TOWARDS A PLUG AND PLAY ARCHITECTURE FOR TELECOMMUNICATIONS

Finn Arve Aagesen, Bjarne E. Helvik, Vilas Wuwongse, Hein Meling, Rolv Bræk and Ulrik Johansen

Department of Telematics, The Norwegian University of Science and Technology

N-7491 Trondheim, Norway

finn.arve.aagesen@item.ntnu.no

Abstract This paper presents an architecture specified within the project "Plug-and-play for Network and Teleservice Components" supported by The Norwegian Research Council. The hardware and software parts, as well as complete network elements that constitute a communication system, shall have the ability to configure themselves when installed into a network and then to provide services according to their own capabilities, the service repertoire and the operating policies of the system.

Plug-and-play components and functional objects are defined. A theatre analogy is chosen for the specification of the needed PaP support functionality. 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.

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

1 INTRODUCTION

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.

The concept PaP stems from the personal computing area. The objective of PaP in these systems is the handling of the plug-in of new devices and software into the desktop system. 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 kind of PaP as *static* PaP. As an example, static PaP in telecommunications occurs when a mobile plugs into the network when switched on. The system provides static PaP with respect to the telephone service.

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 kind of PaP 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. We foresee that dynamic PaP is needed to cope with the highly dynamic, heterogeneous and rapidly evolving networks and service provision of the future, and at the same time keep the networks manageable. The extra complexity introduced by PaP will make development more costly. However, it is expected that this cost is small compared to the benefits related to deployment, installation, operation, management, maintenance and evolution.

Section 3 defines the properties of a PaP system. Section 4 defines a PaP reference architecture which makes dynamic PaP possible. Section 5 discusses component categories and capabilities. Section 6 gives conclusions.

2 RELATED WORKS

The "grade of network intelligence" is here defined as *the efficient flexibility in the execution of teleservices and the efficient flexibility in the introduction of new teleservices.* The semantics of network intelligence is obviously related to time period in question. Intelligent Networks (IN) ([ITU92], [ITU93], [ITU97]), Telecommunication Information Networking Architecture (TINA) ([TINA94], [TINA95a], [TINA95b], [TINA95c], [TINA97a]) , Mobile Agents and Active Networks ([Bies97], [Bies98a], [Bies98b], [Raza99], [Tenn96a], [Tenn96b], [Tenn97]) are all solutions aimed to improve the network intelligence.

The PaP system described in this paper is based on *mobile code.* Both Mobile Agents and Active Networks are based on mobile code.

The mobile code can be based on *Code On Demand* (i.e. get the code when you need it), *Remote Evaluation* (i.e. send pieces of code to the destination, where the code can cooperate with other pieces of code), and *Mobile Agents* (i.e. the code has intelligence to take decisions of where to go to get executed). Our model will mainly be based on Code on Demand. 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 any of the above refered mobile agent and active network approaches.

3 PROPERTIES OF THE PAP SYSTEM

3.1 PaP components

The entities in the system subject to PaP are the PaP components. A *PaP component* is a real-world "concrete" reactive hardware or software module. PaP components can be:

- *Combined hardware/software* modules with one or more external hardware interfaces and a software platform capable of running PaP application software. Typical hardware modules are user nodes (i.e. user equipment with network connection), network nodes (i.e. circuit switches, packet switches, routers, cross-connects) and service nodes (i.e. service providing servers and databases).

- *Pure software modules* with interface to a software platform capable of running PaP application software. Typical software modules provide: Teleservice functionality, service and network management functionality, resource and QoS handling functionality, network protocol, routing and switching functionality and also plug-in and plug-out functionality.

Pure hardware modules are not feasible in the context of dynamic PaP. PaP components will exist together with components that do not have the PaP functionality. These are denoted as non-PaP components. In the following it is assumed that all PaP components interact via a common DPE (Distributed Processing Environment. PaP components can interact with non-PaP components. Figure 1 shows a general PaP component model.

A PaP component can *contain* other PaP components. All PaP components have a relationship to *PaP support*. The PaP support is based on database support. All PaP components can have a local data store

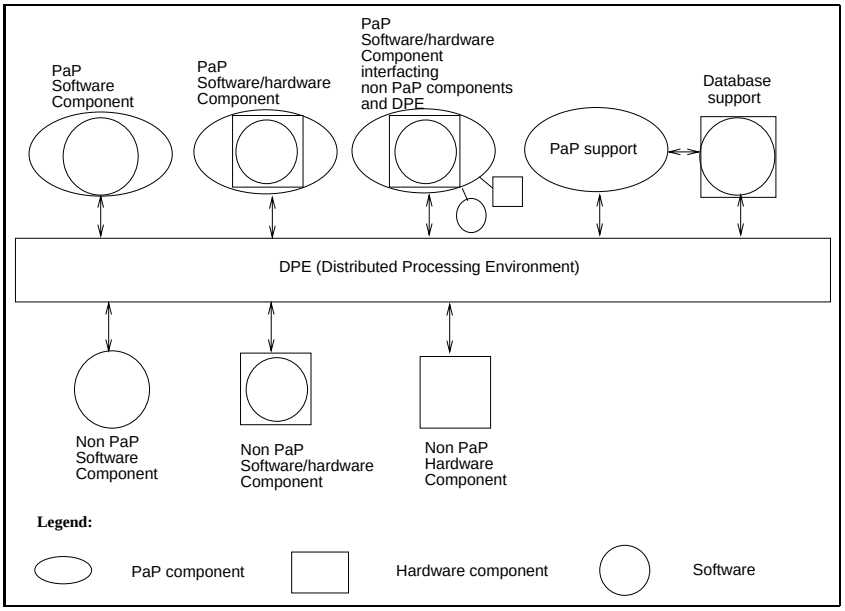

Figure 1 A general component model.

not indicated in the figure. The logical relationships between the PaP components is partly related to how PaP is solved and partly to the specific functionality of the system.

As will be discussed in next section, the PaP *component* functionality is defined by a *functional object model* consisting of *functional PaP objects*. The relationship between two PaP components is the result of the external communication between the functional PaP objects that are used for the "realisation" of the PaP components. A functional PaP object is an instance of an PaP object type. More details on this subject will be given in Section 4.

3.2 Requirements to the PaP system

PaP, as discussed here, is a technology intended for systems that shall be:

- flexible and adaptable,
- robust and survivable, and
- 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, management, evo-

lution and maintenance. However, system structure and functionality to ensure a dependable and traffic handling capable solution are also important system properties. Note that neither the three "qualities" listed above, nor the requirements listed below are strictly disjoint. PaP properties that contribute to **flexible and adaptable** systems are:

1. A system structure and functionality that is not fixed, but which allows structural operations like adding, moving and removing components on demand, blocking/deblocking of components in response to failures, etc. Hence, it should be possible to add and change the functionality of the entire system by adding or changing (software) components.

2. New components and their external capabilities are found automatically. When a new component is plugged into a system, the system shall become aware of its presence. For instance, a new transmission line plugged between two routers should be recognised by the network and its capabilities, e.g. capacity, should be propagated to the routing algorithm of the network.

3. Continuous adaptation to the environment and operation strategies/policies. These are responses of a PaP system to changes in its operational environment and the conditions and rules it operates under. The system shall for instance be able to adapt to a change in the volume and interest of the traffic offered. Strategies/policies are set by the owner/operator of the system to guide its behaviour. Examples of strategies/policies are: priorities between services and/or users, and routing in the transport network.

4. Recursive PaP functionality. A PaP system may be a PaP component of a larger PaP system, i.e. aggregation of components. For instance, a router may be a PaP component of a local area network. This network may again be a component of a corporate network. With this recursiveness in mind, Property 3) also applies to the components of a system. Hence, they should have the ability to configure themselves when they are added into a system or network (ideally without human intervention) and have the ability to adapt themselves to changes in their surroundings in the system or particular events.

To be **robust and survivable** a PaP system must:

5. Be based on a dependable distributed architecture encompassing both the resources and the functionality of the system. The system shall

not have a single point of failure, i.e. rely on unreplicated centralised resources and information. The system shall inhibit malicious and/or unauthorised modules to be plugged into the system.

6. Reconfigure itself in the presence of failures due to logical and physical faults. Failures due to physical faults will inevitably occur. The system shall be able to detect failed hardware components and automatically reconfigure itself so it can handle the workload and the functionality of the system to the extent that the physical resources in the system allow. Failures due to logical (software) faults are the more frequent. The system shall be able to detect these and to reinitialise failed components. It should also be able to prevent the propagation of errors in the system and import of errors from its environment.

7. Provide continuous operation. The system shall continue operation and shall not interrupt service provision more than strictly necessary under: plug-in and plug-out of components, changes in services, operation and maintenance policies and manifestation of physical and logical faults.

To be **QoS aware and to provide resource control** a PaP system must:

8. Negotiate QoS and allocate resources in an optimal way. The basic types of resources of the system will be transfer (transmission) capacity, storage capacity and computation power. Limitations of these types of resource will restrict the QoS provided for a given workload and service mix. The system (and its components) shall negotiate with the users of its resources to ensure a good/optimal utilisation of its resources under the operating policies of the system.

9. Provide monitoring of the resource utilisation and dynamically take actions to improve it. Improvement of resource utilisation includes, reallocation of workload to other network elements and rearranging the location of pure software components. With respect to physical resources, it is expected that the system should be able to advice network operators and service providers about resources needed and changes required in the physical structure.

4 A PAP REFERENCE ARCHITECTURE

4.1 The functional object model

The PaP components are modelled and designed using object-oriented principles. 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. ISO's reference model for Open Distributed Processing (ODP) [Duts96] defines the enterprise, computational, information, engineering and technical viewpoints. These viewpoints are tools for some kind of separation of the total system complexity. The computational and the engineering viewpoints are the viewpoint of *primary interest* with respect to PaP. The PaP components are basically engineering viewpoint objects. However, the PaP components have a computational viewpoint specification by the PaP functional objects, which are basically computational viewpoint objects.

We consider behaviour specification to be a computational aspect independent of the language used to express it. This because PaP always involves behaviour. The PaP of pure information will also be described as a part of the dynamic functionality. The computational model will also model the information which is subject to dynamic changes caused by the behaviour. In the PaP context, information models are supplementary models supporting the behaviour models.

A *functional PaP object* is a model of an aspect of a PaP component focusing on some behaviour aspect of the component. 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, i.e. to change their type,
- 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 2. 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 actors are the entities "realising" the PaP functionality objects.

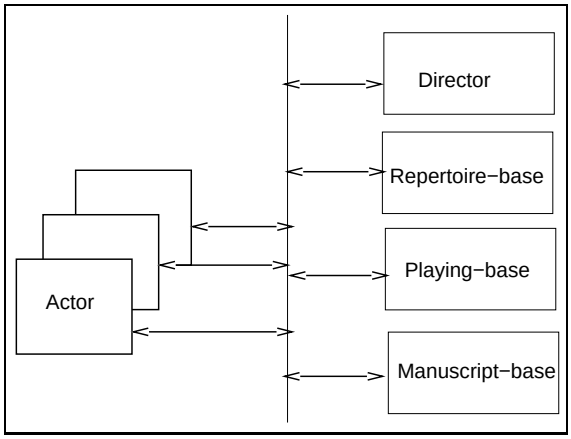

Figure 2 PaP system - Basic structure.

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 3.

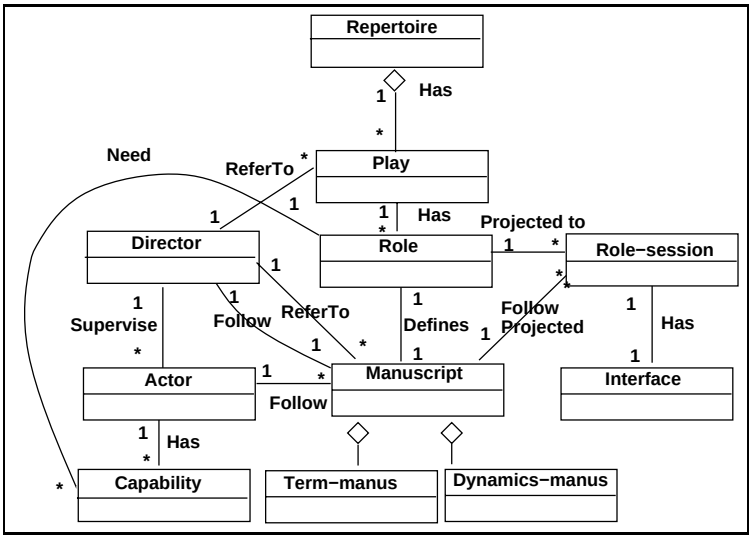

Figure 3 PaP concepts.

An *actor* is a generic object with a generic behaviour. Actors are 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. Capabilities are inherent properties of an actor, which can not be removed, replaced

or copied without removing, replacing or copying the entire PaP actor. The capabilities are the result of the available hardware functionality connected to the hardware executing the actor software behaviour, but also the quantitative aspects such as processing capacity. In other words, an actor is a generic abstraction of the whole or part of the functionality of a real-world PaP component as defined above. The actors capability will define which real-world PaP component functionality it is able to act on behalf of.

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. An actor has three distinct behaviour phases: 1): the plug-in phase, 2): the play phase and 3): the plug-out phase. The director guides actors in the plug-in phase as well as in the plug-out phase. Important functionality related to the plug-in-phase is actor identification, actor access control, actor capability control and actor resource and QoS negotiation and allocation.

The initiative to the plug-in of an actor will come from another PaP object instance than the one that is to be plugged-in (i.e. an actor object instance playing the same or another role). The initiative to plug-out can come from the same object instance as the plug-out object instance.

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. The manuscript of the PaP-director is also a part of the manuscript-base. Behaviour and accordingly the manuscript is part of a *play*.

An actor is able to play various roles. A role is here similar to the theatre concept. The role is defined by a manuscript. Manuscript defines the total behaviour of an object. A *role session* is a projection of the behaviour of the actor with respect to one of its interacting actors. The role behaviour is specified by an instance of a manuscript. The manuscript specifies: cooperating PaP objects, how to reach the cooperating PaP objects, the interactions with the cooperating PaP objects and internal behaviour resulting from an incoming interaction.

There are two types of manuscripts: *term-manus* and *dynamics-manus*. A term-manus is a dictionary of terms which might be referred to by actors during their interactions to avoid ambiguity and misunderstanding. A dynamics-manus describes the overall dynamic characteristics of an actor. The dynamics-manus will also comprise the definitions of inter-

faces to be used during the play as well as needed capabilities. The dynamical behaviour is specified by an Extended Finite State Machine (EFSM).

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.

4.2 PaP support functionality

The following functionality is part of our dynamic PaP system: 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.

Play plug-in involves the updating of Repertoire-base and the Manuscript-base. The Play and role definition is assumed to have been done properly before the plug-in. The validation of possible conflicting interactions with other plays is a part of the new play plug-in problems to be considered. We denote this as play interaction problem. The service interaction problem [Najm99] discussed for years in the Intelligent networks community is of a similar nature.

Play-changes plug-in involves new object types and modified object types. Problems related to conflicting behaviour is as discussed for Play plug-in above. External changes must be considered as play plug-in. The challange is to keep the system operational during the play-change installation period.

Dynamic detection of need of actors, plays and roles is specified in a manuscript. This is the "dynamic resource determination" solution. The advantage with this solution is that it is dynamically determined if, when, and how many actors and plays are involved. Another possible solution can be to statically determine the needs for actors' prior to the playing of a certain play. In this case all need for, and allocation of actors and plays is done before the play start. The main advantage with this solution is to assure that all needed resources are available when starting a play. This is the "static resource determination" solution. A third solution, the "combined resource determination", may be to allow use of a combination of the two other solutions.

Actor plug-in is the creation of generic objects capable of playing various roles. An actor pending for play is initiated. An actor needs an actor-id. An actor is registered in the playing-base with a location-id and also its defined capabilities.

Actor behaviour plug-out is necessary procedures for taking an actor out of a play. This involves other actors and the playing-base. *Actor plug-out* is the local removal of actors pending for play and also the updating of the playing-base. *Play-plug-out* is the removal of the play from the repertoire-base and also the manuscript-base. The semantics of *Actor change behaviour* is dependent of the nature of the new behaviour. If the role is part of an existing "old" play, then actor change behaviour is equivalent to *Actor behaviour plug-in*. If the role is part of a brand new play, then actor behaviour plug-in must be preceded by play plug-in or play-changes plug-in.

The functions: *actor behaviour plug-in*, *actor play* and *actor behaviour plug-out* comprise the initialisation of a generic actor pending for a play, performing the real play, and finally making the actor pending for a new play. This functionality is denoted as the *basic PaP functionality* and is briefly described as follows. The actor is initialised by first activating its PaP-director. An actor negotiates with the PaP-director in order to obtain its behaviour. The negotiation is related to the limiting capabilities of the actor, the needed capabilities of the role and the optional choices of the role defined by the manuscript. Once the negotiation has been completed, the PaP-director will create an instance of a behaviour manuscript object with all necessary parameters bound particularly for the actor and then send it to the actor. The PaP-director also acts as a binding object which helps to establish communication or interactions among actors. After receiving an instance behaviour manuscript from the PaP-director, an actor will immediately 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.

5 COMPONENT CATEGORIES AND CAPABILITIES

The PaP component can be classified as clients and/or servers. A server can be one of more clients for other servers. This gives a *tree-structured* system of client-server relationships. A resource is a server with limited capacity. Resources are important entities in a PaP system. A resource has a defined capability with respect to a service and quality of service (QoS). The PaP components can be categorised as

- *single objects*,
- *local objects aggregated* and
- *global objects aggregated*.

The single objects category is the simplest case where just one actor is plugged into a given play. An example is a new terminal server. For local objects aggregated, one component is plugged by using several actors at the same location. Examples are user nodes, network nodes and server nodes. In global objects aggreated, one component is plugged by using several actors at different locations. One example is a server requiring new software components at existing clients. In general *a teleservice* is *global* and corresponds to the result of behaviour performed by several distributed objects having various roles. Each teleservice is seen as a play. Each new play adds some new roles. Teleservices can not be defined separately and easily combined.

A capability has previously been introduced as the ability and/or power to perform functions or provide information. A capability of a PaP component is an:

- *inseparable*,
- *non-replaceable* and
- *non-replicable*

by the system itself functionality or attribute of the component. An attribute may be (private) information the component holds as well as performance characteristics. Hence, the functionality and parameters given an actor during the plug-in and during the play according to manuscripts are not capabilities. A component may have several capabilities. A certain set of (provided) capabilities is a prerequisite for filling a role as previously discussed.

The capabilities of an actor represents inherent, basic features that can be used to perform the roles assigned to it. The use of capabilities may be specified in a manuscript dynamically given to the actor, but the capabilities themselves can not be specified by such manuscripts. If an actor is seen as a machine that can be programmed using a manuscript language, then the semantics of this manuscript language will be related to the capabilities. Loosely speaking the capabilities provides the "instruction set" of the actor. They are not programmable (by dynamic PaP operations), but may be used as primitives in manuscripts.

The concept of capabilities are most easily understood for *combined hardware/software components*. In this case, the functionality of the hardware is embedded within the actor. The software part of the actor uses the functionality of the hardware to play roles in a system. As one example consider a hardware crypto module (e.g. a PCI board) and its accompanying software as a PaP component. The ability to perform fast and strong encryption (and decryption) based on the functionality

of the hardware is capabilities of the corresponding actor(s). Actors from this component may take a variety of roles like encryption engine, decryption engine, digital signature generator and verifier and random number generator.

Pure software components also have capabilities, although these do not depend on a physical item. In these components the capabilities will be defined by:

- *Proprietary code.* This code is embedded within the actor and can not be (legally) separated from it or replicated in other ways. A software implementation of crypto functionality mentioned in the example above is an example of such code.
- *Private information.* This may be personal information or company confidential information which the actor may use but not reveal when it performs its functions. Such information will be found in actors with "agent functionality". For instance, an "agent-actor" playing in an auction type of service will have the bidding policies of its owner as one of its capabilities.

Needed capabilities are used together with play roles and related manuscript, while *provided capabilities* are used together with actors and components. During play performance the needed capabilities must be checked against the provided capabilities. This checking can be closely related to the interaction specifications for a play. Each interaction specification has defined needed capability requirements to the interacting actor. Only actors which can offer the needed capabilities are allowed, and able to perform the specified interaction. The conditions to be fulfilled before an interaction is possible due to requested capabilities, may be expressed as calculated optional sets of capabilities.

Since interaction between different plays are allowed, this also means that both play identifiers (play-id) and capabilities must be globally unique.

6 CONCLUSIONS

An 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 results presented are now applied for the specification and implementation of a PaP tele-school application demonstrator.

References

[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.

[Bies98a] Andrzej Bieszczad and Bernard Pagurek and Tony White, Mobile Agents for Network Management, IEEE Commucations Surveys, volume 1 number 1, 1998, http://www.comsoc.org/pubs/surveys.

[Bies98b] Andrzej Bieszczad, S.K. Raza, Bernard Pagurek and Tony White, Agent-based Schemes for Plug-and-Play Network Components, Proceedings of the 3rd International Workshop on Agents in Telecommunications Applications, IATA'98, July 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.

[Najm99] Elie Najm, On Service Feature Interaction, Proceedings of Smartnet'99, Invited paper.

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

[ITU93] ITU-T, Q1204: Intelligent network distributed functional plane architecture, March 1993.

[ITU97] ITU-T, Intelligent network - Service plane architecture, September 1997.

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

[Tenn96a] David L. Tennenhaus,S.J. Garland, L. Shrira and M. Frans Kaashoek, From Internet to ActiveNet, Request for Comments, January 1996.

[Tenn96b] David L. Tennenhouse and David J. Wetherall, Towards an Active Network Architecture, Computer Communication Review, Volume 26 number 2, April 1996.

[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.

[TINA94] TINA Consortium, TINA-C Deliverable: Engineering Modelling Concepts, V2.0, December 1994.

[TINA95a] TINA Consortium, TINA-C Deliverable: Overall Concepts and Principles of TINA V1.0, February 1995.

[TINA95b] TINA Consortium, TINA-C Deliverable: Computational Modelling Concepts, V2.0, February 1995.

[TINA95c] TINA Consortium, TINA-C Deliverable: Information Modelling Concepts, V2.0, December 1995.

[TINA97a] TINA Consortium, TINA-C Deliverable: Service Architecture, V5.0, June 1997.