Plug and Play for Telecommunications – Architecture and Demonstration Issues[*]

Finn Arve Aagesen, Bjarne E. Helvik, Hein Meling and Ulrik Johansen
Department of Telematics
Norwegian University of Science and Technology
N-7491 Trondheim, Norway

Abstract

An architecture for plug-and-play to be applied for telecommunication systems is presented. The architecture is based on a theatre metaphor. Plays define the functionality of the system. PaP components are realised by actors playing roles defined by manuscripts. An actor's capabilities define his possibilities for playing various roles. The usability of the architecture will be validated through specification, implementation and testing of a demonstrator. The PaP system implementation design and the functionality of a tele-school based demonstrator is presented.

Keywords: Plug-and-play, Architecture, Telecommunication, Teleservices, Smart Networks, Intelligent Networks, Active Networks.

1 Introduction

Grade of network intelligence is *the efficient flexibility in the execution of teleservices and the efficient flexibility in the introduction of new teleservices*. Intelligent Networks [ITU92], Telecommunication Information Networking Architecture [TINA95], Mobile Agents and Active Networks ([Bies97], [Bies98], [Raza99], [Tenn97]) are all solutions aimed to improve the network intelligence.

Plug-and-play (PaP) for telecommunications means that the hardware and software "parts", as well as complete network elements, that constitute a communication system, have the ability to configure themselves when installed into a network (to plug) and then to provide services (to play) according to their own capabilities, the service repertoire and the operating policies of the system. Plug-and-play functionality means utterly increase of network intelligence.

A Plug and play system, as discussed here, shall be: *i)* Flexible and adaptable, *ii)* Robust and survivable, and *iii)* QoS aware and have resource control. The goal of the plug-and-play technology is to significantly simplify and speed up the tasks of deployment, installation, operation, management, maintenance and evolution. However, system structure and functionality to ensure a

[*]This work is done within the project *Plug-and-play for Network and Teleservice Components* supported by The Norwegian Research Council.

dependable and traffic handling capable solution are also important system properties. In [Aage99], needed properties to realise the qualities *i)-iii)* are specified.

The concept PaP stems from the personal computing area. PaP simply means that you plug-in and then the system works. In these systems, the plugged in component as well as the framework has *predefined* functionality. We denote this *static* PaP. A more general kind of PaP is when the plugged-in unit has a set of basic capabilities, but its functionality is defined as a part of the plug-in procedure and it can be changed dynamically. We denote this as *dynamic* PaP. An example is a cellular phone which obtains the services it provides depending on its inherent capabilities, which user that logs on, and which network it is attached to.

With dynamic PaP, the definition of individual components and possibly, the overall structure of components can be changed on-line. One aspect of dynamic PaP is to change the services that a component provides. Another is to propagate the ability to use the service to all the service users. *The focus of this paper is on dynamic PaP.* From now on the concept Plug-and-play means *dynamic* Plug-and-play.

Implementation of the PaP system concepts described in this paper is based on *mobile code*. Both Mobile Agents and Active Networks are based on mobile code. Solutions to PaP within telecommunication networks has also been proposed based on Mobile Agents ([Bies97], [Raza99]). However, the PaP functionality proposed are less comprehensive compared to the PaP functionality proposed in this paper. Note also that PaP, as defined here, has a wider scope of the network flexibility and adaptability than the above referred mobile agent and active network approaches.

A PaP demonstrator based on a tele-school application is being specified and implemented. The main purpose is to demonstrate the usability of the dynamic PaP architecture presented in Section 2. Section 3 discusses implementation issues. The tele-school demonstrator is presented in Section 4. Section 5 gives conclusions.

2 A PaP Reference Architecture

2.1 PaP components

The entities in the system subject to PaP are the *PaP components*, which are real-world "concrete" reactive hardware and/or software modules. These can be *combined hardware/software* modules with one or more external hardware interfaces, or *pure software modules*. These must interface with a software platform capable of running PaP application software.

Pure hardware modules are not feasible in the context of dynamic PaP. PaP components will coexist with components that do not have the PaP functionality. These are denoted as non-PaP components. A PaP component can *contain* other PaP components. All PaP components have a relationship to *PaP support*. The logical relationships between the PaP components is partly related to how PaP is solved and partly to the specific functionality of the system.

2.2 The functional object model

PaP components are composed from (one or more) interacting instances of PaP functional objects, where each instance is defined by reference to an object type. This means that the PaP *component* functionality is defined by a *functional object model* consisting of *functional PaP objects*.

ISO's reference model for Open Distributed Processing (ODP) [Duts96] defines the enterprise, computational, information, engineering and technical viewpoints. The computational and the engineering viewpoints are of *primary interest* with respect to PaP. The PaP components are basically engineering viewpoint objects. The PaP components have a computational viewpoint specification by the PaP functional objects, which are basically computational viewpoint objects. The computational model will also model the information which is subject to dynamic changes caused by the behaviour. Information models are supplementary models supporting the behaviour models.

Most object-oriented systems supports dynamic creation and removal of individual object instances. While this may be sufficient for static PaP, dynamic PaP requires in addition that:

- it is possible to change the definition of object instances and object instance structures,
- to propagate the effect of such changes to involved object instances.

Thus, dynamic PaP require a PaP support system with the ability to manipulate type definitions, and to dynamically change object behaviours and object structures according to the changes of the corresponding types. This situation has many similarities with the theatre, which is chosen as a model to describe the support functionality of the PaP system. The basic structure of the PaP system is illustrated in Figure 1. This model also acts as a "bridge" between the PaP component specification and the functional PaP object specification. PaP components are realised by actors and they are the entities realising the PaP functionality objects.

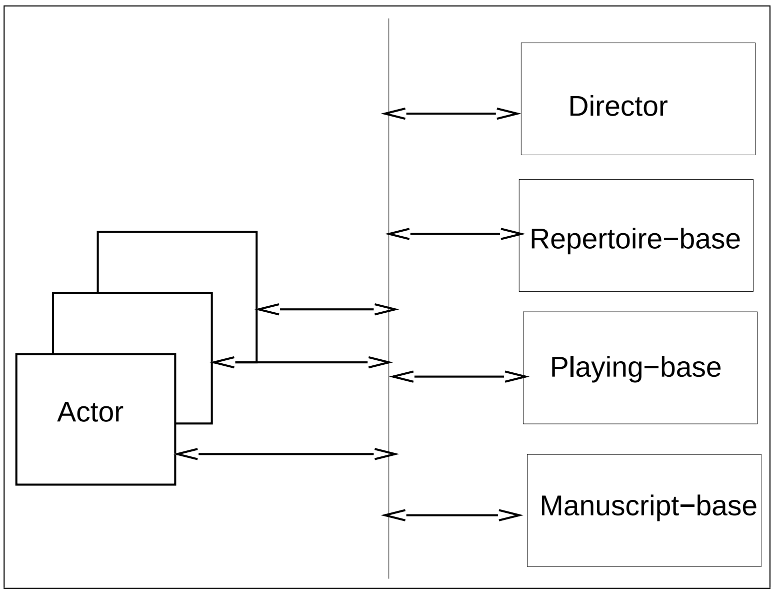

Figure 1: PaP system - Basic structure.

The model has many actor instances, one instance of a PaP-director, one instance of a repertoire-base, one instance of a manuscript-base and one instance of a playing-base. For simplicity, the system is in the present version modelled as a centralised system. An E-R model comprising important PaP concepts is illustrated in Figure 2.

An *actor* is a generic object with a generic behaviour, able to behave according to a *manuscript*. An actor also has a defined set of *capabilities*, which is the ability or power to do something. In the

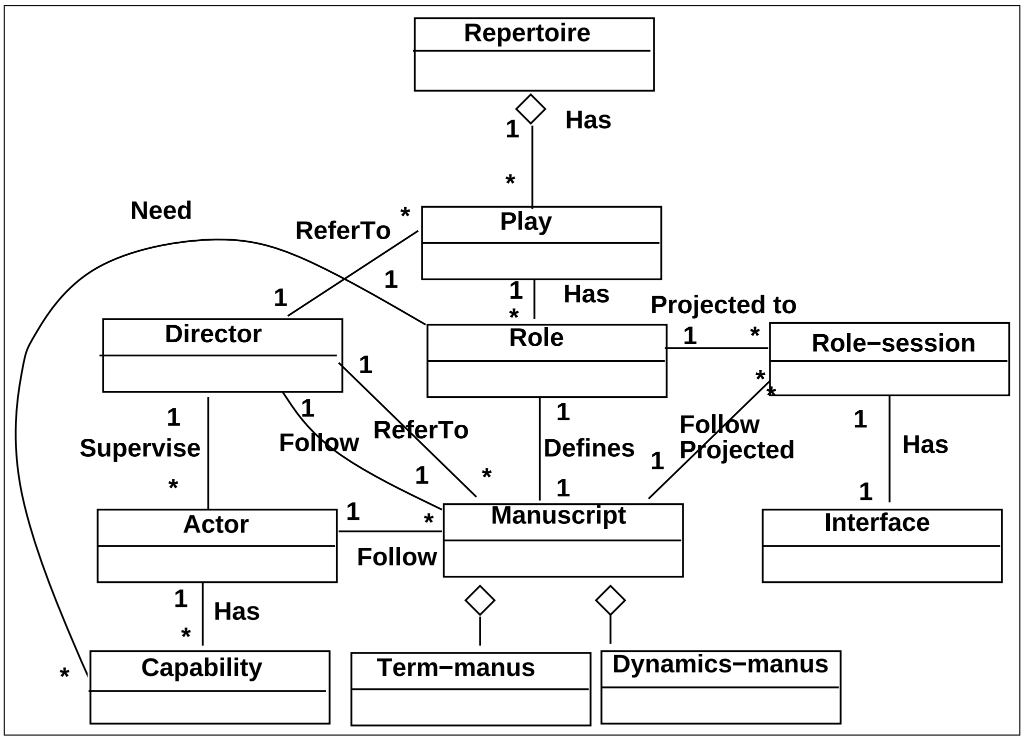

Figure 2: PaP concepts.

following a short explanation of the concepts in Figure 2 is given. For more details it is referred to [Aage99].

A *play* is a defined autonomous functionality. The play defines the context for relationships between PaP objects and their behaviour. One important PaP object functionality necessary to initialise any play is the *director*. A director behaviour is also defined by an instance of a play.

The *manuscript-base* has the manuscripts used by the actors to play their roles. The *playing-base* keeps a structural model of the instances of PaP objects that is actually playing. The *repertoire-base* keeps an overview of the potential plays and roles. Actors get an instance of a manuscript from the manuscript-base via the PaP-director. A *Role-session* is a projection of the behaviour of the actor with respect to one of its interacting actors.

An actor is able to play various roles. The *role* is defined by a manuscript which defines the total behaviour of an actor. Different from the theatre, and caused by the nature of telecommunication service providing systems, an actor can have its behaviour related to various plays at a time. However, an actor performs only one manuscript at a time. A PaP component, however, can handle various manuscripts by using various actors playing different manuscripts.

The behaviour of the role as well as the role-sessions are described as EFSMs. The whole role is composed by the logical adding of the RoleSession and an additional EFSM denoted as the *RoleSessionCombiner*. A RoleSessionCombiner definition specifies how all RoleSessions for a manuscript shall be coordinated.

2.3 PaP support functionality

The following functions are needed: Play plug-in, Play changes plug-in, Dynamic detection of needs for actors/plays/roles, Actor plug-in, Actor behaviour plug-in, Actor play, Actor change behaviour, Actor behaviour plug-out, Actor plug-out and Play plug-out. Further details are given in [Aage99].

The functions: *actor behaviour plug-in, actor play* and *actor behaviour plug-out* comprise the initialisation of a generic actor pending for a manuscript, performing the manuscript, and finally making the actor pending for a new manuscript. This functionality is denoted as the *basic PaP functionality*. The actor is initialised by first activating its PaP-director. An actor negotiates with the PaP-director in order to obtain its behaviour. The PaP-director will create an instance of a manuscript with all necessary parameters bound particularly for the actor. The PaP-director also acts as a binding object which helps to establish communication or interactions among actors. After receiving a manuscript from the PaP-director, an actor will start acting according to the specification described in the manuscript. From this point on in time the actor becomes autonomous and independent of the PaP-director until it terminates or want to change its behaviour.

3 PaP System - Implementation Design

We differ between two types of actors; *ApplicationActor* that implement some non PaP specific functionality by making use of the PaP support functionality, and *DirectorActor* that shall implement the PaP support functionality.

To describe the actor concept as an executing entity, some design related concepts are needed. The most obvious hardware and software specific concepts involved are *node* and *process/thread*. A node maps directly to a computer, which again will map one-to-one to the PaP specific design concept *PaP Support Execution Context (PSE)*. A process/thread will map one-to-one to an operating system process or thread, which again maps to the PaP specific concept *PaP Support System (PSS)/PaP Support Management (PSM)*. In Figure 3 these design related concepts are illustrated.

Figure 4 shows the layering of the functionality (i.e. the PaP layered model). The PaP relevant part consists of five separate layers, each having its specific functionality. PSE also define the PaP functionality that is termed the *PaP Static Basic Support* in the model. Static in this sense means that changes/extensions of the PSE functionality must be backward compatible with earlier versions because this functionality represents the “bootstrap” that is necessary to be able to run PaP applications at nodes. This functionality must be manually installed at the node before PaP applications can be installed and activated.

The totality of use and implementation of the PaP support functionality is distributed between, and performed by the modules defined in the PaP layered model. The execution of one function may involve one or more of the modules in the layered model.

All layers in the model, except for ”PaP specific applications” and ”Non PaP applications” are completely independent of the applications themselves. The PaP functionality interfaces to the non PaP world through the infrastructure technology at the bottom layer, and to any type of non PaP applications at the top layer.

A *PaP communication infrastructure (PCI)* architecture based on ”standardised” technological solutions will usually consist of three layers with the operating system functionality (e.g. Unix

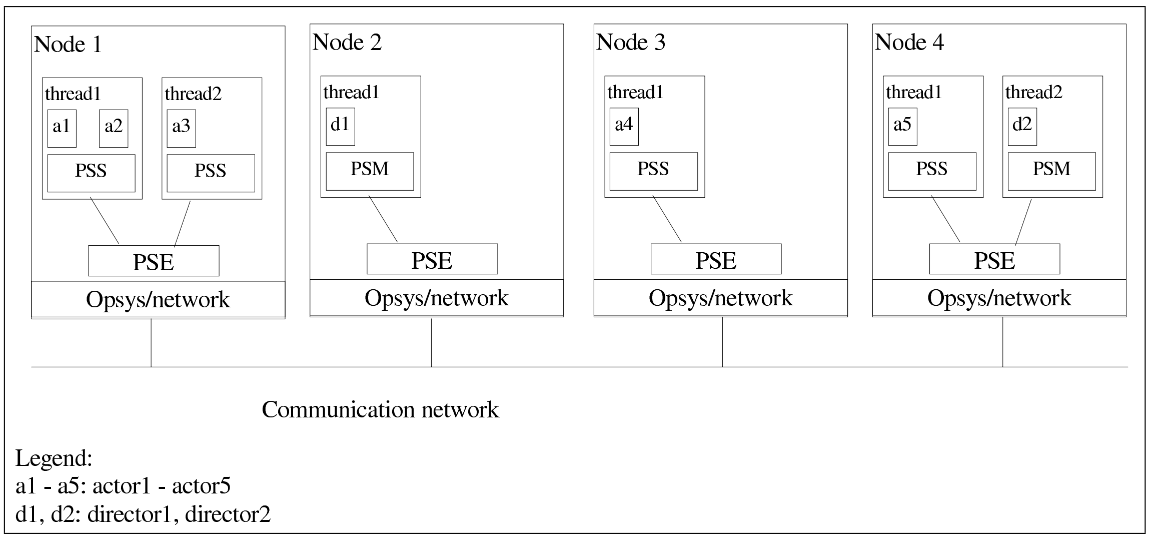

Figure 3: Engineering view of PaP concept representations.

or Windows NT) at the bottom, the network communication functionality (e.g. TCP/IP) in the middle, and some distributed system solution (e.g. CORBA ORB or Java RMI) at the top. The top layer may be omitted, but that will require a more complex implementation of the interfacing module PSE if the PaP functionality require a distributed system solution.

PaP Support Execution Context (PSE) makes it possible to run PaP software on a node, and also PaP functionality (i.e. represented by actors) on different nodes to interact with each other. PSE is able to receive requests from other PSEs, interpret these requests and take proper actions. PSE will also do start-up and initialisation of PSS/PSMs or PCIs if that is required.

PaP Support System/PaP Support Management (PSS/PSM) makes it possible to create actors within the context of an operating system process/thread, to give these actors behaviour, and to communicate information between these actors and their environments. There will be one PSS/PSM instance within each process/thread intended for PaP functionality (i.e. actors). PSM shall support DirectorActors, while PSS shall support ApplicationActors.

PaP Director is both responsible for the management of the PaP application definitions (i.e. the *Repertoire-* and *Manuscript-bases*), and for the management of information concerning Actors (i.e. the *Playing-base*). The PaP Director is involved in the execution of almost all types of PaP support functionality as specified in Section 2.3. A PaP Director instance becomes a DirectorActor.

PaP Extended Management (PXM) is additional PaP services not required for the PaP support functionality, but rather PaP extensions related to PaP operational quality. These services include functionality related to a ”robust and survivable” PaP system, and a PaP System to be ”QoS aware and to provide resource control”.

PaP Extended Support (PXS) is required for the utilisation of PaP Extended Management (PXM) from actors.

PaP applications is the collection of ApplicationActors. Actor instances are created using the ”ActorPlugIn” function, they get their behaviour using ”ActorBehaviourPlugIn” and ”ActorChangeBehaviour”, they start execution using ”ActorPlay”, and they terminates when using ”ActorPlugOut”.

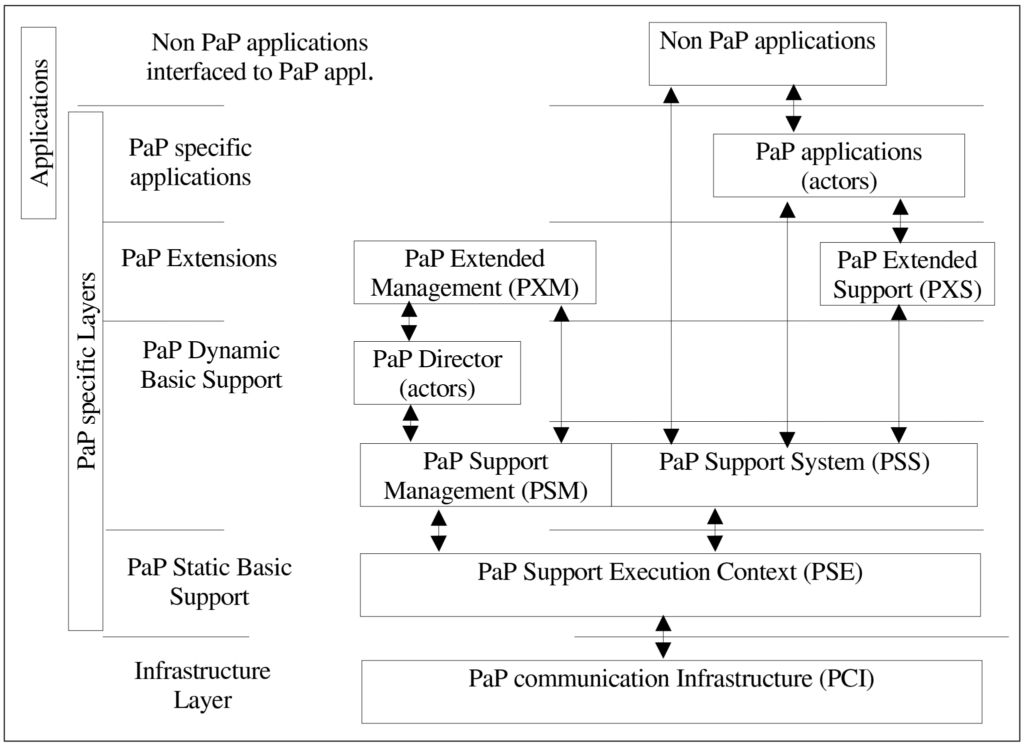
Applications
Non PaP applications interfaced to PaP appl.
Non PaP applications
PaP specific Layers
PaP specific applications
PaP applications (actors)
PaP Extensions
PaP Extended Management (PXM)
PaP Extended Support (PXS)
PaP Dynamic Basic Support
PaP Director (actors)
PaP Support Management (PSM)
PaP Support System (PSS)
PaP Static Basic Support
PaP Support Execution Context (PSE)
Infrastructure Layer
PaP communication Infrastructure (PCI)

Figure 4: PaP layered model.

Non PaP applications is allowed to interact with actors outside the control of PSS. Such interactions can be done without the intervention of any parts of the PaP system. However, such interactions must not result into control actions that are in conflict with the responsibility of the PaP System. Non PaP software is also allowed to use the PaP functionality supported by PSS/PSM. This possibility is actually necessary to be able to install and start the first operational PaP system. In this case the Non PaP application may interface to the same interface as used by the PaP specific applications, however the Non PaP application will and must perform within a separate process/thread and must be considered as one specific actor as seen from the PSS point of view.

4 The Tele-School Application

A tele-school application is selected as the application to be used in a PaP demonstrator. The main concepts are the *school*, the *school application manager*, *students*, *teachers*, *courses* and *lectures*. Basic services for communication of information between teachers and students, where teacher and students physically may be located in different geographical areas, may be either of interactive dialogue (*chat* type service), addressed non-interactive (*mail* type service), and non-addressed non-interactive (*news* type service).

Multimedia *audio*, *video* and *text* are used for communication of information. Multimedia PC type equipment is used for interaction between teachers/students and the application system.

The tele-school services provided are *real-time lecture*, *lecture on demand* and *students off-line support*. The first is the real-time lecture performance by a teacher to attached students. Students may ask questions and get answers from the teacher. Lecture on demand offers the possibility for students to go through already performed lectures that has been ”electronically recorded”. Students may freely select when to go through the lecture and they also may ask questions to the teacher, which will answer question when available by using the Student off-line support functionality. Figure 5 shows an example structure for the real-time lecture function.

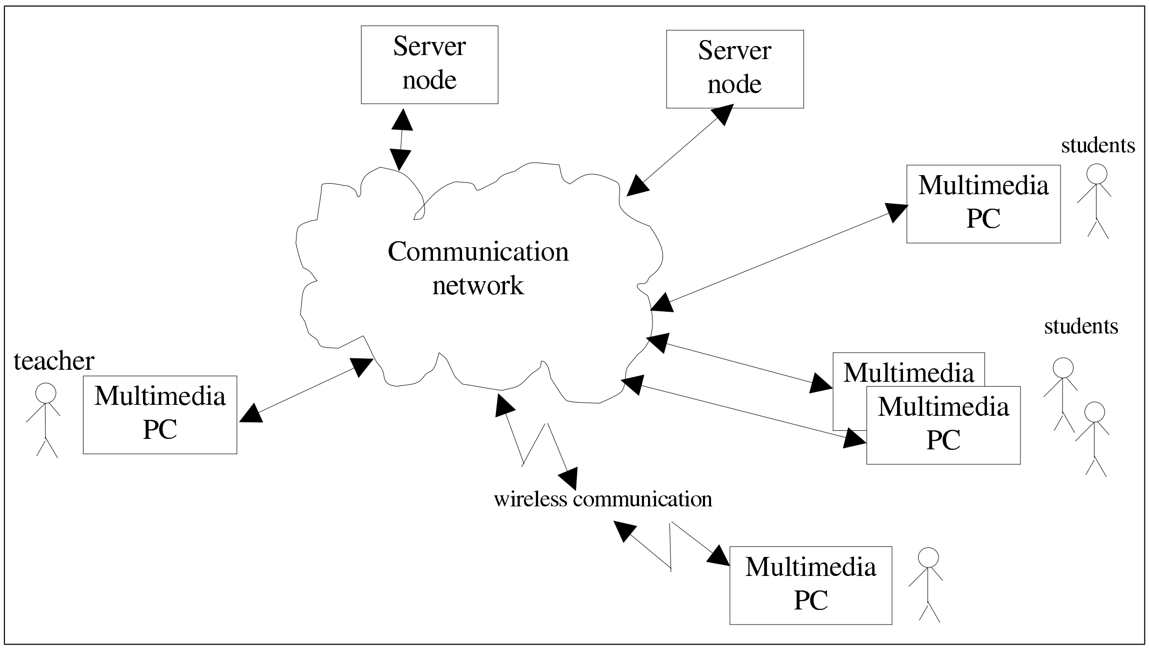

Figure 5: Example real-time lecture architecture.

4.1 An example scenario

First regard an example MSC describing a small part of the tele-school application, as is shown in Figure 6. The ”SchoolUserInterf”, ”SchoolClient” and ”SchoolServer” has been identified as three roles to be played, and the functionality shown in the figure is related to user system log-on, access verification and the user selection of what type of work to do. This is a part of the real-time lecture functionality. A set of information elements (e.g. ”WindowNew”, ”WindowClose”) to be used for actor parties interactions has been identified. RoleSessions are specified as well defined sequences of interactions between two actor parties and are indicated by vertical extra lines in the figure. The first identified RoleSession used by roles ”SchoolUserInterf” and ”SchoolClient” contains the information elements ”WindowNew”, ”LogonEvent” and ”WindowClose”.

The ”SchoolClient” role has two RoleSessions active at the same time – one with the ”SchoolUserInterf” and one with the ”SchoolServer”. The coordination between these two RoleSessions is what shall be the task of the RoleSessionCombiner.

4.2 Integration with PaP functionality

Figure 6 shows application specific aspects of the tele-school, in addition to the identification of roles and role sessions examples.

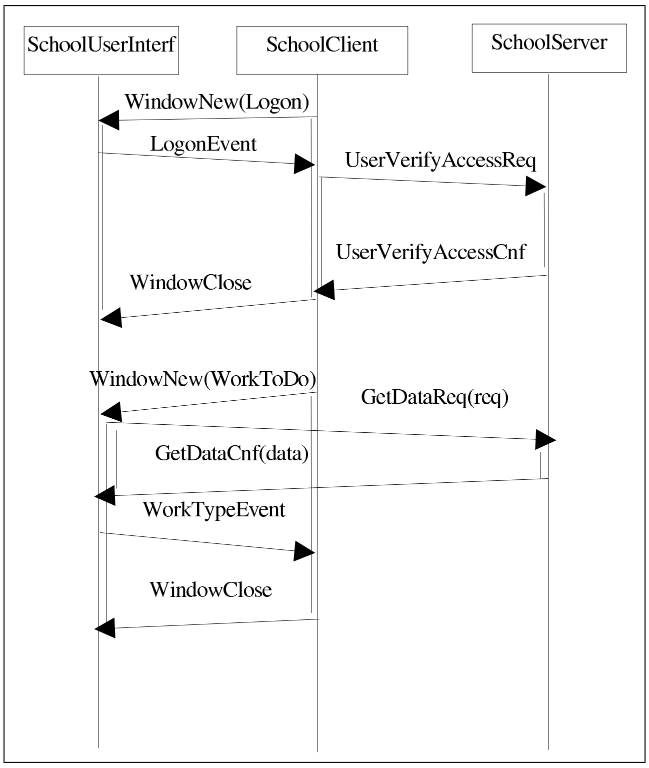

Figure 6: Example interactions between parties in the role description.

So how does the PaP specific concepts and functionality apply together with the tele-school application? The PaP functionality is generic, but it is important to know how it is integrated into the application because the application designers must consider both the application and the PaP functionality. Figure 7 extends Figure 6 to show the mapping from roles to actors, the use of PaP functionality in application execution, and to show how an application become available for use.

The figure shows that each of the three roles defined in Figure 6 are played by separate actors, named ”a1”, ”a2” and ”a3” in the figure. In addition there exists an actor ”ctx”, which has the responsibility of tele-school application installation and start-up, and actor ”d1”, which serves as the ”PaP Director” for all actors.

The application has been distributed on two nodes, namely a *Client Node*, which runs the client specific application part, and a *Server Node*, running the parts of the application common to multiple tele-school clients. The PaP Director actor has been selected to perform on the server node. The MSC example shows that the tele-school application has not already been ”installed” and is therefore installed by the ”ctx” actor (using `PlayPlugIn(play=TeleSchool)`) prior to starting a client part (using `ActorPlugIn(role- =SchoolClient)`) of the application.

`PlayPlugIn` is the PaP function used to install play definitions into a repository managed by the PaP director. This must be done before actors can apply plays. Play definitions in a repository may be changed by `PlayChangesPlugIn`, and removed by using `PlayPlugOut`.

Role sessions are created by `ActorPlugIn` and terminated by `ActorPlugOut`. In addition `ActorPlugIn` will imply the creation of an actor instance if no existing actor is able to satisfy the requirements to cooperating actor considering location, role and capabilities specified by the

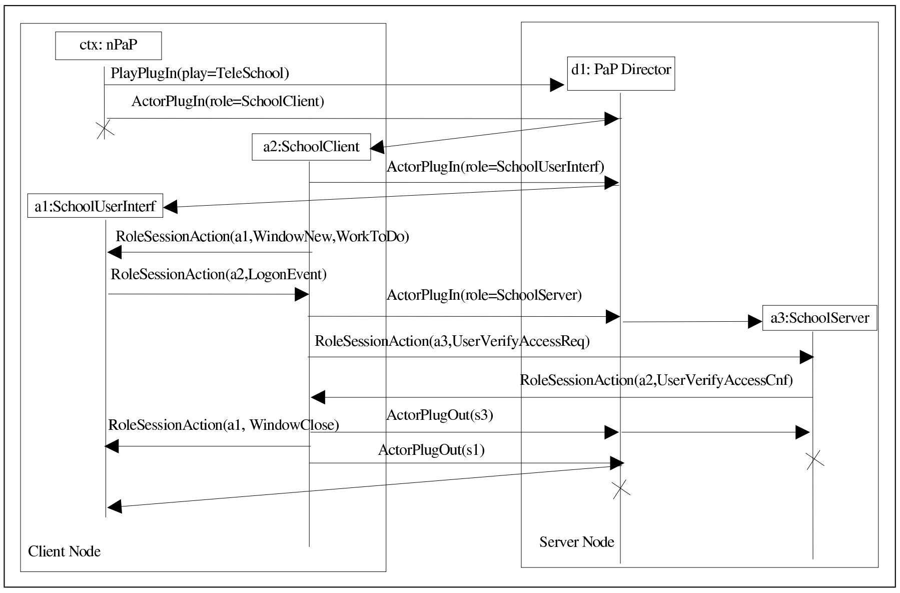
ctx: nPaP
PlayPlugIn(play=TeleSchool)
d1: PaP Director
ActorPlugIn(role=SchoolClient)
a2:SchoolClient
ActorPlugIn(role=SchoolUserInterf)
a1:SchoolUserInterf
RoleSessionAction(a1,WindowNew,WorkToDo)
RoleSessionAction(a2,LogonEvent)
ActorPlugIn(role=SchoolServer)
a3:SchoolServer
RoleSessionAction(a3,UserVerifyAccessReq)
RoleSessionAction(a2,UserVerifyAccessCnf)
ActorPlugOut(s3)
RoleSessionAction(a1, WindowClose)
ActorPlugOut(s1)
Server Node
Client Node

Figure 7: Example interaction between actors in an application.

`ActorPlugIn` parameters. `ActorPlugOut` will destroy an existing role session between two actor parties, and may imply the termination of an actor if that has been specified as a parameter to `ActorPlugOut` or if the actor's termination condition specifies so.

The figure indicates that the Director is involved in the execution of many of the PaP specific functions (e.g. `PlayPlugIn`, `ActorPlugIn`, `ActorPlugOut`). This is necessary, because Director serves, not only as a repository for component specifications, but also as a repository for existing actor instances. This means that, as seen from actors, the PaP Director become a central server for the actor. The PaP specific function `RoleSessionAction`, which is used for communication of information on established role sessions, do not involve the PaP Director because its functionality has no influence on the PaP Director work.

Two levels of plug-in/plug-out is illustrated in the figure. First, the plug-in of the specification of the PaP application functionality (i.e. the component installation) done by the `PlayPlu-gIn`, then the plug-in/out of the applications themselves (i.e. the component execution) done by `ActorPlugIn`/`ActorPlugOut`. Note that `ActorPlugOut` not necessarily means that the plugged out actor terminates. It is the interaction sequence (i.e. the role session) between two actor parties that is terminated.

It has not been shown in the figure that it is possible to dynamically change both component definitions (by using `PlayChangesPlugIn`) and actors' behaviour (by using `ActorChange-Behaviour`) for an operational system. Procedures have been defined for taking care of both these situations that may occur as results of actor requests.

5 Conclusions

An architecture concept as well as an implementation design of this architecture concept for dynamic plug-and-play has been presented. The vision is a concept to be used to simplify and speed up the tasks of deployment, installation, operation, management, maintenance and evolution of various types of telecommunication equipment and services.

Plug-and-play components has been defined as real-world concrete reactive hardware and software modules. The PaP *component* functionality is defined by a *PaP functional object model*, consisting of *functional PaP objects*. A functional PaP object is an instance of an PaP object type. Dynamic PaP requires that it is possible to change the behaviour of an object and to propagate the effect of such changes. A functionality analogous to the theatre is chosen for the realisation of PaP. The most central issues are actors, roles, plays, manuscripts and capabilities. An actors capability defines his possibilities for playing various roles according to manuscripts. The model presented is a step towards a complete architecture specification.

The specification of a tele-school application to be used for demonstration and validation of the various elements of the plug-and-play architecture has also been presented. The application involves several types of services such as real-time lecture, lecture on demand and students off-line support, suitable for the validation process. An example scenario is given. The application is now under implementation.

References

[Aage99] Finn Arve Aagesen, Bjarne E. Helvik, Vilas Wuwongse, Hein Meling, Rolv Bræk and Ulrik Johansen, Towards A Plug and Play Architecture for Telecommunications, Proceedings of IFIP SMARTNET'99, Bangkok, November 1999.

[Bies97] Andrzej Bieszczad and Bernard Pagurek, Towards Plug- and Play Networks with Mobile Code, Proceedings of ICCC'97, November 1997, http://www.sce.carleton.ca/netmanage/publications.html.

[Bies98] Andrzej Bieszczad and Bernard Pagurek and Tony White, Mobile Agents for Network Management, IEEE Commucations Surveys, volume 1 number 1, 1998, http://www.sce.carleton.ca/netmanage/publications.html.

[Duts96] Joubine Dutszadeh and Elie Najm, Formal Support for ODP and Teleservices, Proceedings of the IFIP/ICCC conference on Information Network and Data Communication, June 1996.

[ITU92] ITU-T, Principles of intelligent network architecture, October 1992.

[Raza99] S. K. Raza and Andrzej Bieszczad, Network Configuration with Plug and Play Components, The Sixth IFIP/IEEE International Symposium on Integrated Network Management (to be presented) in 1999, http://www.sce.carleton.ca/netmanage/publications.html.

[Tenn97] David L. Tennenhouse, Jonathan M. Smith, David Sincoskie, David J. Wetherall and Gary J. Minden, A Survey of Active Network Research, IEEE Communications Magazine, Volume 35 no 1, 1997, pages 80-86.

[TINA95] TINA Consortium, TINA-C Deliverable: Overall Concepts and Principles of TINA V1.0, February 1995.