Synchronization of distributed application instances as a learning tracking mechanism

Macaire Ngomo[*,**], Habib Abdulrab[**]

* CM IT Conseil – Département recherche et innovation – 42 rue Milford Haven 10100 Romilly sur Seine (France)
** Institut National des Sciences Appliquées de Rouen – Laboratoire LITIS, Campus INSA de Rouen - Avenue de l'Université, 76801 Saint-Étienne-du-Rouvray Cedex (France)

{macaire.ngomo@cm-itconseil.fr, habib.abdulrab@insa-rouen.fr}

ABSTRACT : We propose to define a protocol that manages the synchronization of two instances of the same application, through a communication network. In this paper, we will present the situation using an asynchronous composition model of extended automata. This formal model will serve to define and propose a first protocol, defined according to the same formalism. Our study will be done within the framework of a general synchronizable application. In the particular context of this study, the application envisaged will correspond to the SERPOLET environment, for synchronous and / or asynchronous monitoring and support between a learner and his tutor.

KEYWORDS: Synchronization of application instances, synchronization protocol, synchronous or asynchronous monitoring and support, Synchronization of application instances, low-speed Internet connection, intelligent system, learning Technologies for Education systems and intelligent environment, Serpolet system.

1. Introduction

As part of our development projects on e-learning management systems, advanced learning technologies, intelligent environments, educational systems, we were faced with the problem of the poor quality of the links in certain geographical areas, due to low-speed Internet connectivity, not allowing us to use the standard software of videoconferencing, remote display, or remote document sharing, typically carrying very high-bandwidth bitmap images.
The application instance synchronization mechanism is used in different contexts.

Figure 1: Synchronizing files with OneDrive.

For example, the synchronization procedure for ADO.NET is to use session variables for collaborative synchronization [ADO.NET 2017]. This is the synchronization of other ADO.NET-compliant databases, showing how databases other than SQL Server can be synchronized using the Sync Framework. Implementations rely on Sync Framework types and properties. Synchronization of application instances also allows you to determine a synchronization policy for each of the sites present on the Envol WordPress application. Synchronizing files using the OneDrive Synchronization Client from OneDrive Enterprise or SharePoint Server allows you to synchronize files on a computer from local instances of SharePoint Server or OneDrive Enterprise [Microsoft OneDrive, 2017]. The files can then be used directly in the file system even in disconnected mode. As soon as the collection is restored, the changes are synchronized automatically.
To address this specific need, we propose in this paper to define a protocol that manages the synchronization of two instances or copies of the same application, through a communication network. In this paper, we present the situation using an asynchronous composition model of extended automata [Sarma, 1996] [Reffay, 2003a] [Betbeder et al., 2006]. This formal model will serve to define and propose a first protocol, defined according to the same formalism. Our study will be done within the framework of a general synchronizable application. In the particular context of this study, the application envisaged will correspond to the environment of the authoring system Serpolet (and that of its derived system Cognifer) [Glaës & all, 1999] [A6, 2008a] [A6, 2008b] [Ngomo, 2005] [Oubachi & all, 2005] [Ngomo & all, 2005], for synchronous and asynchronous monitoring and assistance between a learner and his tutor. In general, it will be a matter of being placed in a situation of assisted learning or tutored learning using a communication mechanism based on the exchange of events based on scripts between the "Master station" and the

"Slave station". Tutoring is more a function related to the accompaniment of learners during their learning pathway [Nivet P., 2010], [Monget M.-C. & al., 2008], [Omeric, 2013a], [Omeric, 2013b]. The role of tutor is reinforced in the distance education systems thanks to a set of tools resulting from the advances of the new technologies for information and communication. The general model for organizing online tutoring can be summarized in the following figure [Nivet P., 2010]:

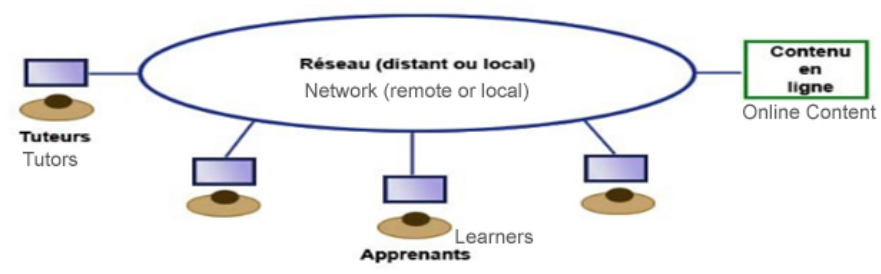


Figure 2. General Model of online tutoring to online resource.

Our model is inspired by this general model of tutoring and modifies it by removing remote access to educational resources (online content) to obtain a model of tutoring called local resource.

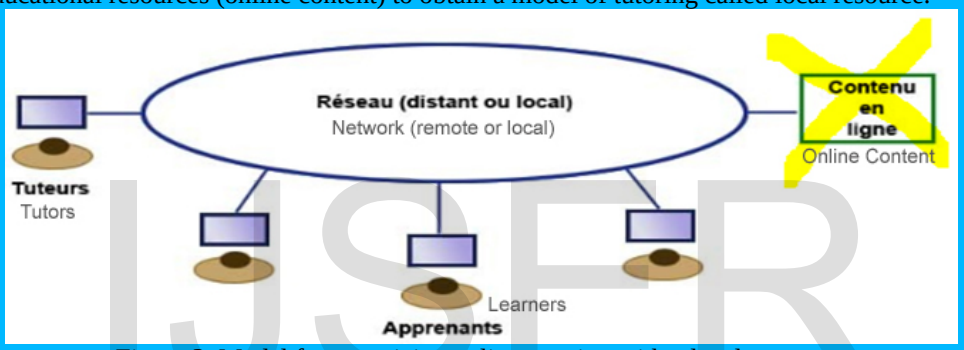


Figure 3. Model for organizing online tutoring with a local resource.

[Nivet P, 2010] makes an inventory of the tools according to the tutoring activities. The palette of the tutor is rich of several families of tools. Each family changes very little in terms of pedagogy because its objectives remain constant but it can evolve very quickly on the technical level according to the contributions of the new technologies.

2. Problematic

A first user, named master (UM), has an application whose localization it manages the evolution or the course. A second user, named slave (UE), has a copy/instance of the same application and, for example for monitoring purposes, assistance, wants to synchronize the state of his copy, and his future evolution on that of UM. We voluntarily chose to detach ourselves from the roles of teacher and pupil, as these roles do not necessarily reflect the sense of synchronization. Indeed, when monitoring the activity of a pupil, the role of the teacher is held by the teacher. In the case of an explanation given by the teacher or guardian on the environment of a student, the role UM is held by the teacher.

The local application is called X. It receives the events transmitted by the user, events to which it reacts. Periodically, it saves its current state X as a series of states XS0, XS1, XS2 ... The last saved state will serve as a starting point when requesting to resynchronize the remote copy of the application. The set of states will be used during an asynchronous exchange (deferred tracking). The following diagram (FIG. 3) describes in a synoptic manner the type of exchanges that one has to consider between the user "Master" and the user.

Each module is interconnected by queues to the modules with which it communicates. These queues make it possible to model the asynchronism of the behaviors between these entities.

The function X '= f (X) describes the sequence of states that the synchronized application saves. The numbers in the queues represent the numbers of the events that transit between the different entities. Because of the asynchronism, these numbers are not all processed at the same time, on the diagram. The sequence of numbers gives the order of processing of the messages by the entities. Any application to synchronize a copy must have three basic characteristics:

a) it must have the possibility of saving at certain moments its state, in order to be able to be restarted from this state. This property is necessary for performance reasons because, of course, it is not possible to replay the entire sequence of events since the beginning of the application. The backup can be done locally or on a server.
b) it must be able, from a saved state performed, to recover the latter state, and to resume its execution from this state. The user can specify a particular state to replay the saved sequence as a series of states.
c) finally, it must be able to receive events from the outside. These events intervene for the progress of the application. They are communicated to the copy of the application synchronized by the synchronization driver system.

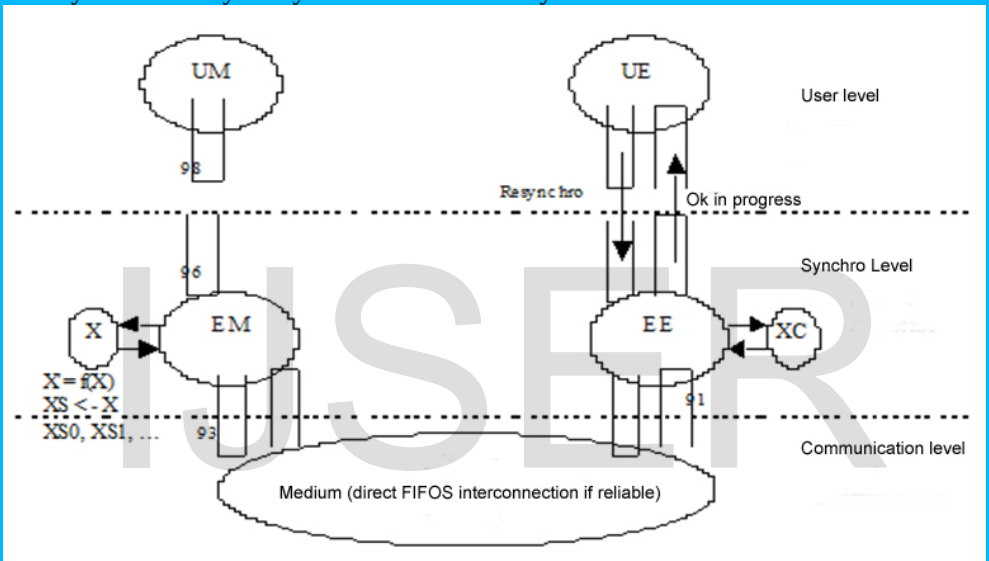


Figure 4. Diagram of master and slave communication.

Of course, the notion of the backup state of an application is dependent on the type of application considered. In the same way, events handled by an application depend on the application, both in terms of their choice and their granularity. The main difficulty for a given application will be the possibility of identifying these events and the notion of backup / recovery state. Subsequently, in our study, we assume that the generic application X possesses the three preceding characteristics, implemented elsewhere in the project. This is the case of the authoring system Serpolet.

3. Synchronization Service Architecture

3.1. Extended automaton model

The model on which the formal specification of the distributed system that we present later is based is that of extended automata. In this model, each specified entity has a behavior represented in the form of a finite state machine.
Briefly, an automaton is in the form of a quadruplet <E, T, f, q0> where
- E represents a finite series of states,
- T a finite sequence of transitions,
- f: E * T -> E, a transition function
- q0 an initial state.

An extended PLC also has:
- variables added to it,
- conditions on the firing of transitions,
- a condition on the reception of a message when shooting a transition (denoted? m),
- actions when shooting a transition
- message transmissions during the shooting of a transition (noted! m)

The interconnection of extended PLCs is done by adding FIFO queues, and the communication between entities is done through the specified queues. This addition of FIFOs is done in the general form of bidirectional channels between automata, to represent the asynchronism between the behaviors of the automata.

3.2 Architecture

The architecture of the system respects the OSI three-layer basic design model. In this model, the design of a new communication service is done by means of a software layer accessible to a set of users in the form of a set of service primitives. This software layer consists of a set of distributed entities, which interact with one another by defining a common message format, called PDUs (Protocol Data Units). These PDUs are physically exchanged between entities using a lower level communication service.
Figure 5 shows the architecture of the application copy resynchronization protocol.

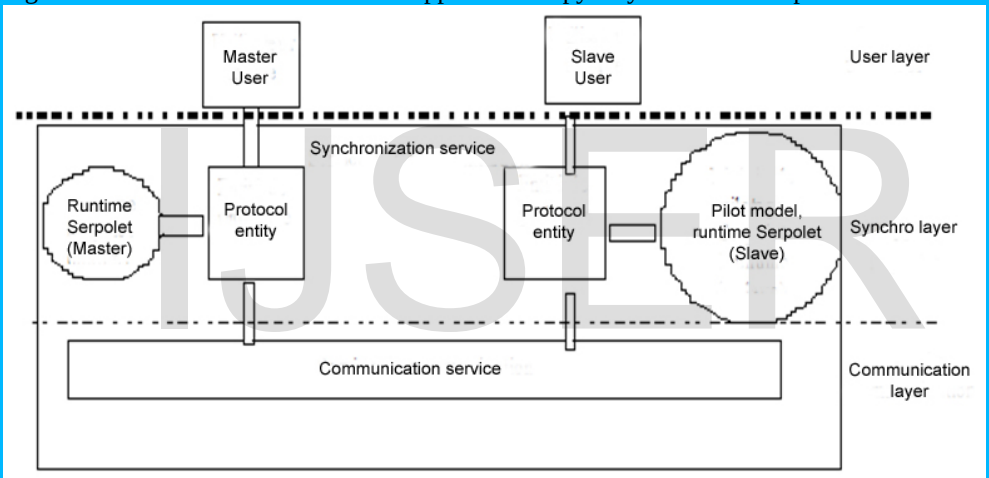


Figure 5. Copy synchronization protocol architecture.

The "master user" and "slave user" boxes respectively represent the two users of the resynchronization service.
This service is physically realized in the form of the two software entities, "protocol entity", which interact with each other using the underlying communication service.
The architecture defined uses two instances of "runtime" Serpolet, one master (linked to the application) and the other slave (linked to the control module of the application). Between these two instances, a distributed service, in the sense of communication service, is specified and developed in the form of a communication protocol to effectively ensure the actual synchronization of these two instances.
The design architecture (Figure 6) will subsequently be defined in the implementation architecture in which the entities corresponding to each user will be physically distributed on remote machines corresponding to each user.
From a conceptual point of view, the synchronization function remains decoupled from the Serpolet modules. In the final phase of implementation, integration of this entity with this module can be envisaged, but in any case it will remain independent of the other functions of the synchronized software.

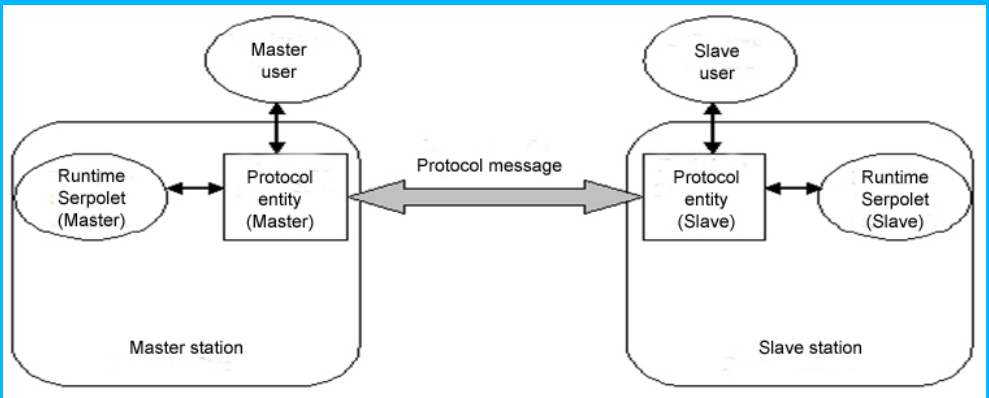


Figure 6. Implementation architecture.

3.3 Hypotheses

The rest of the report presents the description of the communication interfaces in the form of service primitives and then describes in detail the behavior of each of the entities implemented. The specification of the resynchronization service is subject to a number of assumptions on the one hand at the level of the communication service and on the other hand at the level of the service rendered. We have deemed it essential to list these assumptions which guarantee the degree of application and operation of the specified protocol.

Hypothesis 1: the behavior of the specified service is based on a reliable and FIFO communication service, such as a TCP channel. We have assumed that this communication service is part of the software layers high enough to consider this assumption as acceptable. In the case of a sudden connection failure, for example, we assume that the underlying software layers inform the distributed entities, which terminate on error, without attempting to resume.

Hypothesis 2: The context of the X application to be resynchronized is modeled as a backup / restart state Si, and a list of events to be replayed {evi, evj, evk, ...} before considering the two copies as synchronized.

A special case is an application that can save its state after any event received. In this case, the context of such an application is of the form

C'={S'i, { }}.

Hypothesis 3: when the slave user has his application copy slaved to that of the master, he can remove this slave and make his local application independent. In this case, the copy remains in the last state reached during the synchronized work session, and does not return to its state before servo-control. This choice has been made to avoid overloading the management of the contexts of the application because the return to an old state does not always seem judicious (case of a simple follow-up for example).

4. System details

4.1 Service Primitives

The service primitives are used to communicate between the 5 participants of the system.

Communication : Master user – Master entity.

- User-generated event for the X application (mouse click, keyboard key, ...).
- Informs the Master user that a person has requested synchronization on their application.
- Informs the Master user that a person has stopped synchronizing their two applications.

Communication: Slave user - Slave entity.

- The slave user makes a request to synchronize its local application with a remote application (that of the future master user).

- The slave user receives confirmation that his application is successfully slaved to the master user's application.
- The slave user requests a break in the synchronization of his application with that of the master user.

Entity communication - Communication service.

- Sending messages from an entity to the communication service.
- Sending messages from the communication service to one of the entities.

4.2 Communication messages Entity - application copy.

This event was issued by a user, and it is intended for a piloted application. This command is issued by the master entity, and is equivalent to an immobility order, and a context request.
This message responds to Save_CTXT. It is issued by the master application, and has as its final destination the Slave application. It contains all the information needed to position the slave application where the master application is.
Injection of the context emitted by the master application, in the slave application. Event returned by the slave application once the context of the master application has been replayed. This is the perfect synchronization signal between the two applications. Injection of the context emitted by the master application, in the slave application. Event returned by the slave application once the context of the master application has been replayed. This is the perfect synchronization signal between the two applications.

4.3 The P.D.U.

Synchronization request from the slave. Acknowledgment of the master entity to the synchronization request of the slave entity. Context of the master application. Acknowledgment of the slave entity to prevent the master entity that the slave application is actually synchronized. An event initiated by the master user, relayed by the master entity for the slave application. Synchronization request issued by the slave. Event issued by the master entity, which signals the end of application event sends, and the actual end of the synchronized work session.

4.4 Sequence diagram

A supervised work session is divided into 4 phases:

1. **Autonomous work.** Each user is working on their own application.
2. **Synchronization request.** The future slave user makes a synchronization request to its entity. Under Serpolet, it will not be necessary to run the application on the slave station. It is up to the control module Serpolet to pass on the actions coming from the master station to the slave station. Only the execution will be necessary, and it is this which communicates with the slave entity (transmission of data by the slave entity - reception and processing of the data by the control module).The slave entity is responsible for establishing the connection with the remote entity. The two entities exchange the necessary information. The “master” entity blocks the “master” application. Thus, no more events can be issued by the master user. This restriction makes it possible to freeze the state of the master application and to save it in what we will call a Context. This operation is specific to the application used. This phase ends with a perfect synchronization of the two applications, and the unblocking of the master application.
3. **Synchronized work :**During this phase, the slave user does not have access to his application. The control module is in permanent communication with the slave entity. The master user works normally, and all his actions are reflected on the screen of the slave unit (by managing the actions coming from the master station or by transmitting the actual image of the screen of the master station on the slave station).
4. **Sign Out :** This phase is triggered by the slave user, who decides to stop the synchronized work session. After the end of the exchanges in progress, it stops the execution of the control module.

This results in a return to phase 1.
In all cases, the master application will communicate with the master entity to transmit progression data to the master entity at defined time intervals.
The recording of the actions can be managed automatically at the level of the sequences or manually (programming by the author) for actions of finer granularity.
In the sequence diagram that follows, we will use the following conventions:

- XM Application of the slave user.
- EM Slave entity.
- XE Application of the master user.
- EE Master entity.

The arrows which leave or arrive at entities, and remain suspended, are in fact intended for their respective user. This notation has been chosen so as not to overload the scheme.

5. Automatons

5.1. SDL Syntax

The following controllers are represented with the SDL (Specification and Description Language) symbol.
The SDL language [SARM96] is based on the extended automaton interconnection model, and has a standard graphical syntax to represent the detail of the described automata. We give in this part the main elements of graphic representation, used later to describe the behavior of the entities presented.

5.2. Modeling entities

4.2.1. Master User

The master user simply generates events to his local X application. For him, synchronized work is limited to receiving a signal at the beginning and at the end of the connection.

5.2.2. Slave user

He controls his local application, or enslaves it to that of the master. In the latter case, its only possible action is a disconnection request.

5.2.3. Master entity

The main role of the "master" entity is the routing of events from the user to the application and, during the synchronized work phase, to the communication service.

5.2.4 Slave entity

Outside a synchronized work session, the role of the slave entity is to route events from the user to the application. During the synchronized work phase, it is responsible for receiving events transmitted by the "master" entity that it transmits to the Serpolet control module.

6. Synchronization Modules

This part describes the list of interfaces that each user, master and slave, possesses. The state of each interface corresponds to one of the states of the user PLCs. The change of the contents of each interface is caused by the arrival of an event, modeled by the transitions of the user PLCs.
The two synchronization modules are shown in the diagram in the figure below. Once the two modules are connected, a communication channel allows the exchange of data between the two entities. Each entity receives data from the local "application runtime" that it sends to the other remote module, and data it receives via the channel to the local "runtime".
Although it is possible to initiate a communication to several, to optimize the follow-up, in this version the communication is point to point. Indeed, even if our model is not limited and even if

there are technical possibilities, we privilege in this study the quality of the exchanges and therefore did not envisage a simultaneous follow-up of several positions.

Figure 7. Communication between the "Master" and "Slave" modules.

6.1. Master Module

This user interface has only an indication role: it indicates whether the "master" application is operating autonomously, or whether it slaves the slave. The proposed interface consists of two text fields. The first one at the top specifies the state of the application. The second, below, is a message that details the meaning of this state.

Synchronize

Application synchronizing another

Independent

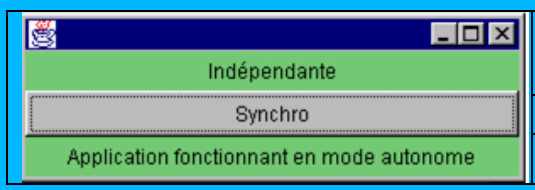
Application running in standalone mode

Figure 8. Application running in standalone mode and synchronizing another.

The transition from the "independent" state to the "enslaved" state is caused by the arrival of the primitive "Sbdy_sync_ind". The reverse path occurs when the primitive "Sbdy_end_sync_ind" arrives.

6.2. Slave Module

The interface of the slave has two roles: it allows to request that the local application is enslaved to that of the master and it informs of the synchronization state of the two applications. The requests for enslavement/servo-control and end of enslavement/servo-control are done using a button. The status of the local application is displayed using two text fields. The first gives a brief description of the state; the second explicitly states this state.
The interface of the slave only shows that both applications are independent when the user is in the "rest" state.
The interface of the slave only shows that both applications are independent when the user is in the "rest" state.



Indépendante.

Synchro

Application fonctionnant en mode autonome.

Figure 9. Application running in standalone mode and not synchronizing another.

The action of the "sync" button causes a synchronization request. This action causes the "Req_sync_req" primitive to be issued, as described by the slave user controller, which changes to the "Wait_conf" state. The interface that corresponds to this state is then as follows:

Figure 10. Application being synchronized.

Once the synchronization is effective, the slave user receives the message "Req_sync_conf" and switches to the "Sync_Work" state. The button becomes active again. It will be used to stop synchronization. The interface that corresponds to this state is then as follows:

Figure 11. Application enslaved to that of the Master.

The action of the user on the "End" button corresponds to an end of synchronization of the applications. It results in the issuing of the primitive "End_sync_req", and a return of the slave user to the "Rest" state. The new interface then corresponds to that of the independent applications.

7. Modes of communication

Three communication modes are possible to allow exchanges between the master module and the slave module: the synchronous mode and the asynchronous communication mode.

7.1. Synchronous communication mode without saving the trace

The synchronous communication mode without backup of the execution trace allows direct real-time exchanges between the master module and the slave module without saving the execution trace of the application copy on the master side. During execution of the master-side application copy, the scripts associated with the executed functions as well as the environment parameters are directly sent in real time to the slave module without saving them.

7.2. Synchronous communication mode with trace backup

The synchronous communication mode with backup of the trace allows delayed exchanges between the master module and the slave module. During execution of the application copy on the master side, the scripts associated with the executed functions as well as the environment parameters are directly sent in real time to the slave module while saving the execution trace. The slave module can then consult this execution trace and replay the recorded sequences in delayed mode.

7.3. Asynchronous communication mode

The asynchronous communication mode only allows delayed exchanges between the master module and the slave module. During execution of the master-side application copy, the scripts associated with the executed functions as well as the environment settings are sent to the server to save the application's run-time trace. The slave module then has the possibility of consulting the

execution trace in delayed mode and replaying the recorded sequences, for example for tracking purposes.

8. Structure of scripts exchanged

This part describes the structure of the scripts exchanged between the master copy and the slave copy of the application. These scripts are composed by:

- The functions performed;
- The environment parameters that enabled the execution of the functions.

For each function executed, the general structure of the sent script is as Function, function settings, environment settings, mouse events, keyboard events, or other devices.

9. Application to monitoring in remote learning situations

One of the fields of application of this mechanism is distance education. The application of this mechanism in the follow-up of learning or on-line assistance (tutoring, tele-education, etc.) is an important step forward, particularly in geographical areas where Internet connections are degraded or low-speed. Copies of the application (master and slave) must then be available on the learner (master or slave) and on the tutor (master or slave) station that provides monitoring or assistance. The "master" module will be on the learner's side when the learner unrolls his application copy which will command the tutor's copy. It will be on the tutor's side when the tutor unrolls his application copy which will command the learner's copy. Each image of the application can operate in master or slave mode, depending on the activity of the actors involved in the device.

10. Conclusion and perspectives

This article presented the formal specification and design, using the SDL language, of a service and a synchronization protocol for copies of distributed applications, and the remote control of one of them. A set of user interfaces was then deduced from the user behavior. This protocol has been tested and validated in an environment built around authoring systems Serpolet and Cognifer. We envision as a continuation of research work to extend it to other environments. This mechanism is an alternative to videoconferencing software, remote screen sharing, or remote document sharing, typically carrying bitmap images that are very bandwidth-intensive (Skype, WhatApp, CU-SeeMe from the White Pine Editor, Team Viewer, Mikogo, Join.me, AnyMeeting, Zipcast, LiveMinutes, GoToMeeting, QuickScreenShare, BigBlueButton, etc.). These software are used to communicate, give webinars, web conferencing, remote assistance or troubleshooting, remote meetings, collaborative online work, etc. which require high-quality links (broadband), and therefore difficult to use in the case of a low-speed network. Compared to these Softwares, the mechanism used here is very simple and economical. It comes down to a very light-based event exchange that does not require a lot of resources or a large bandwidth. This mechanism therefore makes it possible to develop synchronous and / or asynchronous learning tools that are very economical.
Our project on this subject aims at the generalization of this approach on other types of application other than on-line training environments. The idea is to allow any application to be provided with application instance synchronization services in point-to-point or multicast mode.

Références

[ADO.NET 2017] « Synchronisation de bases de données Scénarios de collaboration Synchronisation d'autres bases de données compatibles ADO.NET. https://msdn.microsoft.com/fr-fr/library/cc761645(v=sql.105).aspx.

[Betbeder & al., 2006] Betbeder M.-L., Reffay C. and Chanier T., 2006. Environnement audiographique synchrone : recueil et transcription pour l'analyse des interactions multimodales. In *JOCAIR 2006, Premières journées Communication et Apprentissage instrumentés en réseau*, Amiens, France, pages 406--420, July 2006.

[Glaës & all, 1999] G. Claës, M. Ngomo, M. BenTaarit : La plate-forme d'enseignement à distance COGNIFER, documentation technique, Editions MédiaGuide, Evry, France, 1999.

[Microsoft OneDrive 2017] https://onedrive.live.com/about/fr-FR/business/.

[Monget M.-C. & al., 2008] Marie-Christine Monget et Tomi Kelo, Work towards automated vendor-neutral certification of ICT skills. Dans Actes du congrès Ed-Media2008, pages 37-49, Vienne (au), Juillet 2008.

[Omeric, 2013a] Point sur la FOAD. Coopération Internationale 2012 (2013). Rapport de mission. Mission de suivi du projet FOAD-LSN. Mars 2013, France.

[Omeric, 2013b] List des termes courants du domaine de la formation et de l'enseignement. Août 2013, France.

[OUBAHSSI & al., 2005] OUBAHSSI L., Grandbastien M., Ngomo M. and Claes G., 2005. « The Activity at the Center of the Global Open and Distance Learning Process. » The 12th International Conference on Artificial Intelligence in Education, AIED 2005, Amsterdam.

[OUBAHSSI, 2005] "Conception de plates-formes logicielles pour la formation à distance, présentant des propriétés d'adaptabilité à différentes catégories d'usagers et d'interopérabilité avec d'autres environnements logiciels", œuvre [Thèse de M. L. OUBAHSSI dans le cadre d'A6/OMERIC, décembre/2005], Thèse de doctorat de l'Université René Descartes – Paris V, Centre Universitaire des Saints Pères, UFR de Mathématiques et Informatique, Paris, 2005.

[Reffay, 2003a] Reffay C. and Chanier T., 2003. Mesurer la cohésion d'un groupe d'apprentissage en formation à distance. In *Actes de la conférence Environnements Informatiques pour l'Apprentissage Humain (EIAH'2003)*, Strasbourg, France, pages 367--378, April 2003.

[Reffay, 2003b] Reffay C. and Chanier T., 2003. How social network analysis can help to measure cohesion in collaborative distance-learning. In *Procs. of Computer Supported Collaborative Learning Conference (CSCL'2003)*, Bergen, Norway, pages 343-352, June 2003. Kluwer Academic Publishers : Dordrecht(nl).

[Reffay, 2005] Reffay C., 2005. Réseaux sociaux et analyse de traces des forums d'une communauté d'apprentissage. In G.-L. Baron, E. Bruillard, and M. Sidir (Dir.), editors, *Symposium, formation et nouveaux instruments de communication*, Amiens, France, pages 13 pages, January 2005.

[Nivet P, 2010] Pierre Nivet. 2010. Un inventaire des outils du tutorat en ligne. Collectif OMERIC, Avril 2010, France.

[Sarma, 1996] A. Sarma. An introduction to SDL-92. Computer Networks and ISDN Systems 28(1996):1603-1615.