Plug and Play (PaP) for Telecommunications - Architecture and Demonstration Issues.

Finn Arve Aagesen
Norwegian University of Science and Technology

An extended version of this presentation is available at:
http://www.item.ntnu.no/~plugandplay/IConIT.pdf

Contents

Some general reflections
Vision, objectives and project idea
PaP architecture, design and demonstrator.
Summary and conclusions
The grade of network intelligence:

"the **efficient flexibility** in the introduction of new teleservices and the **efficient flexibility** in the execution of teleservices"

**Teleservice examples:** CS-telephony, IP-telephony, Intelligent Network (IN) - services, Web-services, E-mail, FTP, Mini-banks and admission control services, Video conferencing, Tele-school, Nomadic office, Cooperative work, Tele-medicine, E-commerce, Remote sensing and control, Digital TV.

The creation of **substance** needs some **basis**

Teleservices are specified and implemented with the support of some architecture and language basis
Recursive teleservice and basis functionality

Teleservices

Layer N

Layer N - 1

Specification basis

Implementation basis

One teleservice can enhance some other teleservice

Specification and implementation languages are often different

The “classical teleservice” evolution - One example view:


IN: Intelligent Networks (ITU Rec. Q. 1200)
SDL95: ITU’s Specification and Description Language
CHILL: ITU’s High Level Language

IN: Intelligent Networks (ITU Rec. Q. 1200)
SDL95: ITU’s Specification and Description Language
CHILL: ITU’s High Level Language

FSM based call handling

Clock-based call handling

Assembler

CHILL and automatic Assembler code generation

SDL95 and automatic “any” code generation

IN - based services
Diversity examples on tele-service architectures and language evolution


IN, TINA, WEB, Parlay, Agents
UML, SDL, JAVA, CHILL, C, C++, ANSI, XML

Complex tele-services specifications have states

A): Humans as applications


B): Programs as applications:

Network Management (SNMP), Service Creation and management (IN, TINA, Smart Networks, Active Networks)

State-less tele-service

The idea about a general state-less functionality cannot be extended beyond its very limited application within the transport service and to tele-services directly supporting Human users.
Evaluation directions for architectures and languages

One evaluation direction example:
1) The most popular and common Tele service should be the architectural basis for all kind of tele-services
2) The most popular and common specification language should be used for all kind of tele-services
3) The most common and popular implementaion language should be used for all kind of implementations

Another evaluation direction example:
1) The architectural basis should be adjustable to the needed power of the teleservice
2) The specification languages should be selected according to the needed expressive power, flexibility and executability
3) The implementaion language should meet the needed implementation power, flexibility and efficiency

PaP for Telecommunications

Motivation: How to handle
- the increasing heterogenity, complexity and diversity for technical telecom solutions
- that qualified personnel is the critical factor for development, installation, deployment, operation, maintenance and evolution of Telecom and Tele-service software

Idea: Dynamic Plug and Play (PaP)

Project: “Plug-and-Play for Network and Tele-service Components”

Results: PaP System solution (architecture, software, demonstrator)

Norwegian Research Council
**Static and Dynamic PaP**

**PaP component**: is some real-world active software or software/hardware module

**Static “Plug and Play”**: Components configure themselves at installation and provide services according to its predefined functionality

**Dynamical “Plug and Play”**: Components have a set of basic capabilities. Their functionality is decided during the plug-in procedure and can dynamically be changed during the lifetime of the component.

**PaP system - Required property classes**:

- **P): Flexible and adaptable**
- **Pb): Robust and survivable**
- **Pc): QoS aware and to provide resource control**

New basis for PaP functionality

Extended PaP functionality

Basic PaP functionality
A flexible and adaptable system (Pa) requires:

- A system structure and functionality that is not fixed (adding, moving, removing components and changing component functionality according to needs and capabilities)
- That new components, their external services capabilities and needs are found automatically (awareness of new components and capabilities, propagation of needed information about changes, propagation of needed new functionality)
- Continuous adaption to the environment and operation strategies/policies (new component functionality, new teleservices, new service and network management functionality, new policy functionality)
- Containment and aggregation

The functional architecture is based on a theatre metaphor

Theatre: A metaphor for concepts and functionality definition. Represents a PaP Domain and a global distributed stage.

Repertoire: The set of Plays that may be performed at the theatre. PlayingBase defines all actors.

Play: Defines a set of logically related functionality.

Director role-figure: The manager of plays, and supervisor for application role-figures. Constituted by an actor.

Application role-figures: The performers of plays. Constituted by actors playing roles.

Capability: A unique set of properties of an actor.

RoleSession: A dialogue between two role-figures.

Manuscript: The assigned behaviour, i.e. the defined role of a role-figure, constituted by an an actor.
One solution to the basic flexibility and adaptability requirements (Pa)

**Actors** “implement” **Role-figures** “implement” **PaP-components**

- **PaP-components** are composed from one or more interacting instances of **Role-figures**, which are instances of Role-figure types.
- An **Actor type** is a generic **Role-figure** type. An **Actor** (instance) will execute the functionality of a **Role-figure** (type) and then become a **Role-figure** (instance).

![Diagram of PaP-components and interactions](image)
PaP system – Basic instance structure

Legend:
The Director constitutes a Director role-figure, which Role is defined by a Manuscript executed by an Actor
PaP system functionality (1 of 2)

Context: Play must have been defined

PlayPlugIn: Make new functionality available as a new play version.

PlayChangesPlugIn: Change functionality of an existing play version.

PlayPlugOut: Remove a play version

ActorPlugIn: Establish a role session between two role-figures. An actor may be created implicitly.

ActorPlugOut: Terminate a role-session. An actor may implicitly be discarded.

ActorChangeBehaviour: Replace actor behaviour by new manuscript plug-in.

PaP system functionality (2 of 2)

Context: Play plugged in
Actors plugged in

RoleSessionAction: An information unit from one actor to another or to itself

ChangeActorCapabilities: Change specific properties for own actor

SubscribeEvents: Request to be notified for specified events.
The engineering model

A PaP system example

Legend:
Ai: Actor no i
Di: Director no i
AEEMi: Actor-environment-execution-module no i
B: PaP Boot

Legend:
Static available
Dynamic available
The Java Implementation Model

Java Terms used in PaP implementation

**Classes**
Used for implementation of all functionality, and also used for grouping of logically related information. Inheritance (‘extent’) is used for specialisation of generalised classes.

**Interfaces**
Used to gain access to same object instances from different objects. Defines the interfaces between objects and their environments.

**Objects**
Objects are the instances of classes that defines the executable system.

**Threads**
Used to separate different activities operating either independent of each other, or activities loosely coupled to each other.

**Java RMI**
Used as a common basis for communication between objects located within different Java Virtual Machines (JVM). The ‘rmiregistry’ is used for registration and identification of addressable entities of types PNES and AEEM.

The Distributed PaP Solution

The Synchronous communication model

Legend:
- Single node
- Java Virtual Machine
- Java Object instance (JO)
- JO communication using “local method” calls
- JO communication using “remote method” calls
- Queuing of “RoleSessionAction” and “SubscribeReport” type requests. All other request types are synchronous.
The Distributed PaP Solution
Addressing and routing values

- Local role session identifier
- Local Actor instance identifier
- Local AEEM instance identifier
- PNES instance identifier
- Entity type spec.

PaP Applications are made for validation and demonstrations

"Tele-School" - A Network based learning application
"Watcher" - A PaP Support activity monitor
"TestPaP" - A tool for automatic testing of PaP Support
Actor structure of Play: Real-time-lecture

Legend:
- Actor using Manuscript
- Non-PaP
- The role-session user
- The role-session provider
- Role-session relation

Example Interactions between actors
Summary and conclusions

The PaP architecture has potential to improve software development, deployment, installation, operation, maintenance and evolution for complex telecommunication and tele-service functionality.

The specified flexibility and adaptability requirements has been met and demonstrated.

PaP solutions is based on available and portable technology (Java). Light weight solution for distributed, asynchronous, message based “soft” real-time applications. Executable software and documentation available at Web: [http://www.item.ntnu.no/~plugandplay](http://www.item.ntnu.no/~plugandplay)

Ongoing research related to extensions of the PaP architecture to meet the requirement classes Pb) and Pc). Present Dr.ing research topics: 1): Teleservice modelling, 2): Fault tolerance and intrusion prevention, 3): Mobility and 4): Capability handling.

Advantages of using PaP (1 of 2)

1): Development of PaP Applications

- Flexibility in application modelling (**“Composition” of Plays and Manuscripts from Role-sessions**)
- Transparency in distributed solutions (**Use of Java/RMI**)
- Portable (**Use of Java**)
- Mobile agents become possible (**Uniform operational context for Actors, Java**)
- Easy monitoring and controlling (**Almost all PaP function requests served by Director**)

2): Deployment and Installation

- Easy installation and maintenance of installations (**Web-server, PlayPlugin, PlayChangesPlugin and PlayPlugOut functions**)

Ongoing research related to extensions of the PaP architecture to meet the requirement classes Pb) and Pc). Present Dr.ing research topics: 1): Teleservice modelling, 2): Fault tolerance and intrusion prevention, 3): Mobility and 4): Capability handling.
Advantages of using PaP (2 of 2)

3): Operation

Dynamic change of behaviour at runtime
(Use of Play-plug- and ActorChangeBehaviour- functions)

Collaborative applications, in addition to client/server solutions
(Role-sessions and RoleSessionAction function)

Uniform execution environment for applications
(PaP Actor Support as common context)

Functional consistency assurance at runtime
(Repertoire-base, Play versioning, Playing-base)

Security (Standardised operational environments for applications. All PaP communication routing is known. Utilisation of Operating system and Java security mechanisms)

4): Maintenance and Evolution

Software modification and extension
(Play versions, Manuscripts and Role-session definitions)

Compact solution (requires only PaP Support System (50 classes, 120kb) and JRE™, in addition to the PaP application)

Executable software and documentation available at Web:
http://www.item.ntnu.no/~plugandplay