TAPAS platform messages and procedures

Version 1.0

Patcharee Thongtra, Finn Arve Aagesen

Telematics Architecture for Play-based Adaptable System (TAPAS)
Department of Telematics, NTNU

Table of Contents

1. Introduction

TAPAS platform consists of core platform and management platform. Core platform supports the functionality of TAPAS computing architecture. Management platform supports the functionality of TAPAS service functionality architecture. This document describes the messages and procedures of TAPAS platform. Messages and procedures are intended to be used by the service engineers. The procedures are meant to be more high level and user friendly than the messages. The usage of the messages needs more insight to the inner details of architecture. All procedures can be replaced by messages.

Section 2 gives and overview of the messages. Section 3 and 4 describes in more detail the messages of the core and management platform, respectively. Section 4 describes the core platform procedures.

The reader is recommended to read this document together with *the TAPAS platform execution framework* and *Service systems implementation guideline*. These documents can be downloaded from the TAPAS Platform page at TAPAS web site. Two basic roles in the Tapas execution framework to be mentioned here are:

- the Director and
- the Play Administrator.

The 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. The 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 when the TAPAS execution framework is instantiated.

2. TAPAS platform messages

2.1 General

The messages are classified into six groups:

- General request
- Trouble report
- Mobility request and report
- Diagnosis
- Configuration
- Capability Monitoring

A message contains *message ID*, *sender ID*, *receiver(s) ID*, *system time when composing the message* and *message data*. The message ID is a unique identifier for a message. The sender ID and receiver(s) ID are an actor_id and a set of actor_id respectively, where

actor_id = (node_id, port_number, actor_number).

The node_id is defined by the IP address of node, port_number is a port number that is used for the Java socket, and actor_number is a local counter. *Message data* consist of *message name* and *message parameters*. Table 1 gives an overview of message names and parameters related to the various message groups.

Table 1. Messages names and parameters

Message Names	Message Parameters	Platform
General request group		
CheckPlayExistenceRequest	play_name	Core
PlayExistenceResponse	play_name, play_version	Core
PlayNonExistenceResponse	play_name	Core
RegisterPlayRequest	play_name, play_version, specification_location	Core
UnRegisterPlayRequest	play_name, play_version	Core
CheckPlayInUseRequest	play_name	Core
PlayInUseResponse	play_name	Core
PlayNotInUseResponse	play_name	Core
GetPlayLocationRequest	play_name, play_version	Core
PlayLocationResponse	play_name, play_version, specification_location	Core
ChangePlayReport	play_name, play_version	Core
StartServiceComponentOnThisNodeRequest	service_component_id, specification_location	Core
StartServiceComponentOnThisNodeSuccess	service_component_id, actor_id	Core
StartServiceComponentOnThisNodeFailure	service_component_id	Core
PlugOutRoleOnActorRequest	actor_id	Core
PlugOutRoleOnActorSuccess	actor_id	Core
PlugOutRoleOnActorFailure	actor_id	Core
ChangeRoleOnActorRequest	actor_id, new_ service_component_id, new_specification_location	Core
ChangeRoleOnActorSuccess	actor_id, new_ service_component_id	Core
ChangeRoleOnActorFailure	actor_id, new_ service_component_id	Core
ActorTermination	actor_id	Core
PlanServiceComponentRequest	service_component_id, specification_location, capability_req_location	Man
PlanServiceComponentResponse	role_figure_id	Man
StopServiceComponentRequest	service_component_id, role_figure_id	Man
ServiceAdaptationRequest	service_component_id	Man
ServiceAdaptationResponse	new_role_figure_id	Man
RoleFigureDialogue	dialogue_id, *{parameter_on_purpose_of_dialogue}*	Core
Trouble report group		
RoleFigureUnreachable	role_figure_id	Man
InsufficientCapability	actor_id	Man
QoSDegradation	service_component_id	Man
NodeNoResponse	node_id	Man
Mobility request and report group		
RoleFigureMoveRequest	role_figure_id, new_node_id	Man
InformDialogue	dialogue_object	Man
SetDialogue	{dialogue_object}	Man
GetEFSMData	-	Man
EFSMData	curr_state, {curr_var}, {curr_msg}	Man
SetEFSMData	curr_state, {curr_var}, {curr_msg}	Man
Diagnosis group		
RoleFigureFailed	role_figure_id	Man
RoleFigureSuspended	role_figure_id	Man
RoleFigureContinued	old_role_figure_id, new_role_figure_id	Man
HeartBeat	-	Man
HeartBeatLost	role_figure_id	Man
ConfirmAlive	-	Man

Alive	-	Man
Configuration group		
GetNodeCapabilityRequest	node_id	Man
NodeCapability	{cp_mon_object}	Man
GetCapableNode	{cp_mon_object}	Man
CapableNode	node_id	Man
StartServiceComponentRequest	service_component_id, specification_location, node_id	Man
StartServiceComponentResult	role_figure_id	Man
CapabilityMonitoring group		
MonitorCapabilityRequest	node_mon_object, monitoring_interval	Man
StopMonitorCapabilityRequest	node_id	Man
UpdateMonitoredCapability	node_mon_object	Man
InformMonitoringDomain	imm_id, domain_mon_object	Man
SetMonitoringDomain	domain_mon_object	Man
SetMonitoringDomainsUnderMainManager	mmm_mon_object	Man

2.2 TAPAS core platform messages descriptions

CheckPlayExistenceRequest (play_name)

Usage: The message is sent to check whether there a specific play name is registered.
Senders: The procedure plugInPlay, the procedure changePlay, Role figures, Users
Receivers: PlayAdm
Parameters: *play_name*: name of a play

PlayExistenceResponse (play_name, play_version)

Usage: The message is sent as a reply to the message CheckPlayExistenceRequest.
Senders: PlayAdm
Receivers: The procedure plugInPlay, the procedure changePlay, Role figures, Users.
Parameters: *play_name*: name of a registered play, *play_version*: current version of a registered play

PlayNonExistenceResponse (play_name)

Usage: The message is sent is sent as a reply to the message CheckPlayExistenceRequest.
Senders: PlayAdm.
Receivers: The procedure plugInPlay, the procedure changePlay, Role figures, Users
Parameters: *play_name*: name of a play

RegisterPlayRequest (play_name, play_version, specification_location)

Usage: The message is sent to register a play.
Senders: The procedure plugInPlay, Role figures, Users
Receivers: PlayAdm.
Parameters: *play_name*: name of a play, *play_version*: version number of a play, *specification_location*: location of a JAR file on Web server

UnRegisterPlayRequest (play_name, play_version)

Usage: The message is sent to unregister a play.
Senders: The procedure plugOutPlay, Role figures, Users

Receivers: PlayAdm.
Parameters: *play_name*: name of a play, *play_version*: version number of a play

CheckPlayInUseRequest (play_name)

Usage: The message is sent to check if a play is in use.
Senders: The procedure plugOutPlay, Role figures, Users
Receivers: PlayAdm.
Parameters: *play_name*: name of a play

PlayInUseResponse (play_name)

Usage: The message is sent as a reply to CheckPlayInUseRequest .
Senders: PlayAdm.
Receivers: The procedure plugOutPlay, Role figures, Users
Parameters: *play_name*: name of a play

PlayNotInUseResponse (play_name}

Usage: The message is sent as a reply to CheckPlayInUseRequest.
Senders: PlayAdm
Receivers: the procedure plugOutPlay, Role figures, Users
Parameters: *play_name*: name of a play

GetPlayLocationRequest (play_name, play_version)

Usage: The message is sent to get a location of the latest version of a play.
Senders: the procedure plugOutPlay, Role figures, Users
Receivers: PlayAdm.
Parameters: *play_name*: name of a play, *play_version*: version number of a play

PlayLocationResponse (play_name, play_version, specification_location)

Usage: The message is sent as a reply GetPlayLocationRequest.
Senders: PlayAdm
Receivers: The procedure plugOutPlay, Role figures, Users
Parameters: *play_name*: name of a play, *play_version*: version number of a play, *specification_location*: location of the latest version of a play on Web server

ChangePlayReport (play_name, play_version)

Usage: The message is sent to inform that a registered play is changed and re-uploaded.
Senders: The procedure changePlay, Role figures, Users
Receivers: PlayAdm.
Parameters: *play_name*: name of a play, *play_version*: version number of a play

StartServiceComponentOnThisNodeRequest (service_component_id, specification_location)

Usage: The message is sent to start an available actor, on an already selected node, to play a given role.
Senders: The procedure plugInRole, Role figures, Users
Receivers: Director
Parameters: *service_component_id*: a service component id, which is constituted by (role_id, play_name, play_version), where role_id is name of a role, play_name is name of a play, and

play_version is version number of a play, *specification_location*: location of the role manuscript on Web server.

StartServiceComponentOnThisNodeSuccess (service_component_id, actor_id)

Usage: The message is sent as a reply to the message PlugInRoleOnActorRequest.
Senders: Director
Receivers: The procedure plugInRole, Role figures, Users
Parameters: *service_component_id*: a service component id, *actor_id*: an actor id

StartServiceComponentOnThisNodeFailure (service_component_id)

Usage: The message is sent as a reply to the message PlugInRoleOnActorRequest.
Senders: Director
Receivers: The procedure plugInRole, Role figures, Users
Parameters: *service_component_id*: a service component id

PlugOutRoleOnActorRequest (actor_id)

Usage: The message is sent to stop an actor playing a role.
Senders: The procedure plugOutRole, Role figures, Users
Receivers: Director
Parameters: *actor_id*: an actor id

PlugOutRoleOnActorSuccess (actor_id)

Usage: The message is sent as a reply to the message PlugOutRoleOnActorRequest
Senders: Director
Receivers: The procedure plugOutRole, Role figures, Users
Parameters: *actor_id*: an actor id

PlugOutRoleOnActorFailure (actor_id)

Usage: The message is sent as a reply to the message PlugOutRoleOnActorRequest
Senders: Director
Receivers: The procedure plugOutRole, Role figures, Users
Parameters: *actor_id*: an actor id

ChangeRoleOnActorRequest (actor_id, new_service_component_id, new_specification_location)

Usage: The message is sent to make an Actor change to a new role.
Senders: The procedure changeRole, Role figures, Users
Receivers: Director
Parameters: *actor_id*: an actor id, *new_service_component_id*: a new service component id, *new_specification_location*: location of the new role manuscript on Web server.

ChangeRoleOnActorSuccess (actor_id, new_service_component_id)

Usage: The message is sent a reply to the message ChangeRoleOnActorRequest.
Senders: Director
Receivers: The procedure changeRole, Role figures, Users
Parameters: *actor_id*: an actor id, *new_service_component_id*: a new service component id

ChangeRoleOnActorFailure (actor_id, new_service_component_id)

Usage: The message is sent a reply to the message ChangeRoleOnActorRequest.
Senders: Director
Receivers: The procedure changeRole, Role figures, Users
Parameters: *actor_id*: an actor id, *new_service_component_id*: a new service component id

ActorTermination (actor_id)

Usage: The message is sent to terminate an actor.
Senders: Role Figures, Users
Receivers: Director
Parameters: *actor_id*: an actor id

RoleFigureDialogue (dialogue_id, {parameter_on_purpose_of_dialogue})

Usage: The message is sent to communicate with other role figures.
Senders: Role Figures
Receivers: Role Figures
Parameters: *dialogue_id*: a dialogue id which represents the dialogue purpose, *{parameter_on_purpose_of_dialogue}*: list of parameters, which vary and depend on the dialogue purpose.

2.3 TAPAS management platform messages descriptions

PlanServiceComponentRequest (service_component_id, specification_location, capability_req_location)

Usage: The message is sent to initiate a *service component* on some node.
Senders: Role figures, Users
Receivers: Capability Configuration Manager
Parameters: *service_component_id*: a service component id, *specification_location*: location of the role manuscript on Web server, *capability_req_location*: location of the capability requirement of the role on Web server

PlanServiceComponentResponse (role_figure_id)

Usage: The message is sent as a response to the message PlanServiceComponentRequest.
Senders: Capability Configuration Manager
Receivers: Role figures, Users
Parameters: *role_figure_id*: a role figure id, which is constituted by (service_component_id, actor_id), where service_component_id is a service component id and actor_id is an actor id of whom constitutes the role figure.

StopServiceComponentRequest (service_component_id, role_figure_id)

This message is not implemented yet. The purpose is to terminate a service component.

ServiceAdaptationRequest (service_component_id, role_figure_id)

This message is not implemented yet. The purpose is to adapt a service component.

ServiceAdaptationResponse (new_role_figure_id)

This message is not implemented yet. The purpose is for the response to the message ServiceAdaptationRequest.

RoleFigureUnreachable (role_figure_id)

Usage: The message is sent to report that a role figure is unreachable.
Senders: Role figures, Users
Receivers: Fault Diagnosis Manager
Parameters: *role_figure_id*: a role figure id

InsufficientCapability (actor_id)

This message is not implemented yet. The purpose is to report that an actor does not have inherent capabilities to constitute a role figure.

QoSDegradation (service_component_id)

This message is not implemented yet. The purpose is to degrade the QoS performance of a service component.

NodeNoResponse (node_id)

Usage: The message is sent to report that a registered node is unreachable.
Senders: Intermediate Monitoring Manager
Receivers: Capability and Service Administration Manager
Parameters: *node_id*: IP address of a node

RoleFigureMoveRequest (role_figure_id, new_node_id)

Usage: The message is sent to request the move a role figure to a new Node and then to start the role figure. Then, the EFSM data of the role figure will be maintained during the movement.
Senders: Users, Role Figure
Receivers: Mobility Manager
Parameters: *role_figure_id*: a role figure id, *new_node_id*: IP address of a new node.

InformDialogue (dialogue_object)

Usage: The message is sent to inform about current dialogue between a role figure and its cooperating role figure. The message is sent out in case a cooperating role figure becomes dead.
Senders: Role Figure
Receivers: Mobility Manager
Parameters: *dialogue_object*: a dialogue object, which is constituted by (my_role_figure_id, coop_role_figure_id), where my_role_figure_id and coop_role_figure_id are role figure ids.

SetDialogue ({dialogue_object})

Usage: The message is sent to provide re-instantiated dialogue instances. The message is sent out after the new role figure has been instantiated.
Senders: Mobility Manager
Receivers: Role Figure
Parameters: *{dialogue_object}*: list of dialogue objects

GetEFSMData ()

Usage: The message is sent to request EFSM data of a role figure.
Senders: Mobility Manager
Receivers: Role Figure
Parameters: None

EFSMData (curr_state, {curr_var}, {curr_msg})

Usage: The message is sent as a response to the message GetEFSMData.
Receivers: Mobility Manager
Parameters: *curr_state*: current state, *{curr_var}*: list of variables, which a variable is constituted by (variable name, variable value), *{curr_msg}*: list of input messages, which a message is constituted by (message ID, sender ID, receiver(s) ID, system time when composing the message, message data)

SetEFSMData (curr_state, {curr_var}, {curr_msg})

Usage: The message is sent to provide a re-instantiated role figures with its previous EFSM data. The message is sent out after the role figure has been instantiated.
Senders: Mobility Manager
Receivers: Role Figure
Parameters: *curr_state*: current state: *{curr_var}*: list of variables, *{curr_msg}*: list of input messages

RoleFigureFailed (role_figure_id)

Usage: The message is sent to broadcast the death of a role figure.
Senders: Fault Diagnosis Manager
Receivers: Role Figures
Parameters: *role_figure_id*: a role figure id

RoleFigureSuspended (role_figure_id)

Usage: The message is to broadcast about the suspension of a role figure. The message is sent out when a role figure is being moved
Senders: Mobility Manager
Receivers: Role Figures
Parameters: *role_figure_id:* a role figure id

RoleFigureContinued (old_role_figure_id, new_role_figure_id)

Usage: The message is broadcasted when a role figure has been re-instantiated after the death or the suspension.
Senders: Mobility Manager, Fault Diagnosis Manager
Receivers: Role Figures
Parameters: *old_role_figure_id*: a dead role figure id, *new_role_figure_id*: a new instantiated role figure id

Heartbeat ()

Usage: The message is sent to inform that a role figure itself is alive.
Senders: Role Figure
Receivers: Role Figures
Parameters: None

HeartbeatLost (role_figure_id)

Usage: The message is sent to inform Fault Diagnosis Manager that the Heartbeat message from a cooperating role figure is lost.
Senders: Role Figure
Receivers: Fault Diagnosis Manager
Parameters: *role_figure_id:* a cooperating role figure id.

ConfirmAlive ()

Usage: The message is sent to check whether a role figure is still alive or not.
Senders: Fault Diagnosis Manager
Receivers: Role Figure
Parameters: None

Alive ()

Usage: The message is sent to confirm that a role figure itself is alive.
Senders: Role Figure
Receivers: Fault Diagnosis Manager
Parameters: None

GetNodeCapabilityRequest (node_id)

Usage: The message is sent to get capabilities of registered nodes.
Senders: Role Figures, Users
Receivers: Capability and Service Administration Manager
Parameters: *node_id*: IP address of a node

NodeCapability ({cp_mon_object})

Usage: The message is sent to provide information about capabilities of registered nodes.
Senders: Capability and Service Administration Manager
Receivers: Role Figures, Users
Parameters: *{cp_mon_object}*: list of monitored capabilities, which a monitored capability is constituted by (cy_type, cp_param_list), where cp_param_list = {(cp_param, cp_param_value)}, and cp_type and cp_param are SNMP MIB OIDs.

GetCapableNode ({cp_mon_object})

Usage: The message is sent to get a registered node whose has required capabilities.
Senders: Role Figures, Users
Receivers: Capability Configuration Manager
Parameters: *{cp_mon_object}*: list of monitored capabilities

CapableNode (node_id)

Usage: The message is sent to provide IP address of a registered node whose has required capabilities.
Senders: Capability Configuration Manager
Receivers: Role Figures, Users
Parameters: *node_id*: IP address of a node

StartServiceComponentRequest (service_component_id, specification_location, node_id)

Usage: The message is sent to request Deployment and Instantiation Manager to deploy and instantiate a service component in a selected node.
Senders: Capability Configuration Manager
Receivers: Deployment and Instantiation Manager
Parameters: *service_component_id*: a service component id, *specification_location*: location of the role manuscript on Web server, *node_id*: IP address of a selected node

StartServiceComponentResult (role_figure_id)

Usage: The message is sent as a response to the message StartServiceComponentRequest.
Senders: Deployment and Instantiation Manager
Receivers: Capability Configuration Manager
Parameters: *role_figure_id*: a role figure id

MonitoringCapabilityRequest (node_mon_object, monitoring_interval)

Usage: The message is sent to request the capability monitoring of a registered node.
Senders: Capability and Service Administration Manager, Main Monitoring Manager, Users
Receivers: Main Monitoring Manager, Intermediate Monitoring Manager
Parameters: *node_mon_object*: a monitored node object, which is constituted by (node_id, {cp_mon_object}), *monitoring_interval*: monitoring time interval.

StopMonitorCapabilityRequest (node_id)

This message is not implemented yet. The purpose is to stop the capability monitoring on a node.

UpdateMonitoredCapability (node_mon_object)

Usage: The message is sent to update the current view of the monitored capabilities.
Senders: Intermediate Monitoring Manager
Receivers: Capability and Service Administration Manager
Parameters: *node_mon_object*: a monitored node object

InformMonitoringDomain (imm_id, domain_mon_object)

This message is not implemented yet. The purpose is to inform a monitoring domain under an Intermediate Monitoring Manager to Mobility Manager. The parameter *imm_id* is the Intermediate Monitoring Manager id, and *domain_mon_object* is a monitored domain object, which is constituted by {node_mon_object}.

SetMonitoringDomain (domain_mon_object)

This message is not implemented yet. The purpose is to set a monitoring domain under an Intermediate Monitoring Manager.

SetMonitoringDomainsUnderMainManager (mmm_mon_object)

This message is not implemented yet. The purpose is to set list of monitoring domains under a Main Monitoring Manager. The parameter *mmm_mon_object* is a main manager monitoring object, which is constituted by {(imm_id, domain_mon_object)}.

3. TAPAS core platform procedures

3.1 General

A procedure has the general format *procedure_name* (*input_parameter_list*, *output_parameter_list*) An overview of the procedures and parameters are given in Table 2. A more detailed description of the procedures is given in Sec 3.2.

Table 2. Procedure names and parameters

Procedure Names	Input Parameters	Output Parameters
plugInPlay	play_name, play_version, local_location	*plug_in_play_result*
plugOutPlay	play_name, play_version	*plug_out_play_result*
changePlay	play_name, play_version, local_location	*change_role_result*
plugInRole	node_id, service_component_id, specification_location	*actor_id*
plugOutRole	actor_id	*plug_out_role_result*
changeRole	actor_id, new_ service_component_id, new_specification_location	*change_play_result*

3.2 Procedures descriptions

plugInPlay ({play_name, play_version, local_location}, {play_in_play_result})

Usage: The procedure uploads a Java project (JAR file) of a play to Web server. After the JAR file is uploaded to Web server, the play is registered. The role manuscripts in the play are also ready to be downloaded and executed by actors.

Parameters: *play_name*: name of a play, *play_version*: version number of a play (for example 1.0), *local_location*: physical local location of a JAR file to be uploaded, *plug_in_play_result*: a boolean representing the procedure result

Conditions: There is no registered play with the same name.

Interactions: This procedure sends a message CheckPlayExistanceRequest to *PlayAdm* to check whether there is a registered play with the same name.

- If there is no registered play with the same name, this procedure will get a message PlayNonExistenceResponse. Then this procedure will upload the JAR file to Web server, and it will send a message RegisterPlayRequest to *PlayAdm* to register the play.
- If there is a registered play with the same name, this procedure will get a message PlayExistenceResponse. So that, this procedure stops processing.

Results:

- *On success*: A JAR file of a play is uploaded to Web server. The play is registered and it is ready for use. This means from now the role manuscripts in the play can be downloaded and executed by actors. The procedure returns output = true.
- *On failure*: A JAR file of a play is not uploaded to Web server. The play is not registered and it is not for use. The procedure returns output = false.

plugOutPlay ({play_name, play_version}, {plug_out_play_result})

Usage: The procedure unregisters a play, and it removes an uploaded JAR file of the play from Web server.

Parameters: *play_name*: name of a play, *play_version*: version number of play, *plug_out_play_result*: a boolean representing the procedure result

Conditions: The play is not in use. It means that there is no actor executing the role manuscripts in the play.

Interactions: This procedure sends a message CheckPlayInUseRequest to *PlayAdm* to check whether a play is in use.

- If the play is not in use, this procedure will get a message PlayNotInUseResponse. Then this procedure will send a message GetPlayLocationRequest to *PlayAdm* to get a current location of the uploaded JAR file on Web server, and it will get a message PlayLocationResponse. Finally, this procedure will send a message UnregisterPlayRequest to *PlayAdm*, and it will remove the JAR file from Web server.

- If the play is in use, this procedure will get a message PlayInUseResponse. So that, this procedure stops processing.

Results:

- *On success*: A play is unregistered, and its JAR file is removed from Web server. The procedure returns output = true.

- *On failure*: A play is still registered and for use, and its JAR file is on Web server further. The procedure returns output = false.

changePlay ({play_name, play_version, local_location}, {change_play_result})

Usage: The procedure re-uploads a JAR file of a registered play to Web server, and informs about this new version of the play.

Parameters: *play_name*: name of a play, *play_version*: version number of a play, *local_location*: physical local location of a JAR file to be uploaded, *change_play_result*: a boolean representing the procedure result

Conditions: The play has been registered. A play can be changed and re-uploaded to Web server. However, in case a play is in use, it depends on the nature of services and service components that whether a new version of that play as well as the role manuscripts in that play can be applied immediately or not.

Interactions: This procedure sends a message CheckPlayExistanceRequest to *PlayAdm* check whether the play has been registered.

- If the play has been registered, this procedure will get a message PlayExistenceResponse. Then this procedure will upload the JAR file to Web server. Finally, it will send a message ChangePlayReport to *PlayAdm* to inform that a new version of the play is re-uploaded.

- If the play has not been registered, this procedure will get a message PlayNonExistenceResponse. So that, this procedure stops processing.

Results:

- *On success*: A new version of a registered play is uploaded to Web server. *PlayAdm* is aware of the new version. The procedure returns output = true.
- *On failure*: A new version of a registered play is not uploaded. The procedure returns output = false.

plugInRole ({node_id, service_component_id, specification_location}, {actor_id})

Usage: The procedure starts a role figure in a selected node. It is to assign a role from a registered play to an actor on the node.

Parameters: *node_id*: IP address of a selected node, *service_component_id*: a service component id, *specification_location*: location of the role manuscript on Web server, *actor_id*: an actor id of whom constitutes the role figure

Conditions: The play has been uploaded and registered.

Interactions: This procedure sends a message StartServiceComponentOnThisNodeRequest to Director, which is executing in a selected node.

- If the play has been registered and the role exists in the play, this procedure will get a message StartServiceComponentOnThisNodeSuccess.
- If the play has not been registered or the role does not exist, this procedure will get a message StartServiceComponentOnThisNodeFailure.

Results:

- *On success*: An actor in a selected node is assigned a role. The actor automatically downloads the role manuscript and starts executing the manuscript. It is also defined as the actor plays the role, or the actor constitutes a role figure. The procedure returns the actor id.
- *On failure*: A role is not assigned to any actor. The procedure returns null.

plugOutRole ({actor_id}, {plug_out_role_result})

Usage: The procedure stops an actor playing a role.

Parameters: *actor_id*: an actor id, *plug_out_role_result*: a boolean representing the procedure result

Conditions: An actor exists.

Interactions: This procedure sends a message plugOutRoleFromActorRequest to *Director* in a node.

- If the actor exists, this procedure will get a message PlugOutRoleOnActorSuccess.
- If the actor does not exist, this procedure will get a message PlugOutRoleOnActorFailure.

Results:

- *On success*: An actor stops playing a role. The procedure returns output = true.

- *On failure*: An actor, which could be in another node, still plays a role. The procedure returns output = false.

changeRole ({actor_id, new_service_component_id, new_specification_location}, {change_role_result})

Usage: The procedure makes an actor change from a current role to a new role.

Parameters: *actor_id*: the identification of an actor, *new_service_component_id*: a new service component id, *new_specification_location*: location of the new role manuscript on Web server, *change_role_result*: a boolean representing the procedure result

Conditions: The play, which is composed of the new role, has been uploaded and registered. Also, an actor exists.

Interactions: This procedure sends a message ChangeRoleOnActorRequest to *Director* in a node.

- If the play has been registered, the role exists in the play and the actor exists, this procedure will get a message ChangeRoleOnActorSuccess.
- If the play has not been registered, the role does not exist or the actor does not exist, this procedure will get a message ChangeRoleOnActorFailure.

Results:

- *On success*: An actor starts playing a new role. The procedure returns output = true.
- *On failure*: An actor continues playing a current role. The procedure returns output = false.