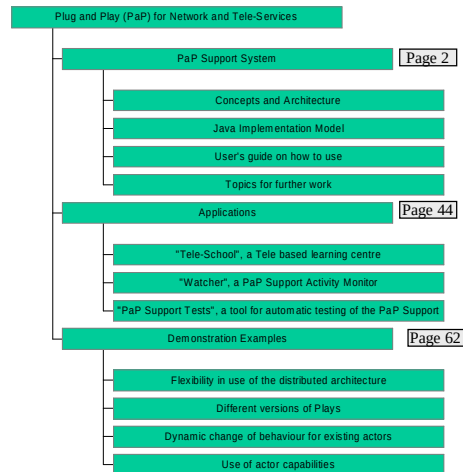


## Plug and Play (PaP) for Network and Tele-Services Documentation Survey, Top Level



16.10.00

Plug and Play (PaP) for Network and Tele-Services

1

NOTE 1: The notes pages has been extensively used for additional comments to the figures, and is therefore considered as a part of this PaP documentation.

NOTE 2: The last section about “Demonstration examples” is not complete.

NOTE 3: All types of comments are requested.

## Start of “PaP Support” documentation

16.10.00

Plug and Play (PaP) for Network and Tele-Services

2

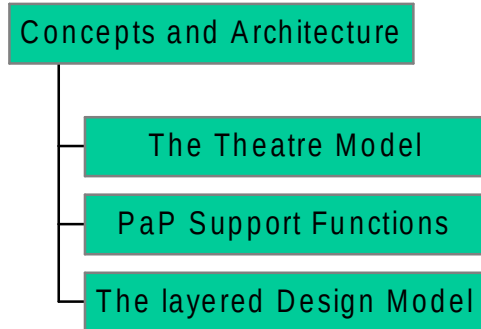
# PaP Support System

Documentation Survey (1 of 5), Overview

|                            |         |
|----------------------------|---------|
| PaP Support System         |         |
| Concepts and Architecture  | Page 8  |
| Java Implementation Model  | Page 19 |
| User's guide on how to use | Page 31 |
| Topics for further work    | Page 43 |

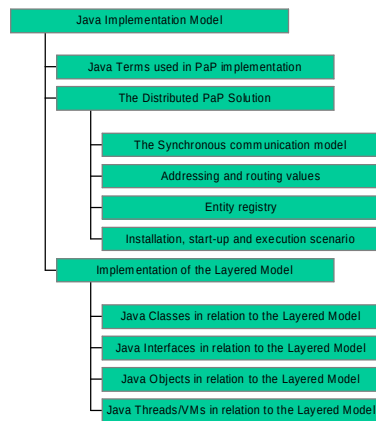
# PaP Support System

Documentation Survey (2 of 5), Concepts and Architecture



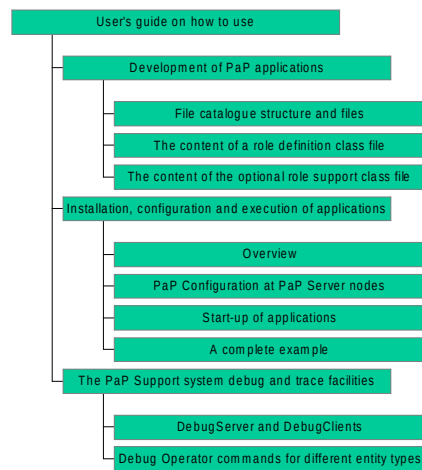
# PaP Support System

## Documentation Survey (3 of 5), Java Implementation Model



# PaP Support System

Documentation Survey (4 of 5), User's guide on how to use



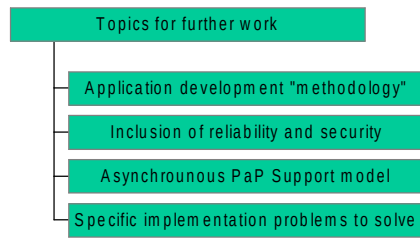
16.10.00

Plug and Play (PaP) for Network and Tele-Services

6

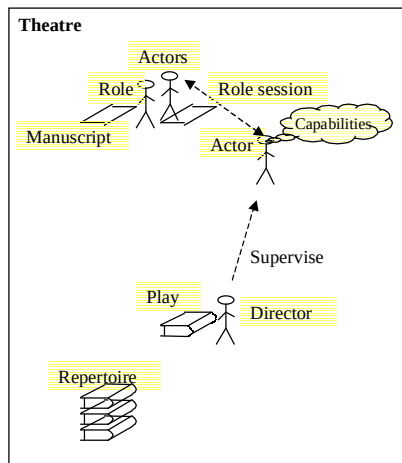
## PaP Support System

Documentation Survey (5 of 5), Topics for further work



# Concepts and Architecture

## The Theatre Model



**Repertoire:** Total set of 'Plays' available.

**Play:** A defined "autonomous" functionality.

**Role:** Partial behaviour of a 'Play'.

**Actor:** "Generic" performing active entity. Behaves according to its assigned 'Role'. Has 'Capabilities'.

**Manuscript:** The implementation of a 'Role' behaviour assigned to an 'Actor' when defining its behaviour.

**Capability:** Inherent properties of an actor.

**Role session:** Projection of behaviour of an 'Actor', in relation to one of its interacting actors.

**Director:** 'Repertoire' and 'Actor' manager. A guide to actors in plug-in/plugin-out phases.

## Concepts and Architecture

### PaP Support Functions (1 of 6)

The set of PaP specific operation types that may be used by the active entities (i.e. the actors) within the PaP context.

The operation types are grouped according to:

- Managing the availability of application functionality (i.e. the plays)
- Managing the existence of active entities in operational systems (i.e. the actors)
- Dynamic redefinition of actor behaviour (i.e. roles)
- Interactions between actors, actors capability change, and monitoring PaP activities

*Detailed description of functions and parameters is given in the implementation*

## Concepts and Architecture

### PaP Support Functions (2 of 6)

#### *Managing the availability of application functionality*

- **PlayPlugIn**(PlayId, PlayVer, PlayLoc)  
Play definition with specified identifier (PlayId) and version (PlayVer) to be installed at a Director, and to become known to actors. The play definition exists at the specified web-location (PlayLoc). The Director to install at is defined by the “HomeInterface” for the actor performing the request.
- **PlayChangesPlugIn**(PlayId, PlayVer, PlayLoc)  
Replace the existing play version with a new version. The location where the play definition is found may be another location than where the original version was located.
- **PlayPlugOut**(PlayId, PlayVer)  
Remove from director specified play definition, or a specified version of the play.

## Concepts and Architecture

### PaP Support Functions (3 of 6)

#### Managing the existence of active entities in operational systems

- **ActorPlugIn**(Location, RoleName, RoleSessionName, RequiredCapabilities, RequestedCapabilities, RequiredQoS, RequestedQoS, Visibility, SelectStrategy)

(NOTE: Parameters RoleSessionName, RequestedCapabilities, RequiredQoS, RequestedQoS, Visibility and SelectStrategy are not implemented)

Request the establishment of a role session to another actor according to the conditions specified by all parameters. The request may result in the creation of a new actor if no existing actor is able to satisfy the required conditions.

- **ActorPlugOut**(RoleSessionId)  
Request to remove an already existing role session. Internal conditions related to the involved actors may cause the termination of one or both of them.
- **ActorPlay**() (NOTE: Not implemented, assumed implicit at ActorPlugIn)  
Start behaving according to assigned manuscript.

## Concepts and Architecture

### PaP Support Functions (4 of 6)

#### *Dynamic redefinition of actor behaviour*

- **ActorBehaviourPlugIn(RoleName)**  
Define behaviour for an existing actor which only has a generic PaP behaviour assigned.
- **ActorChangeBehaviour(NewRoleName)**  
Request the change of behaviour for an existing actor. The actor may retain some or all of its existing internal and external state information, depending on defined relations between the previous role played and the new role to be played.
- **ActorBehaviourPlugOut()**  
Remove the defined behaviour for an actor. The generic PaP behaviour remains.

## Concepts and Architecture

### PaP Support Functions (5 of 6)

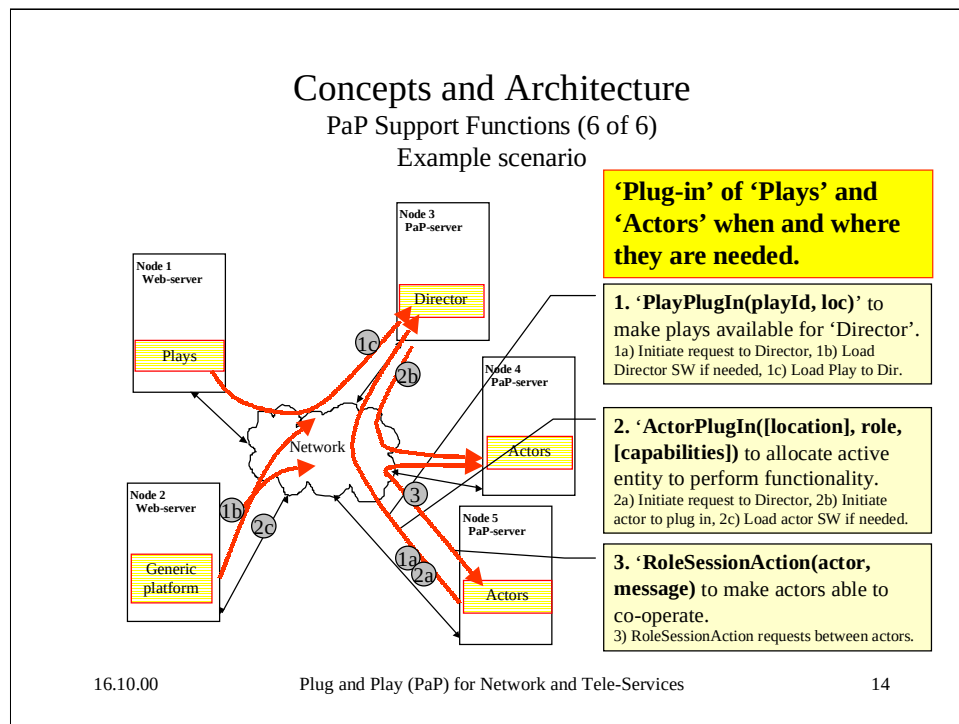
#### Interactions between actors, actors capability change, and monitoring PaP activities

- **RoleSessionAction**(RoleSessionId, MsgType, MsgParameters)  
Communicate specified message and additional parameters on the specified role session.
- **ChangeActorCapabilities**(ChangeType, CapabilitySet)  
Specify change (Set, Add or Remove) of the current capability set for actor. Each actor controls its own capabilities from its performing manuscript.
- **SubscribeRequest**(EventType, Scope, ApplicationTypes, WhenReport)  
Subscribe for specified event types to occur. EventType is related to PaP support or user defined functions to include, Scope specifies the set of actors to include, ApplicationTypes specified set of entity types to include, and WhenReport specified when or how often to report. 'SubscribeCancel' cancels a request, and 'SubscribeReport' is the reporting of occurred events sent to subscribing actor.

16.10.00

Plug and Play (PaP) for Network and Tele-Services

13



The figure shows example of the use of three of the defined PaP functions. An important issue is to show that “plug-in” occurs only when and where plug-in is necessary. “PlayPlugIn” needs not to be done before it is actual to use it through plugging in some actor that are going to play a role that the play defines. Actor instances need not to exist as long as they is not active or hold and state information.

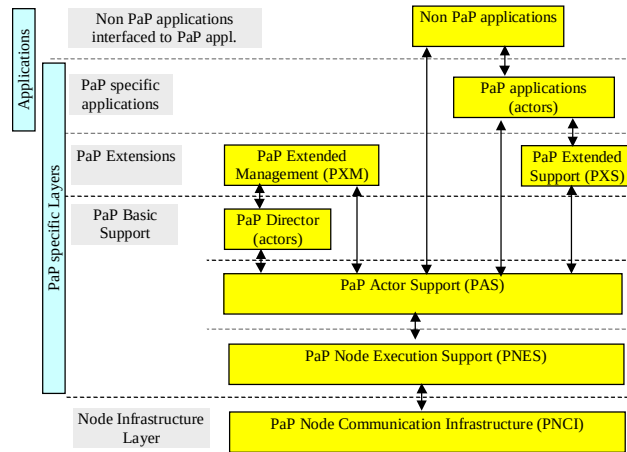
For PlayPlugIn the figure indicates that the request is initiated by some actor at Node 5 (by sending “PlayPlugIn” request to Director), and the request is served by the Director at Node 3. Play is either downloaded from the web-server at Node 1 or a reference to the play is hold. If necessary, at least at first PlayPlugIn, PaP Support software will be “down-loaded” from the web-server at Node 2.

For ActorPlugIn the figure indicates that the request is initiated by some actor at Node 5, the request is served by the Director at Node 3, and a new actor is created at Node 4. Also in this case some PaP Support software may be “down-loaded” from Node 2.

For “RoleSessionAction” only the two actors at Node 4 and Node 5 are involved.

## Concepts and Architecture

### The Layered Design model - Architecture (1 of 4)



16.10.00

Plug and Play (PaP) for Network and Tele-Services

15

A more detailed documentation of the Layered Model is given in the report  
 “Design Specification of the PaP support functionality, Dec. 1999”

## Concepts and Architecture

### The Layered Design model - Module description (2 of 4)

- **PaP Node Communication Infrastructure (PNCI)**  
Using Java/RMI and 'rmiregistry' for communication between nodes.
- **PaP Node Execution Support (PNES)**  
Makes it possible to run PaP at a node. Routing to/from other nodes and PASs.
- **PaP Actor Support (PAS)**  
Makes it possible to create and execute actors within the context of an operating system process. Additional functionality is routing between actors and PNES. Each PAS instance is a separate Java Virtual Machine (JVM) instance.
- **PaP Director**  
Responsible for management of plays, manuscripts and actors for its own PaP domain. Interacts with application actors.

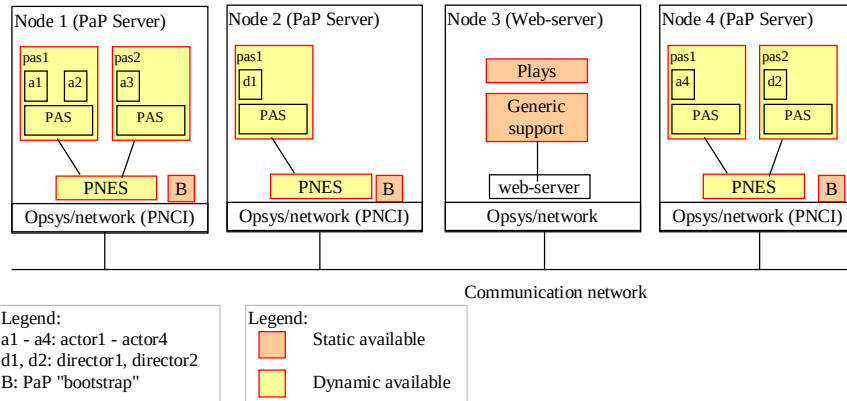
## Concepts and Architecture

### The Layered Design model - Module description (3 of 4)

- **PaP Extended Management (PXM)** (NOTE: Neither specified, nor implemented)  
Support of extended services not required for PaP support functionality, but to satisfy specified operational properties and requirements.
- **PaP Extended Support (PXS)** (NOTE: Neither specified, nor implemented)  
Required for the applications ability to utilise PXM functionality.
- **PaP Applications**  
The collection of application actors. Instances created/removed by using ActorPlugIn/ActorPlugOut support functions. Interfaces to PAS.
- **Non PaP applications**  
Functionality not defined according to Application actor requirements, but are allowed to communicate with actors, and to utilise the PaP Support functionality.

## Concepts and Architecture

### The Layered Design model - Operating PaP System Example (4 of 4)



16.10.00

Plug and Play (PaP) for Network and Tele-Services

18

## 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**

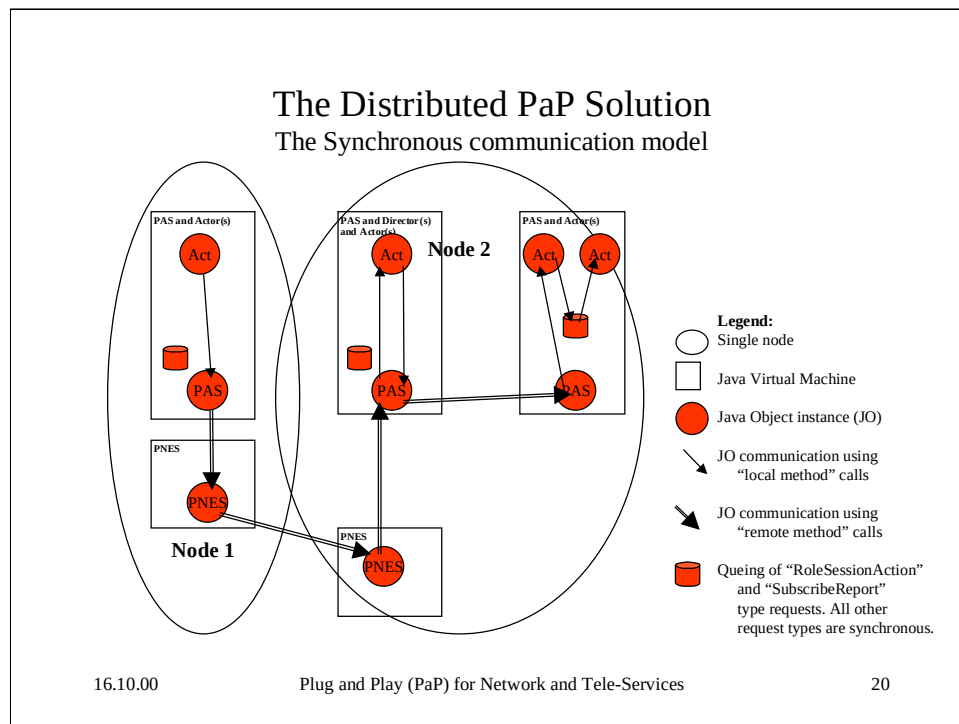
Objects are the instances of classes that defines the executable system.

- **Threads**

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

- **Java RMI**

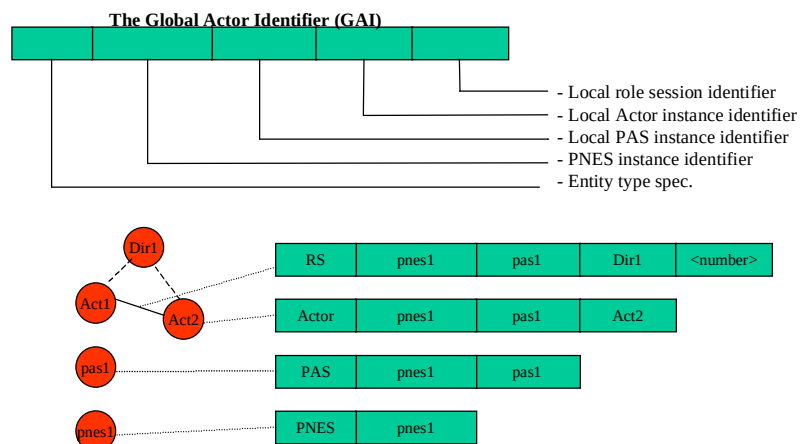
Used as a common basis for communication between objects located within different Java Virtual Machines (JVM). The 'rmiregistry' is used for registration and identification of addressable entities of types PNES and PAS.



- The Synchronous communication model means that a request from one object to another is executed before the execution continues at the requesting object.
- For the existing PaP implementation this is true for all types of requests, except for 'RoleSessionAction' and 'SubscribeReport', where the request is sent to the receiving actor's PAS instance and queued there until the actor is served. It exists only one common queue for all actors owned by the same PAS, and the actors are served according to the 'first-in-queue' principle.
- Actors only communicate directly with its owner PAS by using local method calls. PAS communicates directly with PAS instances located at same node and with the one and only PNES instance at same node, by using remote method calls. PNES instances communicate with PNES instances at other nodes by using remote method calls exact the same way as when PAS communicates with PNES. Using remote or local method calls are completely hidden from the application point of view, except that the server part must be defined as an extension of 'UnicastRemoteObject', and the client part must be an extension of 'Remote'. The Java code needed for remote object communication is generated automatically by using the "rmic" Java tool for the generation of the "...\_Stup.class" and "...\_Skel.class" files for the "PNES" and "PAS" modules.

## The Distributed PaP Solution

Addressing and routing values



16.10.00

Plug and Play (PaP) for Network and Tele-Services

21

The same address identifier type, i.e. the Global Actor Identifier (GAI), is used for addressing any kind of addressable entity. The four addressable types are PNES, PAS, Actor and RS (i.e. Role Session).

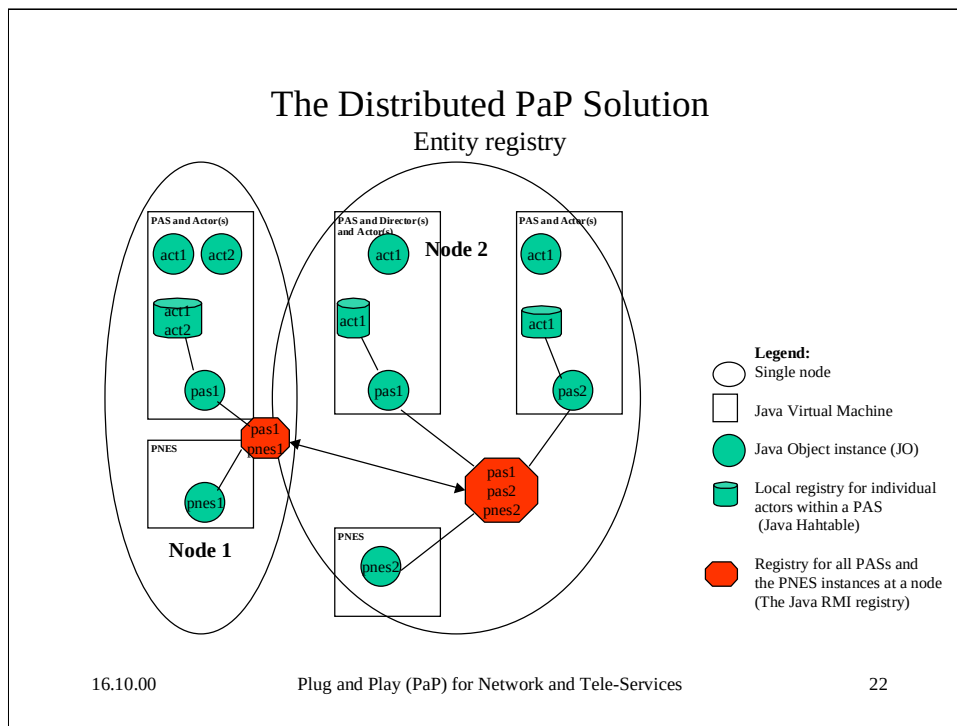
An GAI value is subdivided into different parts, and where the mandatory part denoting the addressed entity type (i.e. the values PNES, PAS, Actor, RS) determines which of the remainder parts that needs to be used. E.g. a PNES instance is uniquely identified by the PNES identifier value, while a role session is identified by a PNES identifier, a PAS identifier, a Director Actor identifier and a unique number within the specified Director Actor.

Using this identifier type, any PaP specific entity instance that needs to be addresses within the whole world can be done so.

The figure describes what parts a GAI is subdividet into, and shows which parts of a GAI that is used for the different PaP entity types.

(GAI is described in section 4.1 in the “Design Specification of the PaP Support functionality” PaP project report.).

GAI is implemented in the “GAI” Java class.



Some entity registry mechanism must be used to be able to find addressed PaP entities within the world, when that is necessary. For this purpose the Java “rmiregistry” mechanism has been used for registration of PNES and PAS entity types, while a PAS internal mechanism is used for the registry of all actor instances the PAS instance is the owner of.

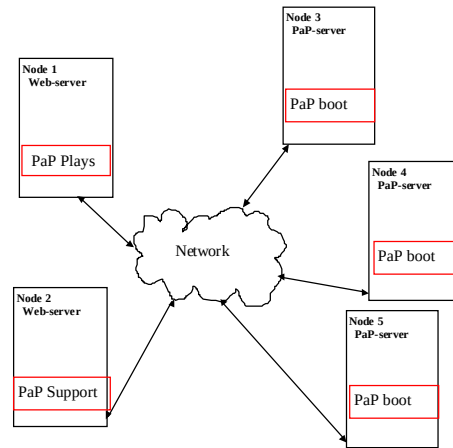
For both these registry mechanisms the GAI identifier for the addressed entity is the key for identifying a specific entity, and the result of an identification is a handle to a Java object representing the addressed entity instance.

The figure shows that this mechanism is used by PAS and PNES instances. There will be separate “rmiregistry” “instances” on each node, and the different rmiregistry instances do interactions with each other that is transparent for the applications.

Java applications have access to the Java “rmiregistry” mechanism through the “java.rmi.Naming” class.

## The Distributed PaP Solution

Installation, start-up and execution scenario (1 of 4), Initial installed system



16.10.00

Plug and Play (PaP) for Network and Tele-Services

23

- This scenario intends to describe what is involved in PaP installation, start-up and execution, without specifying the detailed procedures how this is done. Those procedures are described in a separate section, and as a part of the Demonstration examples.

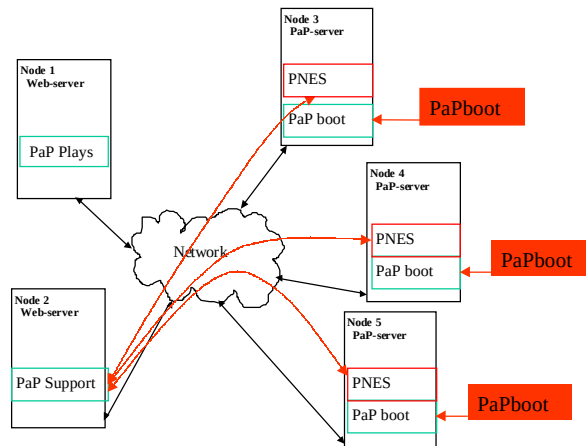
- Any Node that shall be able to run PaP applications must have some PaP specific software installed. The only thing that needs to be “statically” installed/available prior to running PaP applications is the PaP bootstrap. The only functionality for the PaP bootstrap is that it shall be able to start exact one PNES instance at a node. Note that neither the PaP bootstrap nor a PNES instance has nothing to do with any applications. So this is only a question whether or not a node shall be able to run PaP applications. A node that is able to run PaP applications is called a ‘PaP Server’.

- Before a node is able to run a PaP application the software that implements the PaP Support system and also the applications that shall be run must be available in some way. Existing implementation is based on that both PaP Support and the applications are available at some web-server.

- The figure indicates five nodes where three of them shall be able to run PaP applications and is therefore defined as PaP Servers with their ‘PaP Boot’ installed. Two web-server nodes are used to keep the PaP Support and PaP application software. Other example configurations could have been used. E.g. the same node may both be a PaP Server and a web-server. This means that PaP Support software, PaP applications and the executable PaP applications could all have been located at the same node.

## The Distributed PaP Solution

Installation, start-up and execution scenario (2 of 4), Start-up of PaP Support



16.10.00

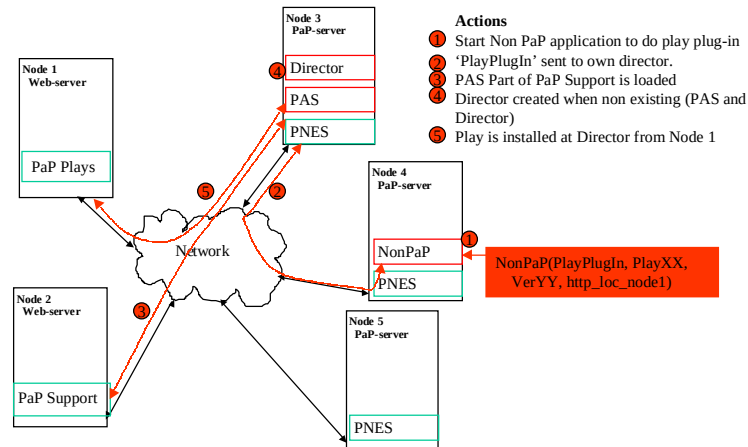
Plug and Play (PaP) for Network and Tele-Services

24

- Executing the PaP bootstrap will download the part of the PaP support software that is necessary to start a PNES instance at the node where the bootstrap is executed. A specification where the PaP support software can be located can either be specified when PaP bootstrap is executed, or it can exist in a configuration file used by the bootstrap. When a PNES instance has been started the node is ready to run PaP applications.
- The figure shows the preparation of three nodes for running PaP applications. The PaP Support software are loaded from the same web-server, however that is not necessary - the only requirement is that the PaP support systems must be compatible if they in some way will be related to each other (i.e. if operating within the same PaP Domain).

## The Distributed PaP Solution

Installation, start-up and execution scenario (3 of 4), Plug-in of “first” play



16.10.00

Plug and Play (PaP) for Network and Tele-Services

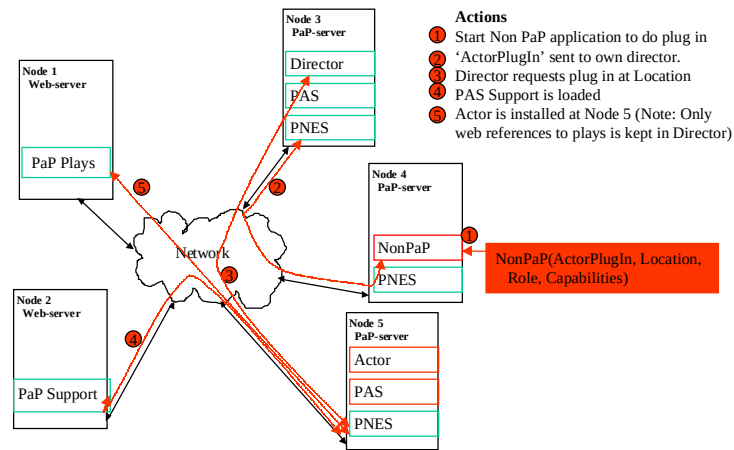
25

- Before any application actor is able to perform, the director actor that shall manage the application actor must exist. The director actor will implicitly be created if not existing when a request for play plug-in or actor plug-in occurs (i.e. the plug-in request is sent to the PAS instance where the director shall be located, and the PAS instance recognizes the plug-in request that always shall be routed to a director). If the director does not exist, the PAS instance will then create the director instance and assign it the director behaviour.

- The figure shows a request for a play plug-in that is initiated from a NonPaP application. The default location of the director for applications at 'Node 4' is 'Node 3' and the plug-in request will therefore be routed to 'Node 3' and the director will be created there. The PaP software necessary for running the generic part of PaP at Node 3 (i.e. the PAS instance and the generic part of the director) is downloaded from the web-server at 'Node 2', and the play to plug in is downloaded or referenced from 'Node 1'. Plug-ins have two operational modes: a) The play is downloaded and stored at the director when the play plug-in is performed, or b) only the web-reference is held by the director and the play is downloaded when actors are created and assigned manuscripts.

## The Distributed PaP Solution

Installation, start-up and execution scenario (4 of 4), Plug-in of “first” actor



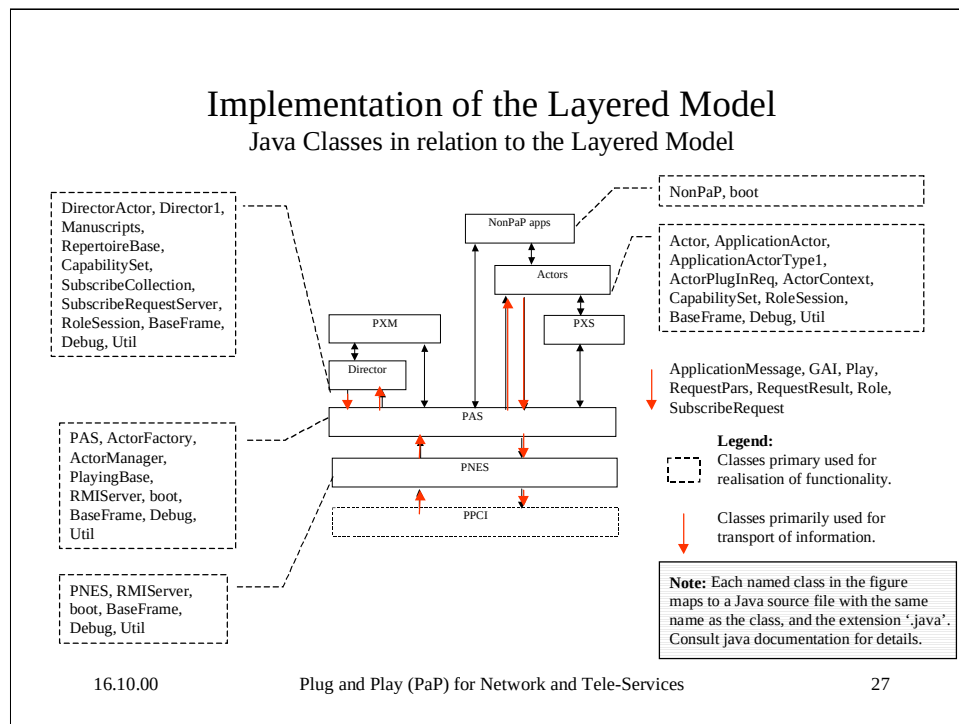
16.10.00

Plug and Play (PaP) for Network and Tele-Services

26

- The first actor to be plugged into a specific ‘PaP Domain’ must be plugged in from a Non PaP application, because there will not exist any actor that can do the plug-in. Succeeding plug-in of actors will normally be requested from other application actors. Prior to plug-in of the first actor, the play that contains the role specified at actor plug-in must already have been plugged in.

- The actor plug-in request is sent to the ‘home interface’ (i.e. the director) for PNES instance the request is routed via, and it is the director that serves the plug-in request. The director decides that the plug-in implies the creation of a new actor instance, and a request is sent to the PAS instance where the new actor shall be located (i.e. at Node 5). Since there exists no PAS instance with the specified identifier at Node 5 the PNES instance is responsible for the creation of that PAS instance. The PAS instance is then responsible for creating the actor instance. The necessary PaP generic software for running PAS and the actor instance will be downloaded from the PaP Support codebase, and the manuscript that implements the role the actor shall perform is downloaded from the web-location defined for the play that defines the specified role.

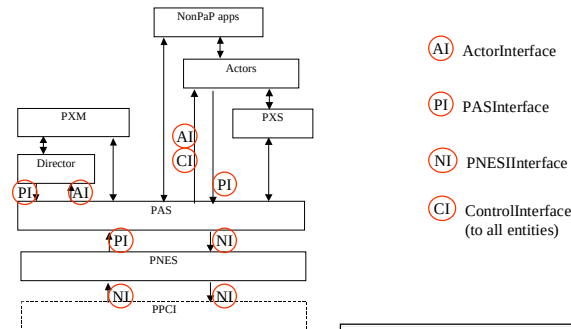


- The modules that together defines the PaP layered model are all implemented by a set of Java class definitions. Some of these class definitions are taken from existing Java packages, while other class definitions are PaP specific and therefore defined in a separate 'PaP' package. It is only the PaP package class definitions that are shown in the figure. Relations between classes are implicit through their definitions, and the figure is only to give an overview between modules in the PaP layered model and the corresponding implementation by a set of Java class definitions. The figure also indicates that some of the class definitions primarily is used for the implementation of PaP functionality, while others mainly are use as information carriers for transportation of information between Java object instances. Note however that also information carrier objects have their own behaviour definition.

- The figure is primarily intended to be a documentation survey for the 'javadoc' generated documentation that represents the detailed documentation of classes, their method definitions and data. That documentation is a presentation of extracted information from the Java code.

## Implementation of the Layered Model

Java Interfaces in relation to the Layered Model



**Note:** Each named interface in the figure maps to a Java source file with the same name as the class, and the extension '.java'. Consult java documentation for details.

16.10.00

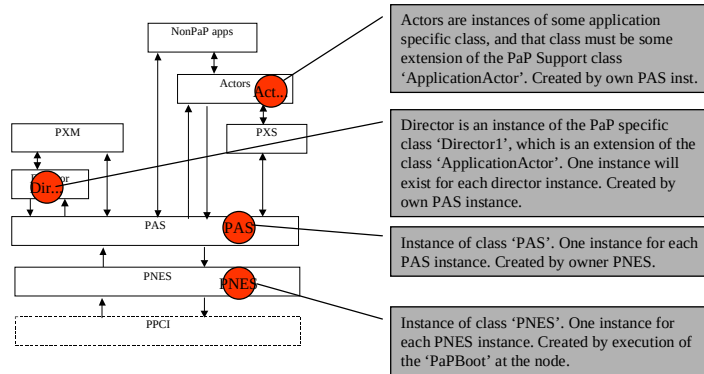
Plug and Play (PaP) for Network and Tele-Services

28

- Java interfaces are used when you do not want to know what class type that really implements an interface as seen from the user of the interface. I.e. that means that different classes may implement the same “user interface”.
- PaP has defined some interfaces ment to represent external user interfaces to different modules in the PaP layered model. The following interfaces exists:
  - ActorInterface: Common access point for triggering the execution of an actor with a received request.
  - PASInterface: Common access points for accessing a PAS instance either from any of its owned actors or from its owner PNES instance.
  - PNESInterface: Common access point for accessing a PNES instances either from its owned PAS instances or from any other PNES instance.
  - ControlInterface: A possibility for controlling actors from the debug interface. The same interface is also built-in as a part of PASInterface and PNESInterface.
  - DebugInterface (not indicated in the figure): is used as a part of the common debugging tool set.

## Implementation of the Layered Model

### Java Objects in relation to the Layered Model



16.10.00

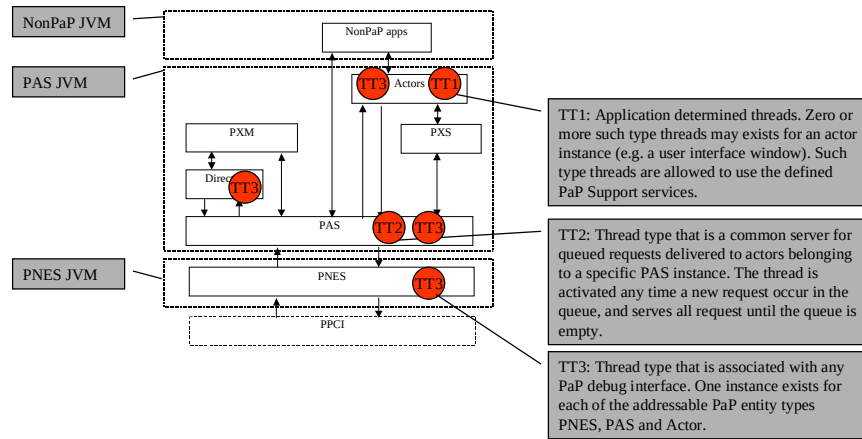
Plug and Play (PaP) for Network and Tele-Services

29

- The figure shows the main Java objects that forms the architecture for an operational PaP system. This means the objects that represents the PaP specific modules shown in the Layered Model. PXM and PXS has no Java object representations, because they are not implemented.

## Implementation of the Layered Model

Java Threads/VMs in relation to the Layered Model

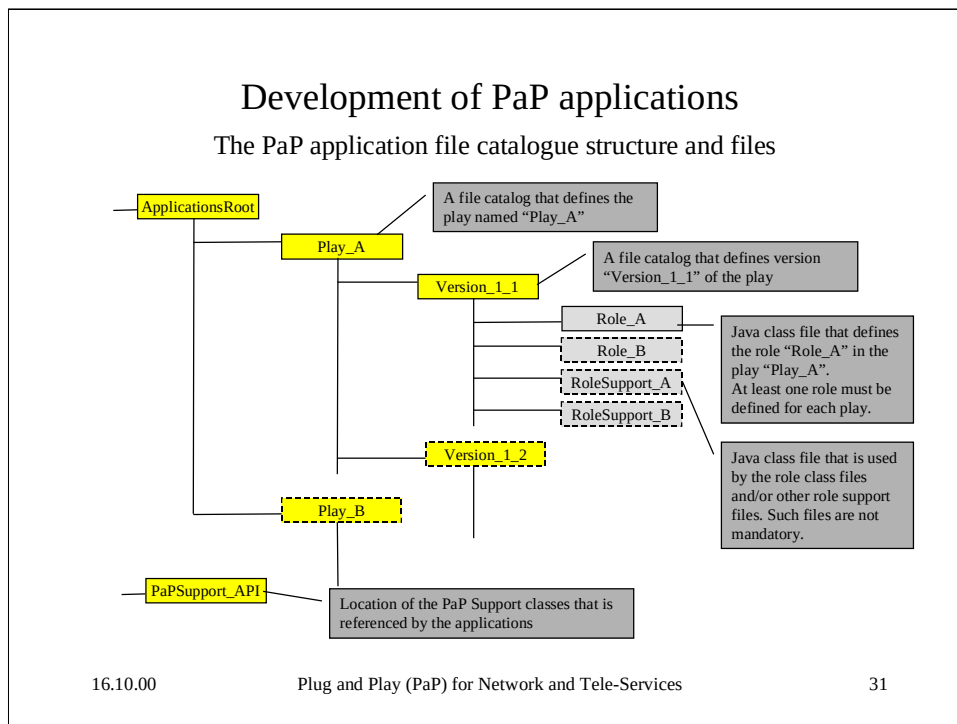


16.10.00

Plug and Play (PaP) for Network and Tele-Services

30

- Java Threads is in some way related to Java Virtual Machines (JVMs). This because when starting a JVM (i.e. means usually to start an application), the consequence is usually to have one running thread. For the PaP system this is a bit more complicated, because such a system will consist of multiple JVMs, and the different JVMs “communicates” by making use of java remote method calls. This gives the situation that there may exist JVMs with no threads at all, and other with multiple threads.
- The figure shows that each of the PaP entity types PNES, PAS and Actor has a separate “debug” thread. However, this thread is not necessary to obtain an executable system, and these threads will not exist in a “production” version of the system. This means that the only required threads that remains is the “common server” threads that exists within each PAS instance, and that is used to serve queued actor requests. In addition is the application specific threads that is outside the control of the PaP system. Note however that these application specific threads are allowed to make use of the PaP Support services. Note also that these threads is not synchronised with the PaP specific server threads and the PaP Support services that may be applied by both needs to be synchronised.



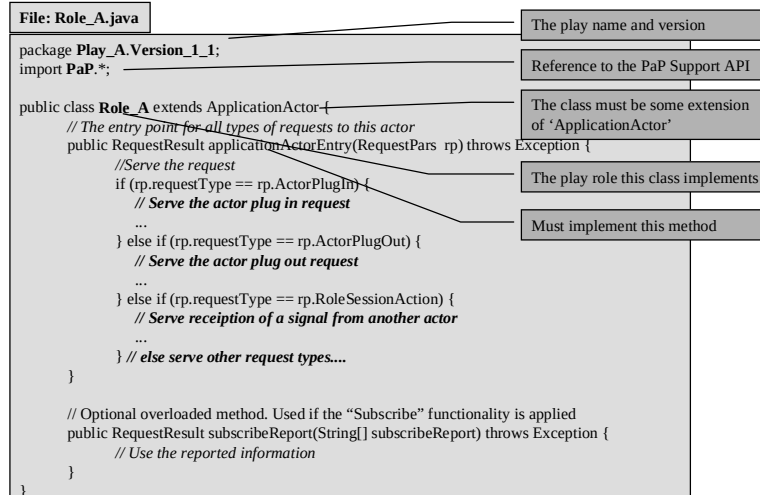
- Development of PaP specific applications is as development of any other Java application, except for some requirements related to the structuring of source files, and to some extents their contents in relation to their external interfaces. A PaP application will be related to its “ApplicationRoot” catalogue, and a sub-level catalogue defines the name of a PaP play. Several related and/or unrelated plays may be located within the same ApplicationRoot and related/unrelated plays may also be located within completely different ApplicationRoots. This means that it is completely possible to develop related plays at different locations.

For each play at least one version of the play must be defined. The version identifier is defined by a sub-catalogue to the catalogue that defines the play. All code defining a specific version of a play is located within this catalogue. The catalogue must at least have one Java code file representing the manuscript that implements a play role. There are no limitations on how many roles that can be implemented for the same plays, and the version catalogue may also contain Java source files that do not represent manuscript definitions, but are additional class definitions used by one or more manuscript definition classes.

- The figure indicates ‘Play\_A’ and ‘Play\_B’ located within the same root. Play\_A has defined versions ‘Version\_1\_1’ and ‘Version\_1\_2’, where the former version implements roles ‘Role\_A’ and ‘Role\_B’, and class definitions ‘RoleSupport\_A’ and ‘RoleSupport\_B’ are class definitions used by classes Role\_A and RoleB.

## Development of PaP applications

The content of a role definition class file



16.10.00

Plug and Play (PaP) for Network and Tele-Services

32

- There are some specific requirements that must be satisfied to make a Java source file to become a PaP manuscript that implements a play role.
- As the figure shows, these requirements are:
  - The class name of the class the file implements must correspond to the role name for the role the manuscript implements.
  - The class definition must be defined as a part of a java package that corresponds to '<play name>.<play version name>'.
  - The class must import the PaP Support definitions to be used.
  - The class definition must be defined as an extension of the 'ApplicationActor' class that is defined within the PaP support, or it may be an extension of some other class that directly or indirectly is an extension of the ApplicationActor class.
  - The class definition must implement the 'applicationActorEntry()' method that is defined as abstract in the 'ApplicationActor' class.
  - The 'applicationActorEntry()' method must implement the serving of the different PaP request types that are going to be served by this role. All relevant request types are defined in the PaP Support API definition.
  - The 'subscribeReport()' method must be implemented if the PaP Support subscribe service shall be used.

## Development of PaP applications

### The content of the optional role support class file

File: Rolesupport\_A.java

```
package Play_A.Version_1_1;

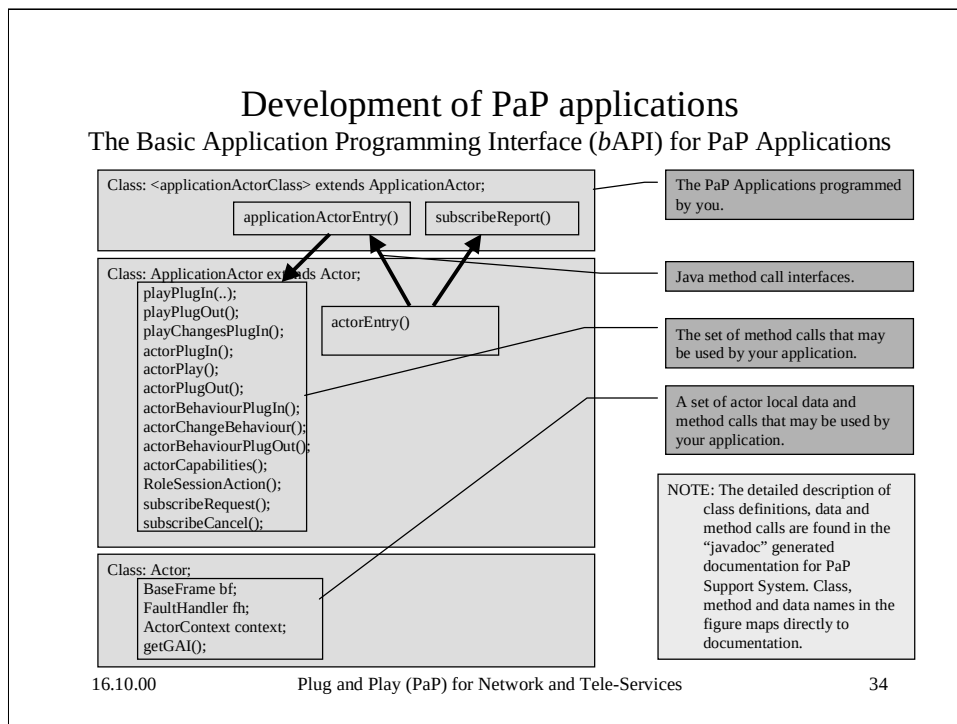
public class RoleSupport_A {
    // May be anything to support other role and role support classes
    // However, the functionality must not conflict with the PaP functionality responsibility
    ...
}
```

16.10.00

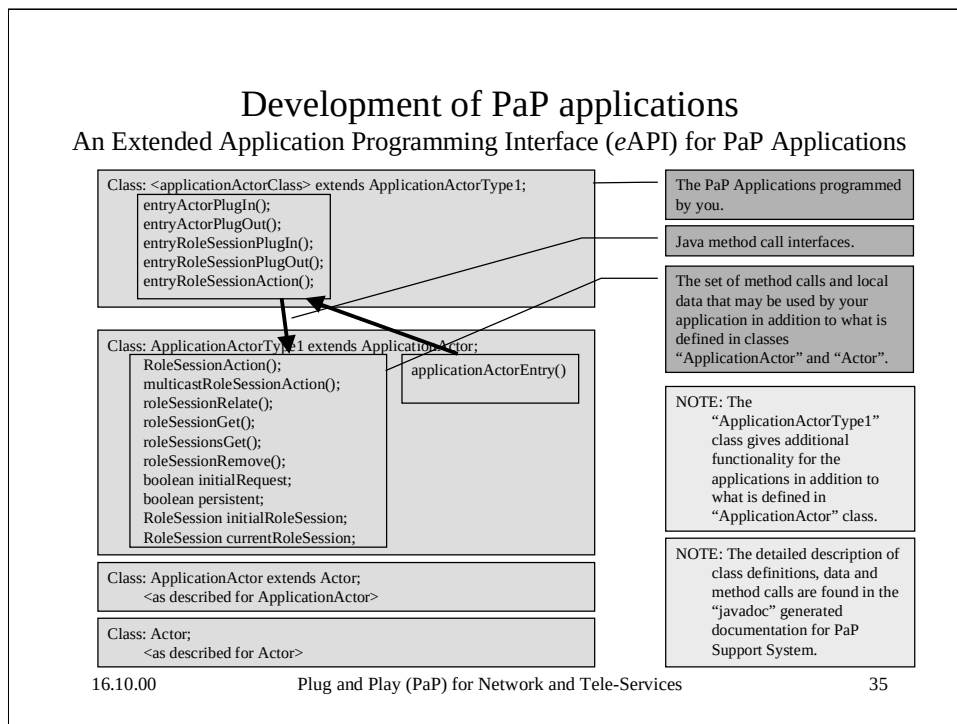
Plug and Play (PaP) for Network and Tele-Services

33

The only requirement to the content of role support class files is that they must belong to the same package as the role that make use of them.



- The Basic Application Programming Interface (bAPI) is defined as a set of java object method calls and a set of data that can be used by PaP applications.
- The figure is ment to give an overview of what parts of the PaP java code that represents the bAPI, and the detailed information is found in the corresponding 'javadoc' generated information that is a presentation of extracted information from the java code.



Extended Application Programming Interface (eAPI) is an extension of the bAPI interface. The functionality is implemented in the 'ApplicationActorType1' class, and that class is used as an extension of the 'ApplicationActor' class.

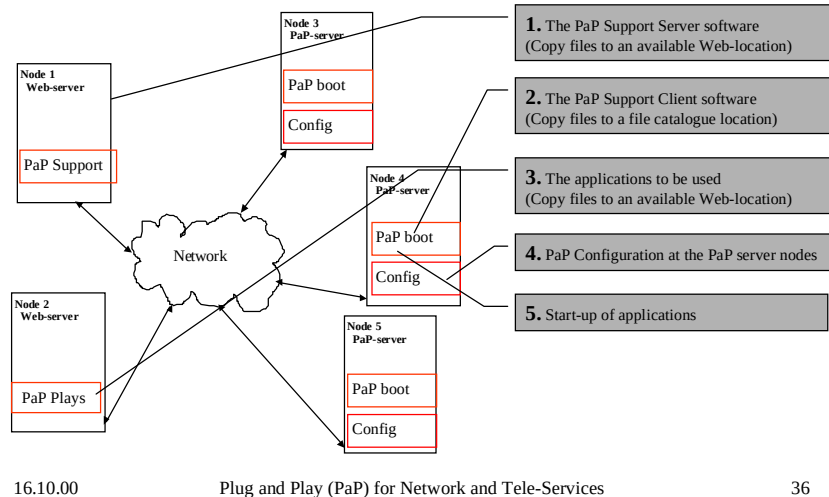
The figure shows an overview of what methods and data that is included in eAPI and therefore may be used by the applications. The functionality supported by eAPI in relation to bAPI is:

- Separation of different request types in separate method calls (entryActorPlugIn(), ...).
- Management of role sessions is a way that makes it easy for applications to operate as a multiuser server and to do multicast for message send.
- Support to have separate behaviour attached to each specific role session - if that is desirable.
- Possibility to do automatic termination of actor when no role sessions exists - if that is desirable.

The user may both make extensions to the eAPI, or even make a new "eAPI" if that is desirable.

The detailed documentation is represented by the 'javadoc' generated documentation for the class ApplicationActorType1.

## Installation, configuration and execution of applications Overview



Running PaP applications require a minimum of installation and configurations. The following installations is required:

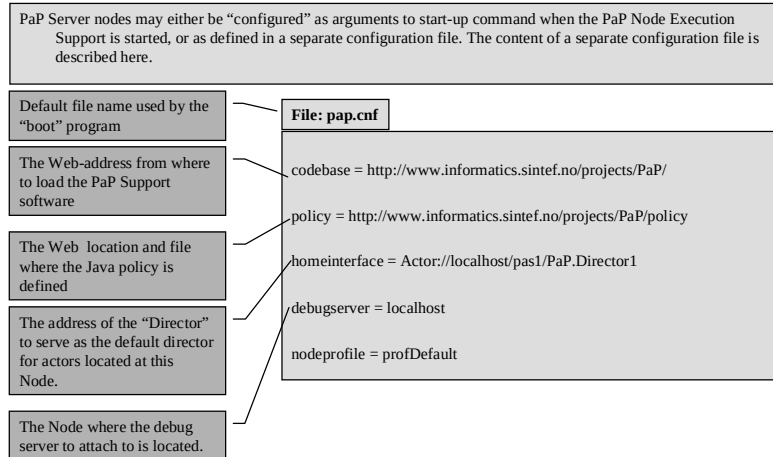
- PaP Support software and PaP Plays (i.e. applications to run) must be available at some web-address location.
- A PaP bootstrap must be available at any node that are going to run PaP applications. This PaP bootstrap consists of:
  - Java Run Time Environments (download from web, free of charge)
  - PaP boot software (a minimum set of PaP specific java classes)
  - A PaP node configuration file (A node specific editable ascii file)

When running PaP applications, at least the start-up of the first must be done from a Non PaP application (because there exists no PaP application to do it). After that, start-up of new applications and extensions of existing ones may either be done from running PaP applications or from Non PaP applications. Applications to be started need not to run on the same node as from where they are started.

A detailed description of how these things looks like in the implementation is given in the following.

## Installation, configuration and execution of applications

### PaP Configuration at PaP Server nodes



16.10.00

Plug and Play (PaP) for Network and Tele-Services

37

A minimum configuration of PaP Server nodes is necessary prior to running PaP applications. This configuration information can either be collected within one specific editable ascii file, or it may be specified at start-up at the node. The figure describes the content of the configuration file.

**'pap.cnf'** is the default configuration file name used by the PaP boot software. If another name is used, the name must be specified as an argument when starting PaP boot software.

**'codebase'** is a web-reference that specifies the URL location where the PaP Support software is located. The PaP Support software is a set of java class files.

**'policy'** is a web-reference that specifies the URL location where the java policy to be used is defined. The java policy is defined in a single ascii file.

**'homeinterface'** is a GAI address specification that specifies the default Director to be used by all actors that executes at this node. Note that one part of the GAI address value specifies a node name, and that node name may be another node than where the actors shall execute. Note that the value specified here is a default value, and the real value to be used may be changed both when creating PAS instances and actor instances.

**'debugserver'** specifies what node are going to be the default server node for this node. This implies that if there exists a 'DebugServer' instance at the node specified all debug trace information will be routed to the DebugServer instance at the specified node.

# Installation, configuration and execution of applications

## Start-up of applications

**Conditions that must be satisfied prior to running PaP applications:**

- Java Run-time environments must be present at the node where applications shall execute.  
Can be done by downloading from internet and installed.
- The PaP Support System software must be available at a web-address.  
Done by copying the set of files representing PaP Support System to a web-address.
- The PaP applications (i.e. Plays) must be available at a web-address.  
Done by copying the set of files representing the PaP applications to a web-address.
- The PaP “boot” software must be present at any node where PaP applications shall execute.  
Done by copying the set of files representing PaP “boot” software to a directory available at the local node.
- A PaP Node Execution Support (PNES) instance must be started at all nodes involved in executing the PaP application.  
Done by executing the procedure described below.

**Start-up of PaP Node Execution Support (PNES)**

If you want to keep a start-up window displaying debug and error listing, use the “startPaP” booter, else use “boot” booter.  
The use description for both these booters are:

***java PaP.startPaP <options> PaP.PNES or java PaP.boot [<options>] PaP.PNES***

The optional set of options are:

- configfile=<configuration definition file> (default: “pap.cnf”) (must be first if more than one option)
- codebase=<web-location where PaP Support software exists> (no default)
- policy=<web-location, including file name, for the Java policy definition file> (no default)
- homeinterface=<Director identifier for the default director for all actors controlled by this PNES> (no default)
- debugserver=<the node identifier for the debug server to attach to> (no default)

# Installation, configuration and execution of applications

## A complete example (1 of 2)

### Locations of software used in the example

- Java Run-time environments can be installed from: "<http://java.sun.com/products/>"
- The PaP Support System software exists at location: "<http://www.informatics.sintef.no/projects/PaP/>"  
The Java policy file to be used is located at: "<http://www.informatics.sintef.no/projects/PaP/policy>"
- The PaP application:
  - "School" is located at: "<http://www.informatics.sintef.no/projects/PaP/School>"
  - "Watcher" is located at: "<http://www.informatics.sintef.no/projects/PaP/Watcher>"
  - "TestPaP" is located at: "<http://www.informatics.sintef.no/projects/PaP/TestPaP>". "TestPaP" uses other plays located at: "<http://www.informatics.sintef.no/projects/PaP/TestPlay>"
- The PaP "boot" software is located at: "<http://www.informatics.sintef.no/projects/PaP/StartClient>"  
The zip-ed file located here must be unpacked to a directory available from the nodes to run PaP applications.

### Preparing the node for running the example software

- Install Java Runtime Environments if necessary.
- Install and adapt the "boot" software at your node if not already done:
  - Unpack the file "PaPBoot.zip" located in "<http://www.informatics.sintef.no/projects/PaP/StartClient>" in available directory (called "<PaP boot directory>" below).
  - Edit the file "pap.cnf" located in "<PaP boot directory>" to adapt to your needs
- Start the PaP Node Execution Support (PNES) instance by e.g. executing the command:  

```
java PaP.startPaP PaP.PNES
```

  
from the <PaP boot directory>. The PNES instance should appear with a separate command/trace window. Must be started at all nodes where PaP shall operate.

16.10.00

Plug and Play (PaP) for Network and Tele-Services

39

## Installation, configuration and execution of applications

### A complete example (2 of 2)

#### Starting applications

Application can either be started from other PaP application, from a NonPaP application, or from the PNES debug command window. In this example we will use the PNES debug command window.

#### Starting the “Watcher” application (Separate documentation of Watcher exists)

- If not already done Plug in the “Watcher” play by the command:  
***“PlayPlugIn Watcher v1\_1 <http://www.informatics.sintef.no/projects/PaP/>”***
- Plug in an actor instance playing the WatchServer role by the command:  
***ActorPlugIn Actor://<nodeName>/pas1/WS1 WatchServer***  
The actor instance should now appear both through its separate debug command/trace window, and its graphical window for presentation of monitored information.

#### Starting and using the “School” application (Separate documentation of School exists)

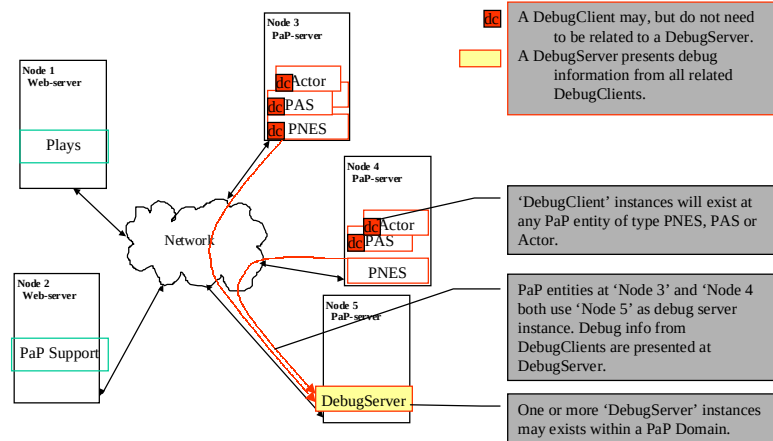
- If not already done Plug in the “School” play by the command:  
***“PlayPlugIn School v1\_1 <http://www.informatics.sintef.no/projects/PaP/>”***
- Plug in two actor instances playing the SchoolClient role by the commands:  
***ActorPlugIn Actor://<nodeName>/pas1/SC1 SchoolClient***  
***ActorPlugIn Actor://<nodeName>/pas1/SC2 SchoolClient***  
The actor instances should now appear both through its separate debug command/trace window, and its graphical window representing the initial Operator Interface window.
- Use the Operator Interface for using the two started School Clients. All PaP related activities is shown in the Watcher presentation window.

16.10.00

Plug and Play (PaP) for Network and Tele-Services

40

## The PaP Support system debug and trace facilities DebugServer and DebugClients



DebugServers and DebugClients are used as debug tool for trace of PaP activities, and to give operator command to PaP applications. Any DebugClient can be related to one specific DebugServer, and each DebugServer will present debug information for all its related DebugClients.

A DebugServer are capable to control the execution of the DebugClients that are related to it through the “Run” and “Step” controls. In addition the DebugServer will present reported trace information from all its related DebugClients. DebugServer is mainly implemented in a class with the same name.

DebugClients operates within the environments to the PaP entity it is attached to. Any of PNES, PAS and Actor type entities will have a DebugClient attached. Each DebugClient may have its separate user dialogue window that can be used to trace activities related to the attached entity, and gives the possibility for an operator to invoke PaP type actions. The set of action types available will differ for each entity type. An overview of different action types is not given in the following. DebugClient is mainly implemented by the classes ‘BaseFrame’ and ‘Debug’.

## The PaP Support system debug and trace facilities

### Debug Operator commands for different entity types

| Operator command type     | Actor | Director | PAS  | PNES | Debug Server | Comments                                      |
|---------------------------|-------|----------|------|------|--------------|-----------------------------------------------|
| ActorPlugIn               | X     |          |      | X    |              | Request for ActorPlugIn at actor              |
| ActorChangeBehaviour      | X     |          |      |      |              | Request for ActorChangeBehaviour at actor.    |
| ActorBehaviourPlugOut     | X     |          |      |      |              | Request for ActorBehaviourplugOut at actor    |
| ActorPlugOut              | X     |          |      | X    |              | Request ActorPlugOut at actor                 |
| PlayPlugIn                | X     |          |      | X    |              | Request PlayPlugIn at director                |
| PlayPlugOut               | X     |          |      | X    |              |                                               |
| PlayChangesPlugIn         | X     |          |      | X    |              | Request PlayChangesPlugIn at director         |
| RoleSessionAction         | X     |          |      | X    |              | Perform RoleSessionAction at actor            |
| ListRoleSessions          | X     | X        |      |      |              | List all role sessions for actor              |
| Status                    | X     | X        | X    | X    |              | Show status information for the entity        |
| ListActors                |       | X        | X    | X 1) |              | List all actors that belongs to the entity    |
| ListPlays                 |       | X        |      |      |              |                                               |
| Unbind                    |       |          | X    | X    |              | Unbind (i.e. discard) own entity              |
| Director Status           |       |          | X 2) |      |              | Show status for director                      |
| Director ListActors       |       |          | X 2) |      |              | Show all actors managed by director           |
| Director ListPlays        |       |          | X 2) |      |              | List all plays managed by director            |
| Director ListRoleSessions |       |          | X 2) |      |              | List all role sessions managed by director    |
| CreatePAS                 |       |          |      | X    |              | Create a PAS instance at this node            |
| GetKeywords               | X     | X        | X    | X    | X            | Display trace conditions                      |
| SetKeywords               | X     | X        | X    | X    | X            | Set trace conditions                          |
| Run                       |       |          |      |      | X            | Continue execution a specific number of steps |
| Step                      |       |          |      |      | X            | Step mode                                     |

X Command is defined

1) Listed in PAS debug window

2) Listed in Director debug window

The figure shows the set of debug operator commands that can be used by operators when controlling PaP applications through the use of the debug window for different PaP entity types.

The figure is just a summary overview and do not show any details about e.g. what arguments the different operation types require. That information is given in separate documentation.

## PaP Support System

### Topics for further work

- **Application development “methodology”**  
There is a need for a guide on how to write PaP Applications.
- **Inclusion of reliability and security**  
A strategy about how to include the reliability and security aspects in the implementation.
- **Asynchronous PaP Support model**  
A Synchronous implementation model exists. How can an asynchronous PaP Support model be included in addition to the synchronous model.
- **Specific implementation problems to solve**  
During the implementation of the PaP Support some practical and/or logical problems have been noticed. None of them are blocking problems, but solving them will contribute to increase the quality of the PaP Support system and to ease the use of it.  
Main problems are: 1) Long start-up time of JVMs, 2) Possible to use common libraries for multiple plays.

## Start of “Tele-School” application documentation

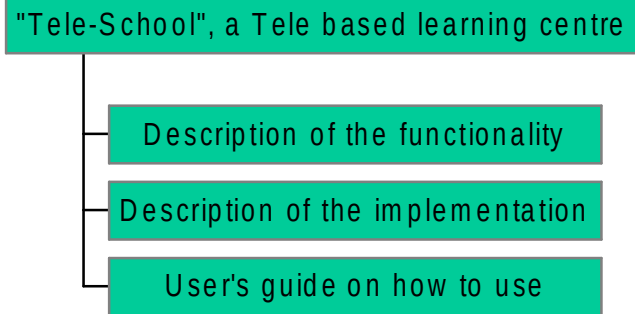
16.10.00

Plug and Play (PaP) for Network and Tele-Services

44

## “Tele-School”, a Tele based learning centre

Documentation Survey



Notes:  
•The requirements and functionality is defined in a separate report  
“Demonstrator - Requirements and functional description” (Dec. 1999)  
•For detailed documentation, consult ‘javadoc’ generated information from source code.

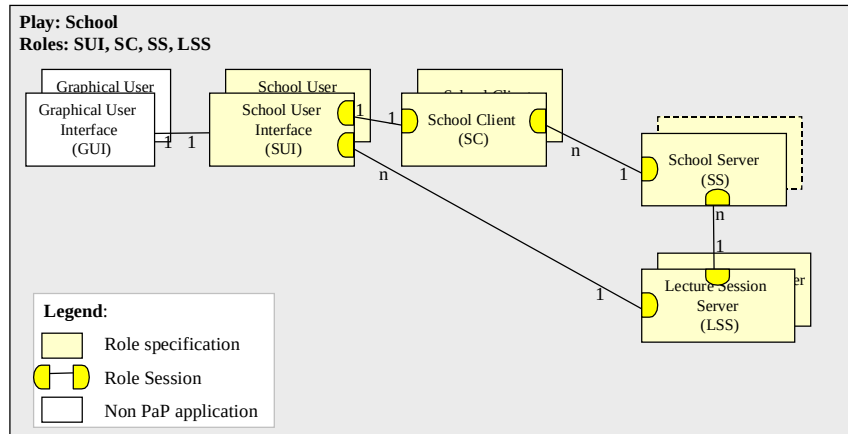
## **“Tele-School”, a Tele based learning centre**

Description of the functionality

- A Network based learning application
- Main functional terms are:
  - Teacher, Student, School, Course, Lecture
- Main implementation terms are:
  - Distributed solution, Mobility, Multimedia, Chat, Mail, News.
- Main functions are:
  - Real-time interactive lecture session (partly implemented)
  - Off-line learning session (not implemented)
  - Off-line teacher support to students (not implemented)

## “Tele-School”, a Tele based learning centre

### Description of the Implementation - Modelling



16.10.00

Plug and Play (PaP) for Network and Tele-Services

47

The “Tele-School” application is modelled as a Play with roles and role sessions. The figure defines the play ‘School’ and four different related roles. In addition a co-operating Non PaP application (GUI) is included in School.

The role ‘SchoolUserInterface’ (SUI) defines the user interface design specific parts, which make use of the graphical user interface implemented in GUI.

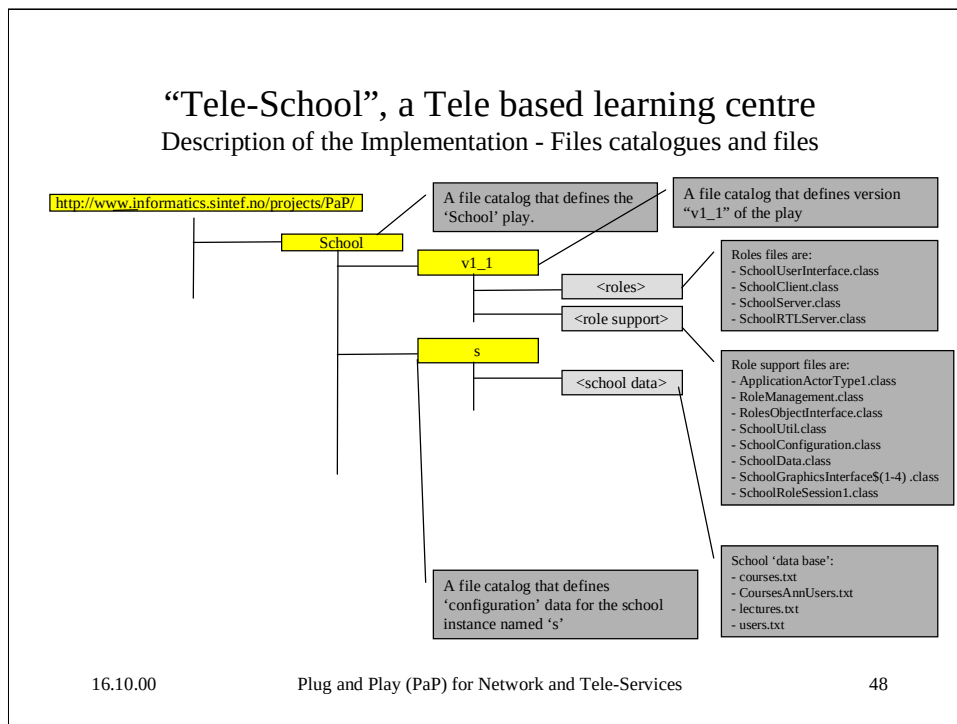
The role ‘SchoolClient’ (SC) defines the user interface independent part of the School clients functionality. School clients are Student and Teacher entities.

The role ‘SchoolServer’ (SS) is the common server for all clients (i.e. Students and Teachers) that make use of this school services. The figure indicates that there may exist multiple SS instances, and each instance will have defined its specific set of clients.

The role ‘LectureSessionServer’ (LSS) is a common server for all clients attached to a specific lecture session. That means that there will exist one LSS instance for each specific lecture when some student and/or teacher is attached to that lecture.

For role session definitions the figure indicates a one to one relation between each GUI, SUI and SC triple. Multiple clients and lecture session can be related to the same school server, and multiple clients can be related to the same lecture session.

*The model is also shown in the report “Demonstrator - Requirements and functions description”, Des. 1999*



The structuring of the School application software is according to what is defined for PaP applications - at top level a catalogue defining the play name ('School'), and at next level a catalogue that defines the version ('v1\_1').

In School role modelling four roles is defined. Each of these roles is implemented in a corresponding java class file (as shown in the figure).

Additionally, eleven java class files are defined as 'role support' files. Each of these files are used by one or more of the four role files and/or by other role support files.

An operational instance of a School will need 'semi permanent' information that defines administrative information related to the school instance. Such information is definitions of clients (teachers and students), lectures and courses, and definition of clients relations to defined courses. The figure defines a school named 's' and a set of four ascii text files that defined the specified configuration information. Since the School application has no functionality to manage this information, that must be done manually using a text editor to set the content of the configuration files. Note that other school instances may exists, using another name, or located at another web-location.

## “Tele-School”, a Tele based learning centre

User's guide on how to use

- **Install PaP Support software and the School application**

Both must have been installed according to generally defined directives as defined in the PaP Support documentation.

- **Configuration of School ‘configuration’ data**

Definition of students, teachers, courses and lectures must be done manually - before students/teachers are able to log-on and use the system.

- **Using the School application**

- **Student/Teacher log-on**

By starting an actor playing ‘SchoolClient’ role, either from a Non PaP application, or from another PaP application.

- **Student/Teacher use of School services**

By using the defined graphical user interface.

- **Student/Teacher log-off**

From the graphical user interface

## Start of “Watcher” application documentation

16.10.00

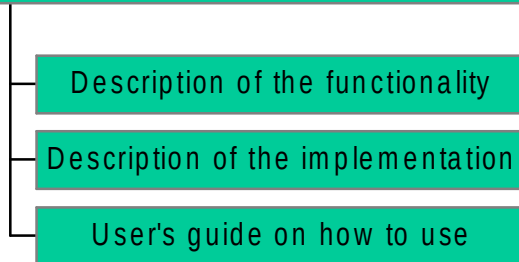
Plug and Play (PaP) for Network and Tele-Services

50

## “Watcher”, a Tele based learning centre

Documentation Survey

### "Watcher", a PaP Support Activity Monitor



*Note:*  
*•For detailed documentation, consult 'javadoc' generated information from source code.*

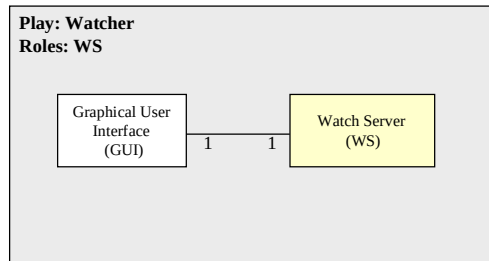
## “Watcher”, a PaP Support Activity Monitor

### Description of the functionality

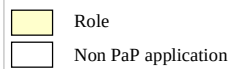
- **Display ongoing PaP specific activity**  
For a specific Director instance, display information about existence of actors and role session, and a log of all PaP type requests to the Director for the PaP domain. Multiple Watcher instances may be started to serve multiple Directors.
- **Demonstrate ‘PlayChangesPlugIn’**  
Plug-in of two different version of the Watcher play, and run actor instances performing from different play versions.
- **Demonstrate ‘ActorChangeBehaviour’**  
On request, change the performing role for an existing Watcher actor instance to another role.
- **Demonstrate alternative use of the ‘Subscribe’ functionality**  
Instead of continuously update of Subscribe information, do update only at 30 seconds intervals.

## “Watcher”, a PaP Support Activity Monitor

### Description of the Implementation - Modelling



#### Legend:



16.10.00

Plug and Play (PaP) for Network and Tele-Services

53

The ‘Watcher’ PaP application is implemented as one play with one defined role ‘WatchServer’ (WS). In addition a co-operating Non PaP application (GUI) is included in the Watcher.

The role ‘WatchServer’ includes all the necessary functionality, except for the graphical output user interface supplied. The functionality is mainly related to the use the PaP Support functionality ‘Subscribe’ to collect all PaP relevant information that shall be displayed for the users through the graphical user interface. An instance of a Watcher is capable to collect information related to one specific Director instance. Other Watcher instances may be started to present information from other Director instances.

Three types of information is presented:

- A list of still existing actors created since the Watcher was started.
- A list of still existing role sessions created since the Watcher was created.
- A log of executed PaP specific operations done since the Watcher instance was started. When the log reach a specific size it is cut off at the start.

A Watcher instance is terminated as a standard window application when pushing X-button.

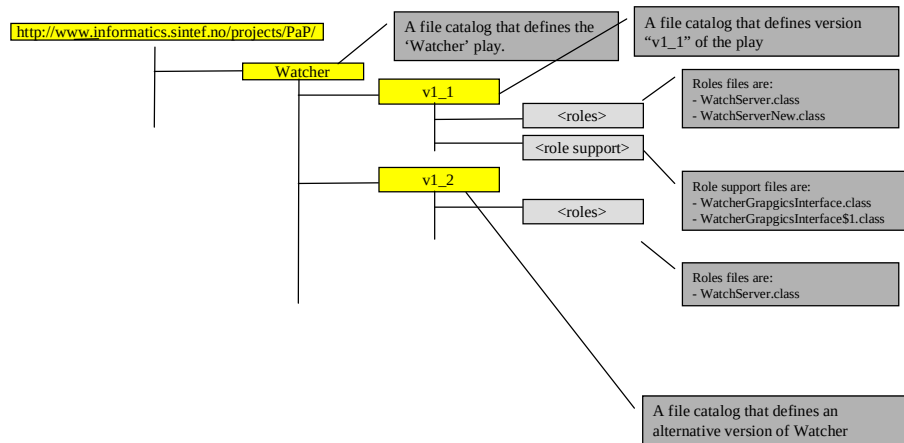
At initial start-up Watcher will update the displayed PaP information status continuously when it occur. The update of information is stopped when ‘WatchServer’ receives the signal ‘SubscribeCancel’, and continued when receiving the signal ‘SubscribeRequestType2’. Now an update of the information will occur only each 30 seconds.

The Watcher application also has functionality that demonstrates the ‘ActorChangeBehaviour’ PaP functionality. This will occur when ‘WatchServer’ receives the ‘UpgradeBehaviour’ signal.

The Watcher application also has functionality that demonstrates use of ‘PlayChangesPlugIn’ functionality. This is obtained by doing plug-in of version ‘v1\_2’ of the play in addition to initial version ‘v1\_1’ and start a new Watcher instance. The difference is viewed by the application displaying the version indication in the tool-bar heading.

## “Watcher”, a PaP Support Activity Monitor

### Description of the Implementation - Files catalogues and files



## “Watcher”, a PaP Support Activity Monitor

User's guide on how to use

- **Install PaP Support software and the Watcher application**  
Both must have been installed according to generally defined directives as defined in the PaP Support documentation.
- **Using the Watcher application**
  - **Starting the Watcher application**  
By starting an actor playing the 'WatchServer' role, either from a Non PaP application, or from another PaP application.
  - **Demonstrate 'PlayChangesPlugIn'**  
Perform a PlayChangesPlugIn of version v1\_1 and start a new Watcher instance.
  - **Demonstrate 'ActorChangeBehaviour'**  
Send signal 'UpgradeBehaviour' to a Watcher instance.
  - **Change registration info. update from continuous to every 30. Sec.**  
Send signals 'SubscribeCancel' and then 'SubscribeRequestType2' to a Watcher instance.

## Start of “TestPaP” application documentation

16.10.00

Plug and Play (PaP) for Network and Tele-Services

56

## “TestPaP”, a Tele based learning centre

### Documentation Survey

"TestPaP", a tool for automatic testing of PaP Support

Description of the functionality

Description of the implementation

User's guide on how to use

**Note:**

•For detailed documentation, consult 'javadoc' generated information from source code.

## **“TestPaP”, a tool for automatic testing of PaP Support**

### **Description of the functionality**

- **Automatic testing of PaP Support functionality**

More or less any of the defined PaP Support functions are tested repeatedly under different conditions. Different conditions include e.g. sequence, time intervals, context.

- **Reporting of what tests are performed, the number of times repeated, and the number of failures detected**

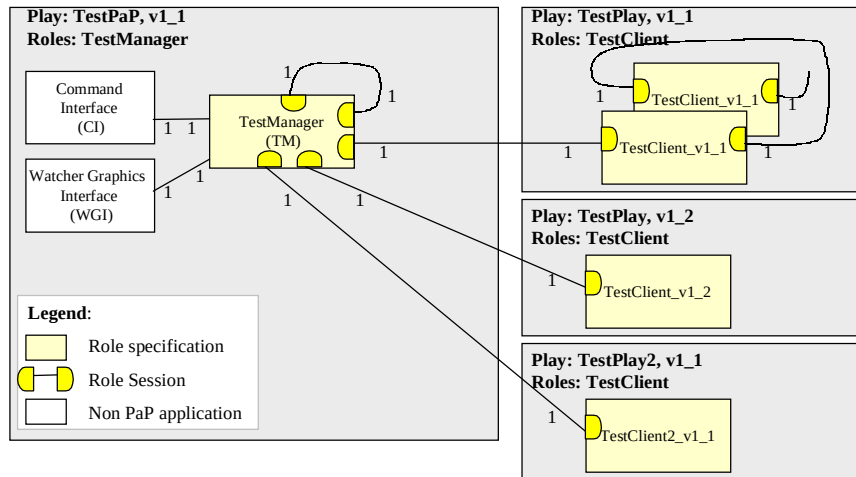
For each test performed the number of repetitions and number of failures is reported. In addition a test execution log indicating the sequence of execution and the time when executed is shown.

- **Operator reconfiguration of how testing shall be done**

An operator may change test set-up configuration through an operator interface. Time intervals between test may be set, and also whether or not to use multiple nodes.

## “TestPaP”, a tool for automatic testing of PaP Support

### Description of the Implementation - Modelling



16.10.00

Plug and Play (PaP) for Network and Tele-Services

59

The ‘TestPaP’ application consists both of multiple plays and of multiple versions for the same play. The reason for that is obviously because it is necessary to be able to test both multiple plays and versions of a play.

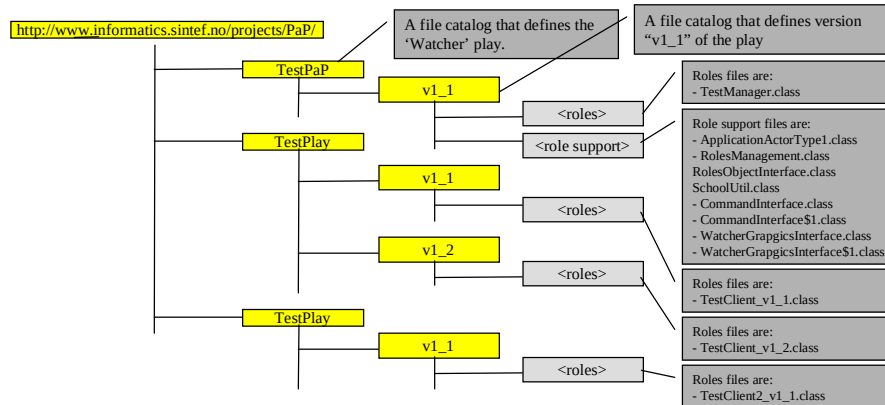
It is the ‘TestManager’ (TM) role in the ‘TestPaP’ play that controls the whole test process. TestManager executes most of the defined PaP Support functions - under different conditions and with different arguments, and test the results of the operations. Detected faults are reported. TestManager testing includes the ActorPlugIn and ActorPlugOut of actors with valid roles as defined in the play definitions shown in the figure, and also for invalid or undefined roles and plays.

Actor ‘TestClient\_v1\_1’ in play ‘TestPlay’ may also perform ActorPlugIn of actors playing the same role as own actor. This functionality may result in the situation that at some time during testing a ‘linked list’ of TestClient\_v1\_1 type of actors may appear in the testing PaP domain.

Note that TestManager also do testing of ActorPlugIn/ActorPlugOut to itself.

## “TestPaP”, a tool for automatic testing of PaP Support

### Description of the Implementation - Files catalogues and files



16.10.00

Plug and Play (PaP) for Network and Tele-Services

60

The structuring of the TestPaP application software is according to what is defined for PaP applications - at top level a catalogue defining the plays names ('TestPaP', 'testPlay' and 'TestPlay2'), and at next level catalogues that defines the versions ('v1\_1' and eventually 'v1\_2'). Note however, the three plays that together constitutes the TestPaP application, could have been located at completely different web-locations, because when plugged-in at Director actor, they would have been 'logically' collected.

Only one role, represented by a java class file is defined for each version of the three plays. In addition, the TestPaP play makes use of a set of role support files.

## “TestPaP”, a tool for automatic testing of PaP Support

User’s guide on how to use

- **Install PaP Support software and the TestPaP application**  
Both must have been installed according to generally defined directives as defined in the PaP Support documentation.
- **Using the TestPaP application**
  - **Starting the TestPaP application**  
By starting an actor playing the ‘TestManager’ role, either from a Non PaP application, or from another PaP application, a user dialogue window will appear, and further actions are taken from that window.
  - **Setting Configuration parameters**  
Includes: Selection of test set, test level, test intervals, max. no. of errors before stop testing, one or multiple nodes included in the testing, and what types of trace to show.
  - **Command actions**  
Includes: Start, Stop (with later possibility to continue), Restart from beginning, and Terminate.

16.10.00

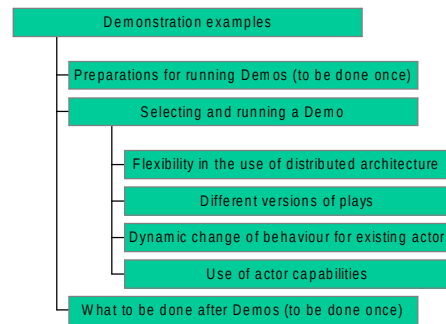
Plug and Play (PaP) for Network and Tele-Services

61

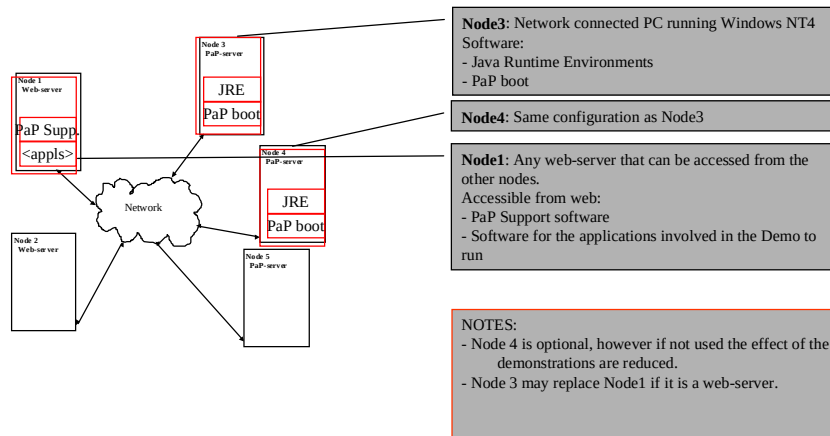
## Start of “Demonstration examples” documentation

## Demonstration examples

### Documentation Survey, Overview



## Preparations for any demo (done once)



16.10.00

Plug and Play (PaP) for Network and Tele-Services

64

Definitions used:

<PaPSupportRoot>: The root location at web-server for PaP Support software.

<PaPApplRoot>: The root location at web-server for PaP a application.

<PaPboot> : The location where the PaP server PaP boot software is located.

### Preparation of Node 1:

- 1) Unpack the file "PaPSupport.zip" at the selected "<PaPRoot>" location.
- 2) Unpack the file "PaPAApplications.zip" at the selected "<PaPApplRoot>" location.

### Preparation of Node 3 and Node 4:

- 1) If Java Runtime Environments missing (e.g directory "c:\Program Files\JavaSoft\Jre" is missing), installation can be done from  
" <http://www.sun.com/>".
- 2) If not done - unpack the file "PaPboot.zip" at the selected "<PaPboot>" location.
- 3) If not done - at Node 4 (if applied) modify the "pap.cnf" file according to:  
homeinterface = Actor://Node3/pas1/PaP.Director1
- 4) Start PaP node execution support from a DOS window by:

```
cd <PaPBoot>
java PaP.boot PaP.PNES
```

A separate window indicating the PNES instance running should appear. The window is divided into two parts: a) The operator command section (OCS), and b) the trace output window. OCS will extensively be used when running the tests. Commands in OCS may either be written, or selected by using arrow up/down keys. Commands are executed when pushing the "Return" key.

## Demonstration example

### Prep1: Preparations for PaP demonstrations

- Determine web-server nodes and PaP server nodes to use
- Installation/preparation of web-server nodes
- Installation/preparation of PaP server nodes
- Start a PaP server at a node
- Run a demo
- Stop a PaP server

*Detailed descriptions is specified in a separate document*

## Demonstration example

### Demo1: Flexibility in use of the distributed architecture

- To demonstrate: Flexibility in where applications executes, where started from, and distribution of parts of an application.
- Used applications: “Watcher” and “School”.  
The Watcher application is included for the purpose to observe where different parts of the School application executes.
- Demonstrate the following aspects of flexibility
  - To control where applications executes
  - Applications on different nodes attached to the same Director instance
  - Configuration of School application determines the distribution

***Detailed descriptions is specified in a separate document***

## Demonstration example

### Demo2: Different versions of the same play

- To demonstrate: Plug in/out different play versions, and plug in/out actors performing roles for that play.
- Used application: Watcher has defined two versions of the same play - versions “v1\_1” and “v1\_2”.

*Detailed descriptions is specified in a separate document*

## Demonstration example

### Demo 3: Dynamic change of behaviour for existing actors

- To demonstrate: Change of behaviour for an performing actor by redefining its role behaviour
- Used application: The application “Watcher” has implemented an alternative role (‘WatchServerNew’) that can be used for ‘ActorChangeBehaviour’ from role ‘WatchServer’.

*Detailed descriptions is specified in a separate document*

## Demonstration example

### Demo 4: Use of actor capabilities

- To demonstrate: actors use of capabilities
- Used application: The “School” application make use of capabilities when selecting common server actors for the clients. This can be verified by using debug and trace facilities in the PaP support.
- *Detailed descriptions is specified in a separate document*