TAPAS platform execution framework

Version 1.0

Patcharee Thongtra, Finn Arve Aagesen

Telematics Architecture for Play-based Adaptable System (TAPAS)
Research Project

Telematics, NTNU

Table of Contents

1. Introduction

This document describes the use of the TAPAS platform for the deployment and execution of service systems.

2. TAPAS platform execution framework components

The TAPAS platform execution framework is illustrated in Figure 1. Each component is explained below.

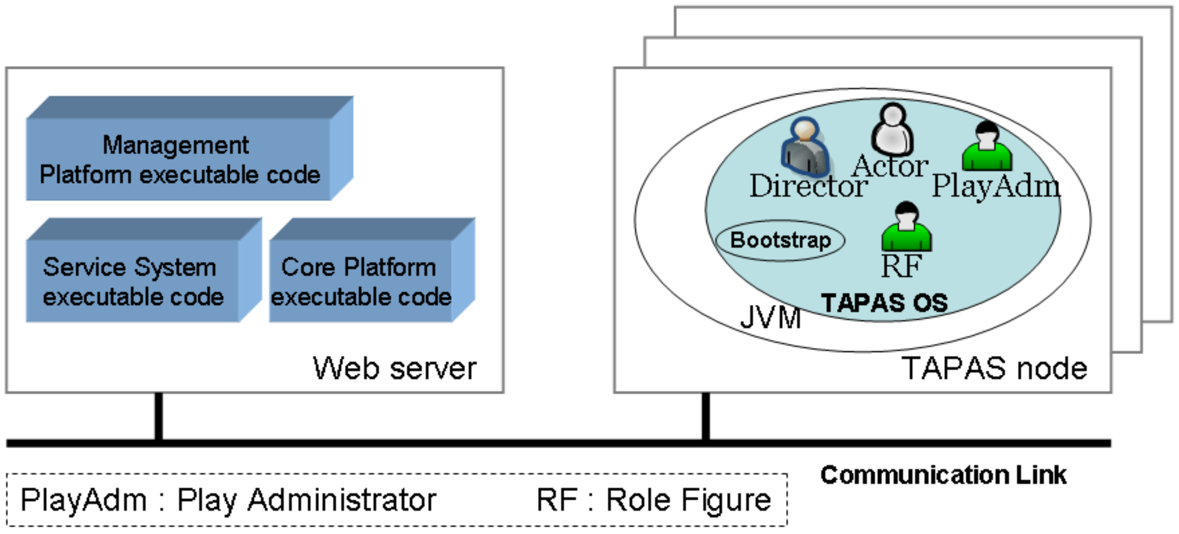
Management Platform executable code
Service System executable code
Core Platform executable code
Web server
Director
Actor
PlayAdm
Bootstrap
RF
TAPAS OS
JVM
TAPAS node
Communication Link
PlayAdm : Play Administrator
RF : Role Figure

Figure 1: TAPAS platform execution framework.

TAPAS operating system is an execution environment created in the TAPAS node. Within this execution environment, communication channels between the TAPAS nodes, input message queues, output message queues, a *Director* are created.

Bootstrap is a Java thread running under JVM environment. The Bootstrap must be installed and started manually. By *Java classloader* feature, the Bootstrap can download executable Java code from remote nodes to execute in a local node. We set up a *Web server* as the remote node. So that, after the Bootstrap is started, it will automatically download and execute the *core platform executable code*; in which that execution creates *TAPAS operating system* in the local node. We call such local node that has the TAPAS operating system as **TAPAS Node**.

Web server is set up as the remote node as mentioned above for the *code deployment*, which related codes are the *core platform executable code*, the *management platform executable code* and the *service system executable code*. *TAPAS engineer* deploys the core platform- and the management platform executable code to the Web server. The *service engineer* deploys the service systems executable *codes*, before the service systems and their service components can be deployed and instantiated in TAPAS node.

Core platform executable code is an exported Jar file of the TAPAS core platform Java project. The TAPAS core platform supports the functionality in TAPAS computing architecture.

Management platform executable code is an exported Jar file of the TAPAS management platform Java project. The TAPAS management platform supports the functionality defined in TAPAS service functionality architecture.

Service system executable code represents an exported Jar file of *any* service systems. As a service system consists of inter-operating service components, so its executable code consists of java classes representing the service components' manuscripts. Also, the service system and the service component are represented respectively by **Play** and **Role Figure (RF)**. As mentioned that the service engineer deploys the service systems executable *codes*, he/she can do this code deployment by using a procedure from the TAPAS core platform.

Actor is a java thread which is created, assigned a role and terminated by the *Director*. When an actor is assigned a role, it will download and execute the role's manuscript.

Director manages other *actors* in a TAPAS node. The management consists of creating new actors, (re-)assigning roles to the actors, releasing the actors from roles as well as terminating them. A Director is automatically instantiated in a TAPAS node by the TAPAS operating system.

Play Administrator (PlayAdm) handles the (un-)registration of plays, maintains the location of executable codes of plays on the Web server, and informs actors who constitutes role figures when their role manuscripts are updated.

3. TAPAS platform execution framework and service system instantiation

Figure 2 and 3 present two sequence diagrams, which illustrate the instantiation of TAPAS platform execution framework and service system, respectively

TAPAS platform execution framework instantiation

1. After the Bootstrap starts, it downloads and executes the core platform executable code. The TAPAS OS is created.

2. The Bootstrap calls init() function of the TAPAS OS. By the init() function, the TAPAS OS creates MultiCastMessageServer, UnicastMessageServer and Director.

3. The TAPAS OS calls init() function of the MultiCastMessageServer. By the init() function, the MultiCastMessageServer starts a communication channel, an input message queue and an output message queue for multicast messages.

4. The TAPAS OS calls init() function of the UniCastMessageServer. By the init() function, the UniCastMessageServer starts a communication channel, an input message queue and an output message queue for unicast messages.

5. The TAPAS OS calls init() function of the Director. By the init() function, the Director sets up some internal variables needed for itself.

6. The Director checks whether a password for instantiate role figure in the TAPAS management platform is provided in a text file or not.

6.1. If there is a password provided and the password is correct, then the Director creates new actors, and assigns them the roles by calling init(role) function. However, only role PlayAdm is illustrated in the diagram.

6.2. An actor downloads the assigned role's manuscript.

6.3. The init(role) function returns boolean true, which it means the download was successful and the actor is ready to constitute the role figure.

6.4. The Director calls startAction() function of the actor. The actor executes the manuscript.

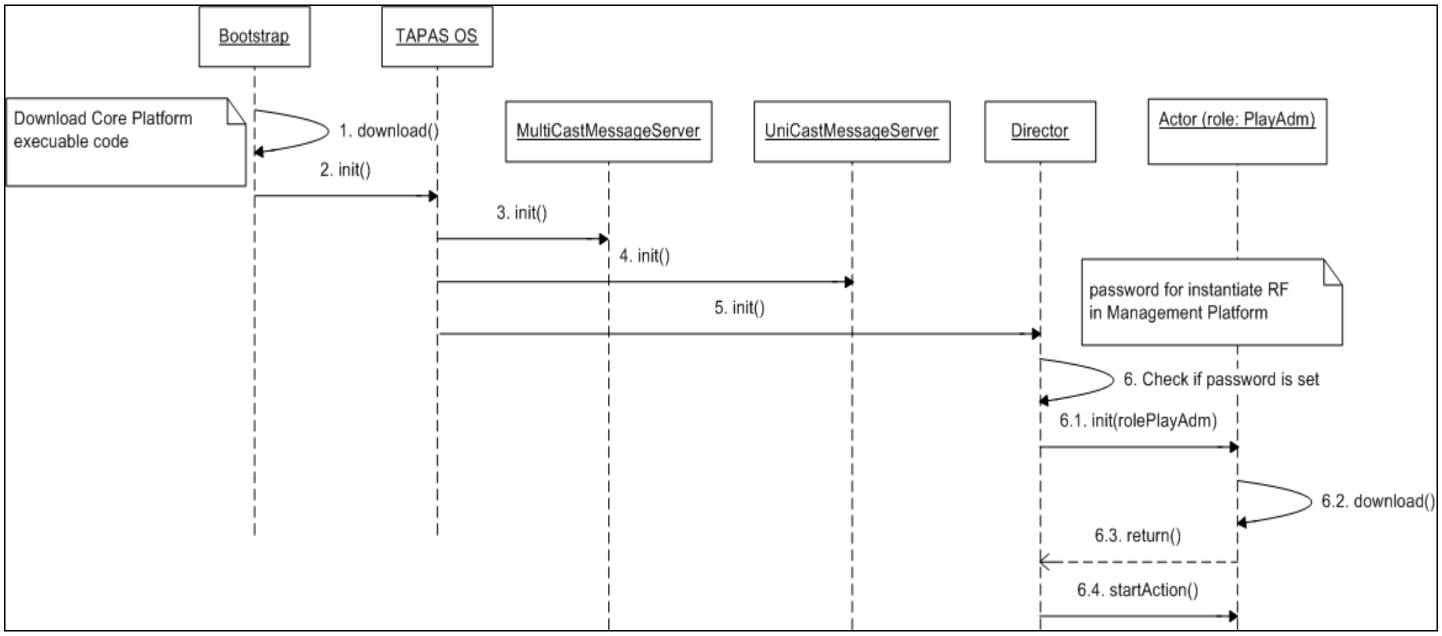

Figure 2: TAPAS platform execution framework instantiation.

TAPAS service system instantiation (This is a simple example case that a user uploads the service system execution code, and instantiates a service component; called *service component example*, in a user-selected node by using the core platform procedures.)

1. User calls plugInPlay() function of the Procedure class.

2. The Procedure sends a CheckPlayExistenceRequest message to the PlayAdm.

3. The MultiCastMessageServer, in the node which the PlayAdm is executing, gets the CheckPlayExistenceRequest message and forwards it to the PlayAdm.

4. The PlayAdm checks if a play, which user wants to plug it in, has been plugged in already or not.

5. The PlayAdm finds that the play has not been plugged in yet. Then the PlayHandler sends a PlayNonExistenceResponse message to the Procedure through the UniCastMessageServer.

6. The Procedure gets the PlayNonExistenceResponse message.

7. The Procedure uploads the service system execution code to the Web server.

8. The Procedure sends a RegisterPlayRequest message to the PlayHandler.

9. The MultiCastMessageServer, in the node which the PlayAdm is executing, gets the RegisterPlayRequest message and forwards it to the PlayAdm.

10. The plugInPlay() function finishes and returns the URL location of the service system execution code to the user.

11. The user calls plugInRole() function of the Procedure class.

12. The Procedure sends a StartServiceComponentOnThisNodeRequest message to a Director that is executing in a user-selected node.

13. The UniCastMessageServer in the user-selected node gets the StartServiceComponentOnThis-NodeRequest message and forwards it to the Director.

14. The Director creates a new actor or finds an available actor, and assigns the actor a role by calling init(role) function. In the diagram, the role Teacher is illustrated as an example.

15. An actor downloads the assigned role's manuscript.

16. The init(role) function returns boolean true, which it means the download was successful and the actor is ready to constitute the role figure.

17. The Director calls startAction() function of the actor. The actor executes the manuscript.

18. The Director sends a StartServiceComponentOnThisNodeSuccess message to the Procedure through the UniCastMessageServer.

19. The Procedure gets the StartServiceComponentOnThisNodeSuccess message.

20. The plugInRole() function finishes and returns the actor identification who is constituting the role *service component example* to the user.

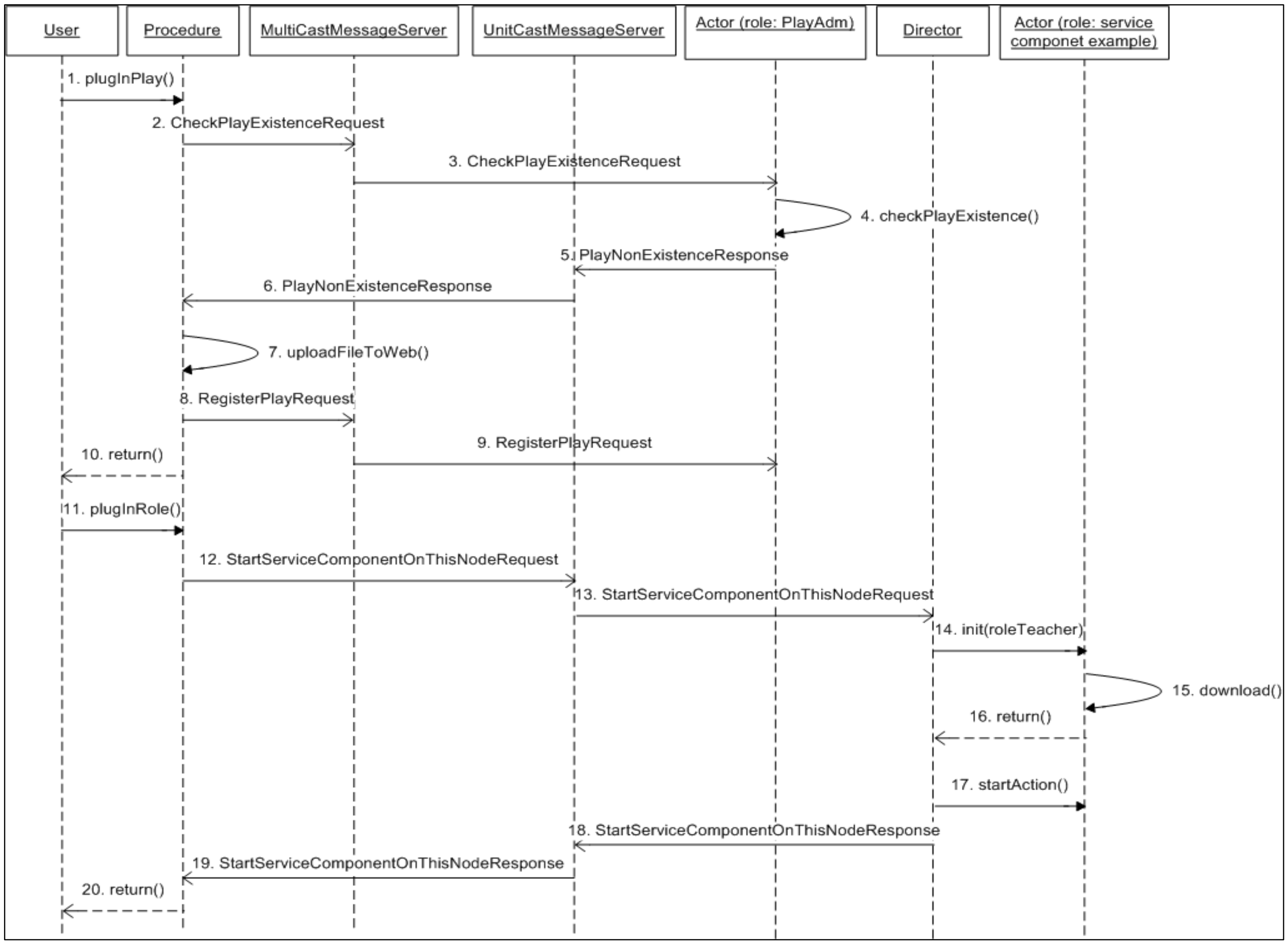
User
Procedure
MultiCastMessageServer
UnitCastMessageServer
Actor (role: PlayAdm)
Director
Actor (role: service componet example)
1. plugInPlay()
2. CheckPlayExistenceRequest
3. CheckPlayExistenceRequest
4. checkPlayExistence()
5. PlayNonExistenceResponse
6. PlayNonExistenceResponse
7. uploadFileToWeb()
8. RegisterPlayRequest
9. RegisterPlayRequest
10. return()
11. plugInRole()
12. StartServiceComponentOnThisNodeRequest
13. StartServiceComponentOnThisNodeRequest
14. init(roleTeacher)
15. download()
16. return()
17. startAction()
18. StartServiceComponentOnThisNodeResponse
19. StartServiceComponentOnThisNodeResponse
20. return()

Figure 3: TAPAS service systems instantiation.