its local copy before being sent to others by update

messages. The other users may perceive the result of the interaction at the same time, with a delay corresponding to the network latency. This approach is not sufficiently flexible, especially if users want to add new objects that are not planned in the initial database. Also, it is difficult to manage user access rights to virtual objects. For example, when two users try simultaneously to manipulate the same object locally, the conflict appears when the object is updated. It is therefore necessary to set up mechanisms to manage the concurrent access of users to virtual objects. Finally, the virtual environment may become inconsistent between nodes due to latency and loss of data in the network during updates.

### 4.3 MVC-3DC in hybrid architecture

To implement the hybrid mode, all object's data are stored and processed in a specific node. The modification requests of the other nodes are sent to this node, which then updates all the other nodes participating in the session.

In order to answer user's modification request in real time, it is interesting to locate the *Model* of object on the node concerned by the modification. In this case, there will be two different situations while interacting with an object, as described Fig. 7 and Fig. 8: either the *Model* of the object is on the same node than the user, either it is on another node.

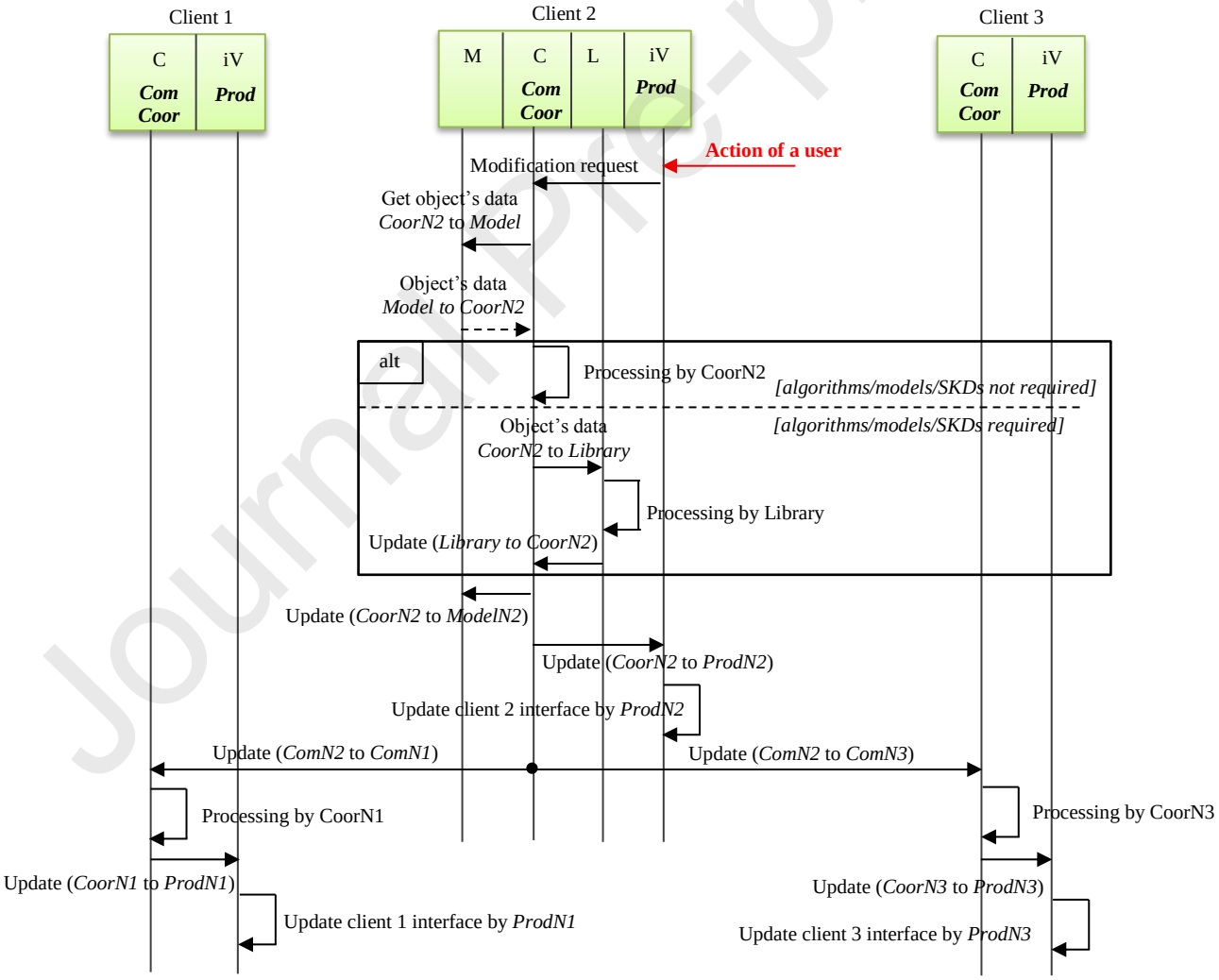


Fig. 7. Instantiation in the hybrid mode with MVC-3DC model (first configuration).

If the *Model* of the object is on the node of the interacting user, this user will obtain an immediate interaction feedback. The other users will perceive the interaction with a small delay due to the network latency. This is very similar to the behaviour of the duplicated architecture. Fig. 7 shows this first configuration.

If the *Model* of the object is not on the node of the interacting user, the node which integrates the *Model* will be asked for access and exchanging data. This node will be the first which perceives the result of the interaction. All the other users (even the one who is interacting) will see the result of the interaction at the same time, but, with a delayed feedback. This second configuration (see Fig. 8) is similar to the behaviour of the centralized architecture.

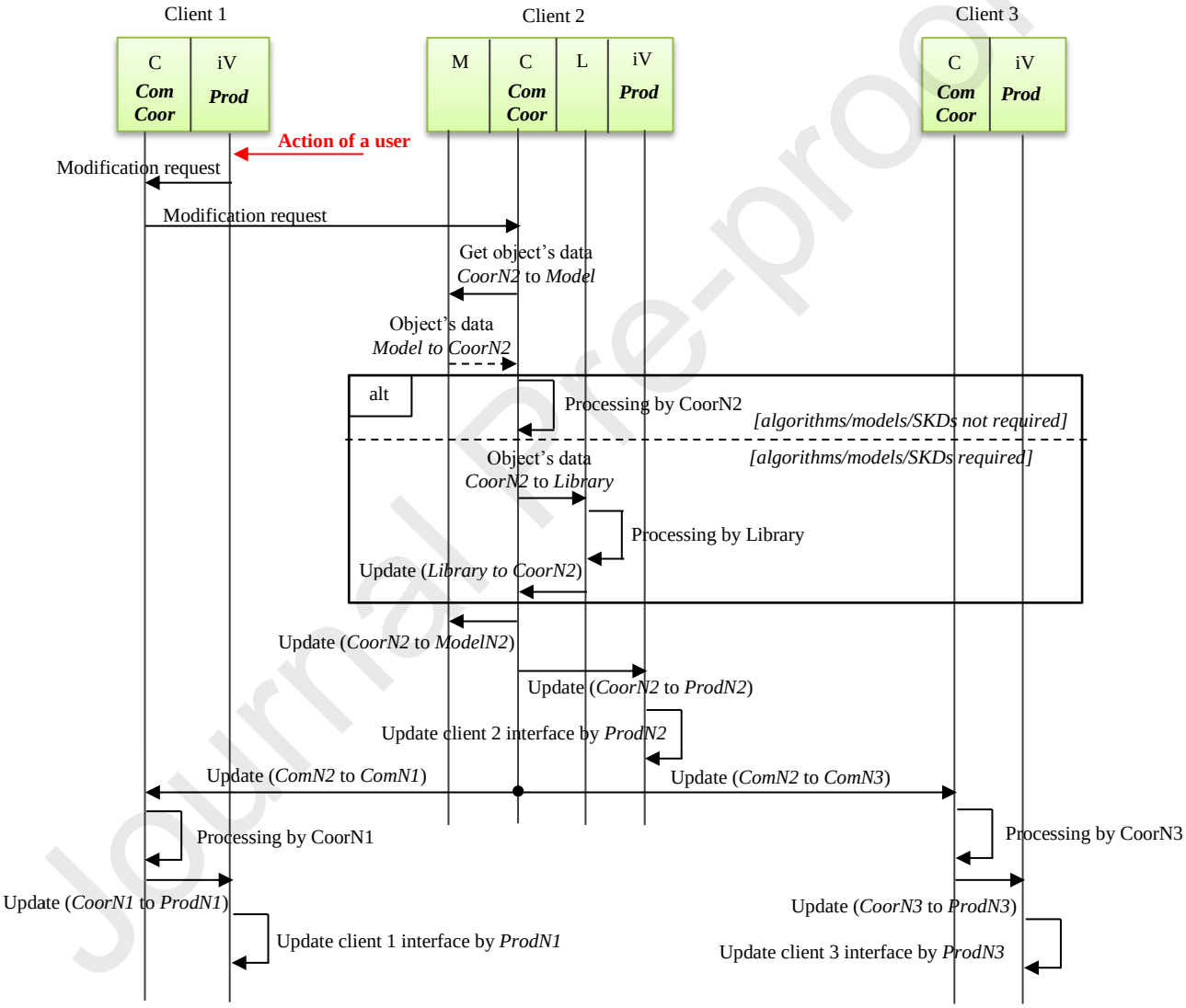


Fig. 8. Instantiation in the hybrid mode with MVC-3DC model (second configuration).

In both cases, this hybrid solution enables several users to interact at the same time with a same object. But it is necessary to synchronize all the nodes.

The hybrid method can be extended to a very large number of users or to a large volume of 3D data that it is not necessary to duplicate for each user. But, the communication cost increases when sending additional 3D data and ensuring consistency. Also, when the state of a 3D object is changed on a user's node, update messages should be sent to all the users of the distributed application even if some of them do not view the concerned 3D object. Finally, if there is a large number of nodes, a user may not know which users modified their own virtual objects. Works presented in (Lee et al., 2007) proposed how efficiently duplicate missing data without disrupting the user's immersion in the virtual environment.

### 4.4 Distribution policies adaptation

Switching from a distribution mode to another according to our MVC-3DC model is possible. For example, to transform a centralized architecture to a hybrid architecture or to a duplicated architecture, we only change the distribution policy of *Controller* facets by replacing its current distribution policy by a new one. This adaptation affects neither *Models*' contents nor both *interactive Views* and *Libraries*' contents.

In this case, we can migrate/duplicate *Models* and *Libraries* components from one node to others and then changing the distribution policies of the *Controllers*. This situation allows a user to directly interact with virtual objects after integrating *Model* and *Library* on its node.

For example, to switch from centralized mode to duplicated mode, both *Model* and *Library* facets located in the server are duplicated to all the nodes of the distributed system. Then, we replace the *Controllers* of the different nodes, by new *Controllers* that have the same roles as the server's *Controller* of the centralized mode.

## 5. MVC-3DC implementation examples

We demonstrate, through two examples, the benefits of our model that shows the complementarity of MVC-3DC separation between *model*, *controller*, *library* and *interactive view* and the MVC-3DC collaboration through *"Comm"* and *"Coord"* distribution policies.

Thus, we make a short point about the current implementations of MVC-3DC by illustrating two examples. The first one explains how MVC-3DC helps the designer to model virtual reality collaborative environment using 3D interaction tools. The second one describes how MVC-3DC has been used to design an augmented reality collaborative system.

The first implementation of our model is dedicated to a virtual reality medical project. This implementation has been made with Unity 3D (Unity) as 3D rendering engine. Our platform implements *"Comm"* and *"Coord"* distribution policies in *Controllers* facets. The communication with nodes is based on TCP protocol for connection and UDP protocol for exchanging data and update messages (Ma and Gao, 2012). We also use multicast facilities to communicate different nodes.

The second implementation is dedicated to industrial collaborative visualization. It concerns e-maintenance project developed in our laboratory (Zenati and Zerhouni, 2007), (Benbelkacem et al., 2011). This implementation is, principally, based on both desktop visualization and mobile visualization. The *Controllers* facets of the distributed architecture developed use TCP for connection and RTCP for data exchange and update messages (Ma and Gao, 2012), (Koistinen, 2000).

**5.1 Sharing 3D medical operating room**

This example consists on designing distributed 3D medical operating room to be visualized by two users (see Fig. 9). One user visualizes this 3D operating room on a PC-Desktop, while the second user visualizes the same environment on a CAVE (cave automatic virtual environment). We design, also, a 3D interaction using mouse event (for PC-Desktop) and 3D ray casting metaphor or virtual free-hand metaphor (for the CAVE). The two metaphors are used for 3D collaborative navigation, selection and interaction.

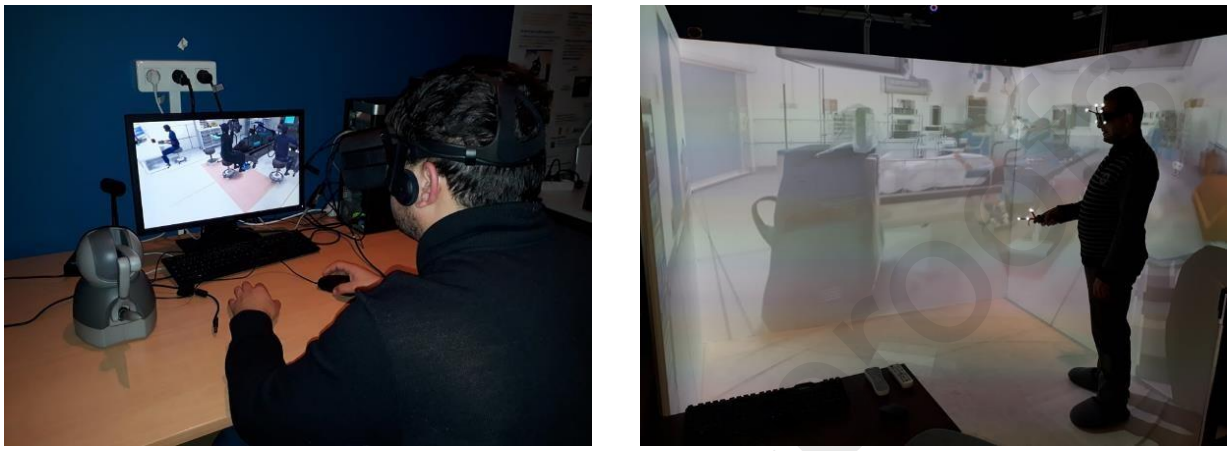

Fig. 9. Two visualizers sharing the same 3D operating room. The first visualizer in front of the PC-Desktop (in the left). The second visualizer immersed in the CAVE (in the right).

Fig. 10 shows how to use MVC-3DC based-duplicated architecture to implement shared 3D medical operating room using Unity 3D, this, by instantiating each interactive View by its proper virtual room according to user's position and device used. The second user, in the CAVE, manipulates a 3D needle using the ray casting or virtual free-hand. The first user, in front of the desktop, views the manipulation and then final position of the 3D needle. The same phenomenon is be produced at the first user which manipulates a 3D needle using a mouse event. The second user, in the CAVE, views the manipulation and then final position of the 3D needle. Form software point of view, we separate clearly the behaviour of ray casting/ virtual free-hand and mouse event from Unity 3D *interactive View* code. It's *Library* component, for each user, which supports Unity 3D mouse events and selection/manipulation using the two metaphors. We notice that these metaphors can be driven by any input device that can be a Wand a Wiimote associated with an ART tracking device.

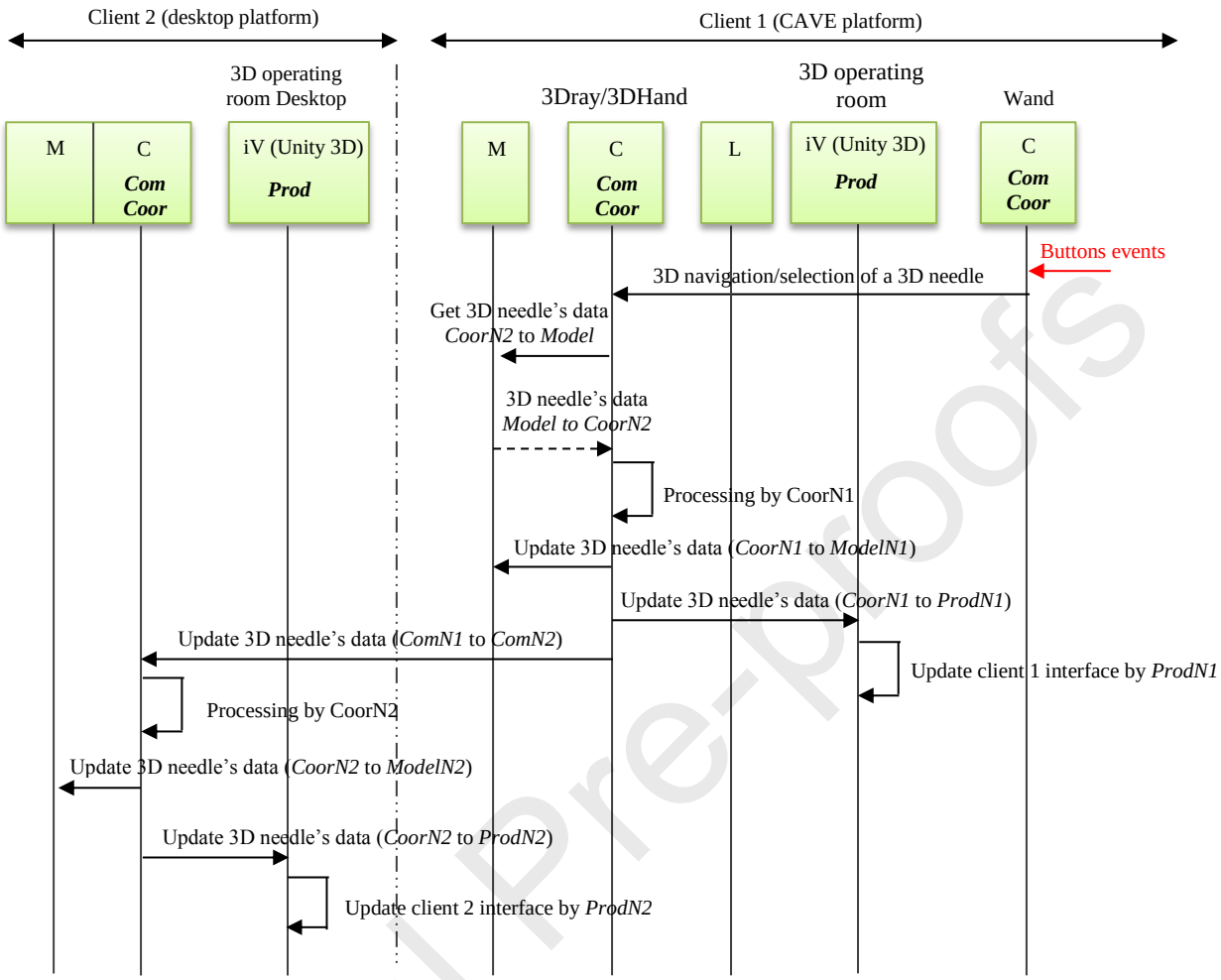


Fig. 10. MVC-3DC model to implement collaborative 3D virtual and immersive operating room.

### 5.2 AR e-maintenance help

This example implements distributing augmented reality e-maintenance assistance where a distant expert can assist a technician to perform maintenance tasks using augmented reality features. A prototype maintenance assistance is given in Fig. 11. We used MVC-3DC for collaboration designing. Fig. 11 shows a collaboration scenario. A technician asks a remote expert for maintenance help. An expert, which is available for help, accept the technician's request. A communication is established by a video streaming. The technician films the scene and transmits it to the remote expert. He, also, sends a chat message and communicates with expert by voice interaction to explain the problem occurring on the equipment. Remote expert views the equipment by video streaming in real time and inserts, through his interactive tablet, 3D objects which are sent, by Internet network, to the technician's filmed scene. These 3D objects are principally arrows, texts and tools, which encompassing one maintenance task. All the maintenance tasks are sent in the same manner from the remote expert to the technician until repairing the equipment. The technician is, therefore, guided to perform the maintenance task sent by the expert.


Fig. 11. Shared augmented reality scene sent by a remote expert visualizer (in the left) to a technician visualizer (in the right).

Fig. 12 shows the interaction between a remote expert and a technician using IRVO formalism introduced in (Chalon, 2004).

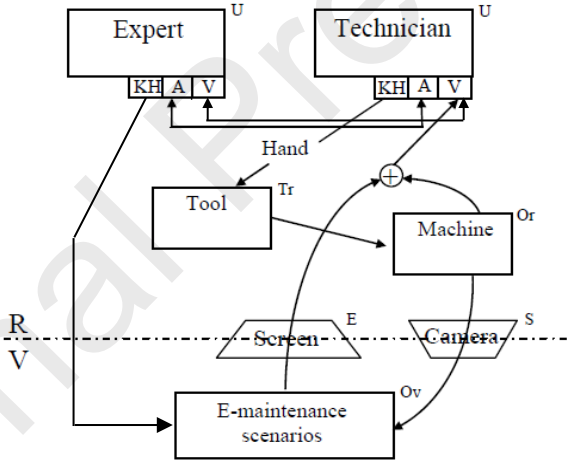


Fig. 12. Description of maintenance procedures of our application using IRVO formalism.

Using MVC-3DC, we can model this collaboration process. In fact, we instantiate *interactive View* of each visualizer to share a common augmented reality e-maintenance environment. We use *"Comm"* and *"Coord"* distribution policies to allow exchanging data and update messages from the remote expert to the technician. Fig. 13 shows this exchange: the ID of the 3D objects inserted by the expert and their position and orientation are sent from remote expert's corresponding server to technician's server and then to his *interactive View*.

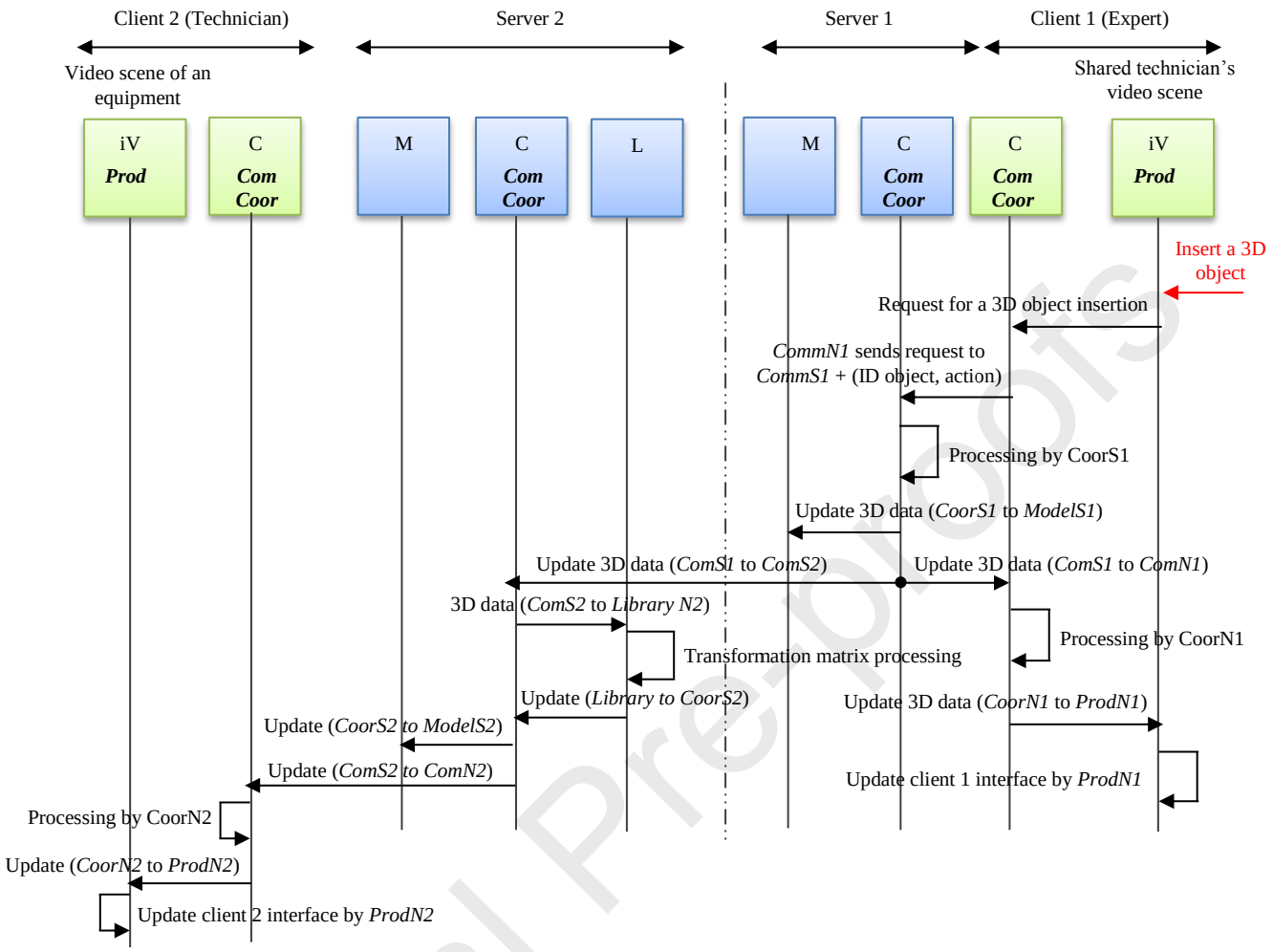


Fig. 13. MVC-3DC model to implement collaborative augmented reality e-maintenance system.

In this example we take into consideration the technician navigation within the scene. For example, the technician turns around his working environment to allow more exploration of augmented e-maintenance environment. Here, *Library* on the technician' node supports navigation processing to be available in real-time for his 3D visualizer, whatever the 3D graphics API and tracking technique used.

## 6. Evaluation of the effectiveness of MVC-3DC

The evaluation of MVC-3DC model consists to study if programmers can generate application composed of reusable, decoupled and bit complex components. Therefore, three software properties are considered in our evaluation: (1) reusability of MVC-3DC components, (2) decoupling between components and (3) complexity. In order to validate these properties, we have used the same application developed in (Benbelkacem et al., 2014) that uses three implementations of a fragment of the same application "car engine maintenance assistance", based on C# language. Here, MVC-3DC is compared to MVC, MVC optimized and AMF-C already evaluated in (Benbelkacem et al., 2014). The quality of the four models is evaluated using the three software properties given above. Each item of the software property is classified on a scale from 1 to 5 (1: disagree, 2: slightly disagree, 3: fair, 4: agree, 5: agree). The score associated with each item (see Table 1) is an average of the three opinions of programmers.

According to Table 1, the programmers estimate that structuring the code into components can be complex using standard MVC model (1/3 programmer is satisfied). The complexity is less significant with optimized MVC, AMF-C and MVC-3DC model (3/3 programmer are satisfied). Furthermore, the participants can decompose and reuse the developed components within the fragment of the application when using MVC-3DC model (3/3 programmer are satisfied). In fact, they estimate that Library (L) component is relevant to reuse or replace SDKs and tools by other ones when necessary.

Table 1. Programmers' opinions on four design patterns

| ***N°*** | ***Architecture software model*** | ***Software properties/opinions of programmers (average scores)*** | | | |
|---|---|---|---|---|---|
| | | *Decoupling between components.* | *Structuring the code into components* | *Reusability of components* | *Simplicity level of components.* |
| 1 | *Standard MVC* | *4.33* | *3.33* | *4* | *3.33* |
| 2 | *Optimized MVC* | *4.33* | *4.33* | *4* | *4.33* |
| 3 | *AMF-C* | *4.66* | *4* | *4.33* | *3.66* |
| 4 | *MVC-3DC* | *4.66* | *4* | *4.66* | *4.33* |

The tests show encouraging results on the effectiveness of MVC-3DC model. The programmers recommend more evaluations to conduct on the proposed groupware. They, also, recommend that tests should be achieved on other case studies.

**7. Conclusion**

In this paper, we proposed MVC-3DC architectural model, an explicit evolution of the MVC-3D model, dedicated to 3D collaborative augmented and virtual environments. The behaviour of each shared 3D object of the environment is distributed to four components (facets). Model (M) integrates the core data and behaviour of objects. Interactive View (iV) presents the objects to the user. Library (L) deals with dedicated augmented reality and virtual reality processing. Finally, Controller (C) takes a care of consistency maintenance between the Model and interactive View, and between all the distributed Controllers (C) of shared objects. Data exchange and distribution policies are carried out through "Comm" and "Coord" processes, while "Prod" process deals with updating Interactive Views.

The proposed model ensures a best separation between functional cores, processing (SDKs and libraries) and visualization (3D graphics) in a collaboration process. This separation between interfaces proposed in MVC-3DC model is benefit since it allows effective collaboration, this, by integrating the three collaboration paradigms of the 3C model: communication, coordination and production. The proposed model provides an explicit solution to implement classic distribution policies in collaborative augmented/virtual environments.

MVC-3DC makes it possible to design these environments with high decoupling of devices and 3D graphics API. It makes it easy to use heterogeneous devices and different 3D graphics API for different nodes involved in the same collaborative session, providing easy interoperability between these devices and 3D graphics API. It is also possible to combine other 3D engines (for example, physics engines) by adding corresponding Interactive Views to MVC-3DC objects.

Finally, MVC-3DC model can help programmers to better structure the programming process and focuses specific processing in the Library (L) component with which the implementation of methods and tools can be well organized. MVC-3DC can be also adapted for applications involving heterogeneous algorithms and

different interaction devises. Using our design approach, we can preserve the programmers' practices and reduce programming complexity.

The next steps are, first, to study the possibility to integrate into MVC-3DC objects engines such as artificial intelligence-based tools to drive virtual objects. Second, we try to consider the "dynamic" aspect of interactive View.

**References**

Bass, L., Faneuf, R., Little, R., Mayer, N., Pellegrino, B., Reed, S., Seacord, R., Sheppard, S. Szczur, M. 1992. A Metamodel for the Runtime Architecture of an Interactive System. ACM Special Interest Group Computer-Human Interface bulletin (SIGCHI), pp. 32-37.

Benbelkacem, S., Zenati-Henda, N., Zerarga, F., Bellarbi, A., Belhocine, M., Malek, S., Tadjine, M., 2011. Augmented reality platform for collaborative E-maintenance systems. In InTech Book Chapter: Augmented Reality - Some Emerging Application Areas, Dr. Andrew Yeh Ching Nee (Ed.), pp. 211-226.

Benbelkacem, S., Belhocine, M., Zenati-Henda, N., Bellarbi, A., Tadjine, M. 2014. Integrating human-computer interaction and business practices for mixed reality systems design: a case study. IET Software 8(2), pp. 86-101.

Benbelkacem, S., Aouam, D., Zenati-Henda, N., Bellarbi, A., Bouhena, A., Otmane, S., 2018. MVC-3D: Adaptive Design Pattern for Virtual and Augmented Reality Systems. In third International Conference on Advanced Aspects of Software Engineering (Icaase'18), Constantine, Algeria.

Calvary, G., Coutaz, J., Nigay, L., 1997. From Single-User Architectural Design to PAC*: a Generic Software Architecture Model for CSCW. In ACM Conference on Human Factors and Computing Systems (CHI'97), 1997, pp. 242-249.

Chalon, R. 2004. Réalité mixte et travail collaboratif : IRVO, un modèle de l'interaction homme-machine. Phd thesis, Ecole Centrale de Lyon, France 214 p.

Cheaib, N., Otmane, S., Mallem, M. 2012. Tailorable Groupware Design Based on the 3C model. In Int. J. Coop. Inf. Sys.

Coutaz, J. 1987. PAC, an object-oriented model for dialog design. In Proceedings of IFIP INTERACT'87: Human-Computer Interaction. pp 431-436.

Dewan, P. 1999. Architectures for Collaborative Applications. In Computer-Supported Cooperative Work Beaudouin-Lafon (Ed.), John Wiley & Sons. pp. 169-194.

Ellis, C. A., 1994. Conceptual model of groupware. ACM Proceedings of CSCW, NY, pp. 79–88.

Fleury, C. 2012. Modèles de conception pour la collaboration distante en environnements virtuels distribués : de l'architecture aux métaphores. Phd Thesis, INSA de Rennes, France, 153 p.

Fleury, C., Duval, T., Gouranton, V., Arnaldi, B. 2010. A New Adaptive Data Distribution Model for Consistency Maintenance in Collaborative Virtual Environments. In Proceedings of JVRC, pp. 29–36.

Hill, R. D. 1992. The Abstraction-Link-View Paradigm: Using Constraints To Connect User Interfaces to Applications. In Conference on Human Factors in Computing Systems, CHI 1992, Monterey, CA, USA.

Huang, W., Alem, L., Tecchia, F., Duh, H B-L. 2018. Augmented 3D hands: a gesture-based mixed reality system for distributed collaboration. Journal on Multimodal User Interfaces. 12, 2 pp. 77-89.

Khezami, N., Otmane, S., Mallem, M. 2005. A new formal model of collaboration by multi-agent systems. Proceedings of IEEE KIMAS: 32-37, Massachusetts, USA.

Koistinen, T. 2000. Protocol overview: RTP and RTCP, Nokia Telecommunications, 2000.

Laurillau, Y., Nigay, L. 2002. Clover architecture for groupware. Proceedings of ACM CSCW, pp. 236-245.

Laurillau, Y. 2002. Conception et réalisation logicielles pour les collecticiels centrées sur l'activité de groupe : le modèle et la plate-forme Clover. Phd Thesis, university of Grenoble 1, France.

Lee, D., Lim, M., Han, S., Lee, K. 2007. ATLAS: A Scalable Network Framework for Distributed Virtual Environments. Presence: Teleoperators and Virtual Environments, 16, 2, pp. 125–156.

Lukosch, S., Billinghurst, M., Alem, L., Kiyokawa, K. 2015. Collaboration in augmented reality. Computer Supported Cooperative Work (CSCW). 24, 6, pp. 515-525.

Nigay L. 1994. Conception et modélisation logicielles des systèmes interactifs application aux interfaces multimodales. Phd thesis, Grenoble, France.

Ouadou, K. 1994. AMF : Un modèle d'architecture multi-agents multifacettes pour Interfaces Homme-Machine et les outils associés. Phd thesis, Ecole Centrale de Lyon, Lyon, France, 210 p.

Patterson, J. 1994. A Taxonomy of Architectures for Synchronous Groupware Applications. In Workshop on Software Architectures for Cooperative Systems. ACM Conference on Computer-Supported Cooperative Work (CSCW'94), pp. 317-328.

Salber, D. 1995. De l'interaction individuelle aux systèmes multiutilisateurs. L'exemple de la Communication Homme-Homme-Médiatisée. Phd thesis university Joseph Fourier, Grenoble, France.

Šašinka, Č., Stachoň, Z., Sedlák, M., Chmelík, J., Herman, L., Kubíček, P., Šašinková, A., Doležal, M., Tejkl, H., Urbánek, T., Svatoňová, H., Ugwitz, P., Juřík, V. 2019. Collaborative Immersive Virtual Environments for Education in Geography. ISPRS Int. J. Geo-Inf. 8, 3.

Tarpin-Bernard, F. 1997. Travail coopératif synchrone assisté par ordinateur : Approche AMF-C. Phd thesis, Ecole Centrale de Lyon, Lyon, France.

Zenati, N., Zerhouni, N. 2007. Augmented reality as a design tool for maintenance applications. Int. Rev. Computer. Sof (IRECOS), pp.149-158.

Unity. https://unity3d.com/.