SOFTWARE DETAILED DESIGN

SmartDeviceLink

56 pages

Table of contents

1. Introduction

This document provides an overview of the design for the SmartDeviceLink AQ project.

1.1. Purpose

The document is intended to provide those persons involved in development, maintenance and support with sufficient, detailed information concerning the design, development and deployment concepts, to accomplish their respective tasks without reliance on the authors.

1.2. Scope

The purpose of the document is to provide the detailed design for the software units described in Software Architecture Design Document (SAD). In particular, this document specifies the following:

- Module decomposition
- Detailed description of the module interfaces
- Detailed design for each module
- Dependencies between modules
- External interfaces of the running processes within the software

1.3. Abbreviations

This subsection provides the definitions of all terms, acronyms, and abbreviations required to properly interpret the Software detailed design.

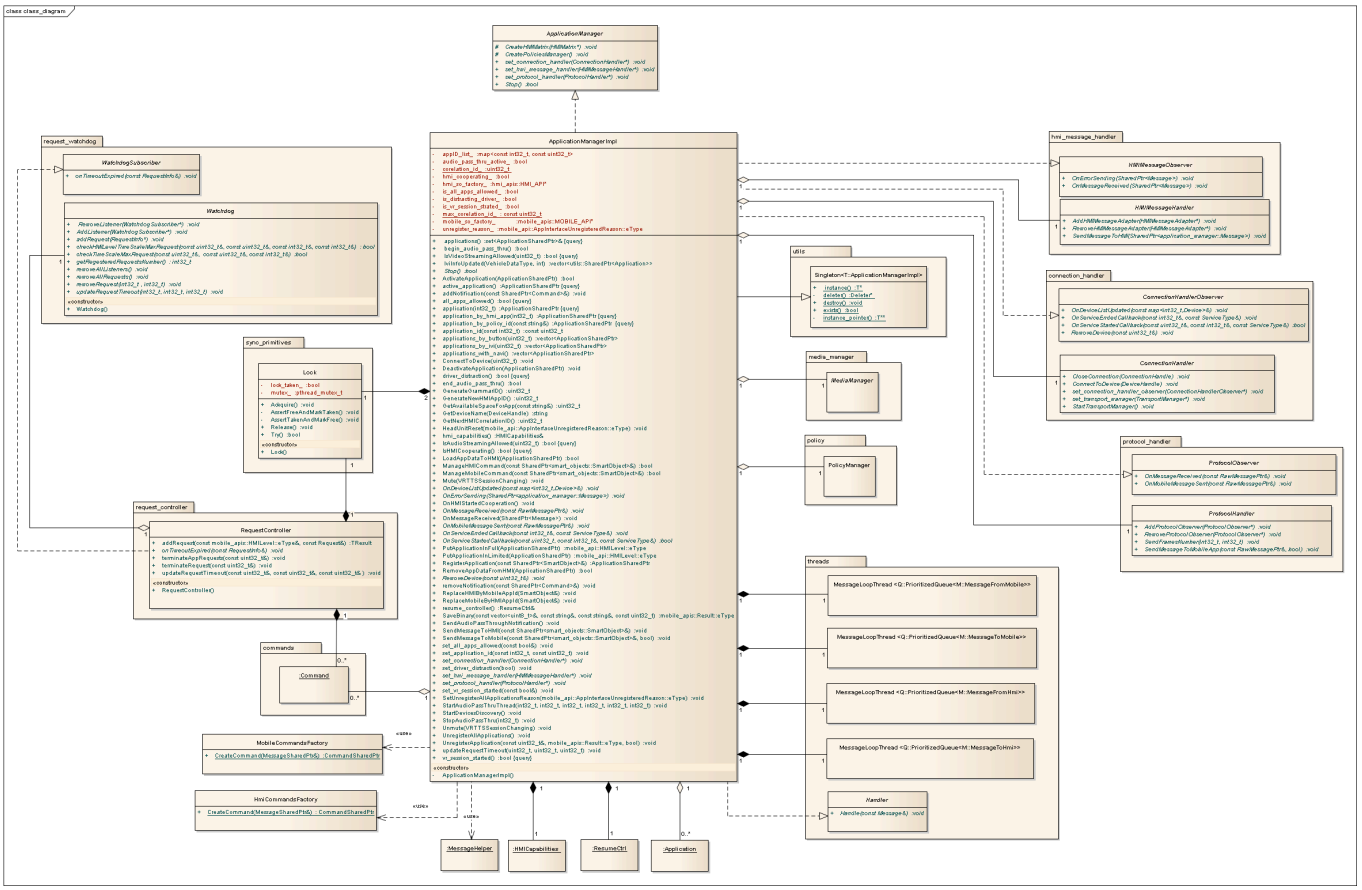
Acronym/Abbreviation	Definition
SDL	SmartDeviceLink
HMI	Human Machine Interface

2. Module Detailed Design

In this section the detailed design for each module (component) of the software is provided. It gives a closer look at what each module does in a systematic way.

2.1. application_manager Detailed Design

2.1.1. Class Diagram



2.1.2. Design Description

- Component is used for main business logic implementation.
- All JSON RPC messages from HMI and mobile application are implemented like a command pattern.
- All commands objects will be created by MobileCommandsFactory or HmiCommandsFactory.
- Watchdog is used for all messages which will go to HMI.
- Implementation of Application Manager can contain various thread counts for Mobile or HMI message implementation (AudioPassThru for example).
- Resume controller is used for saving applications state on disconnect and ignition off.

2.2. HMICapabilities Detailed design

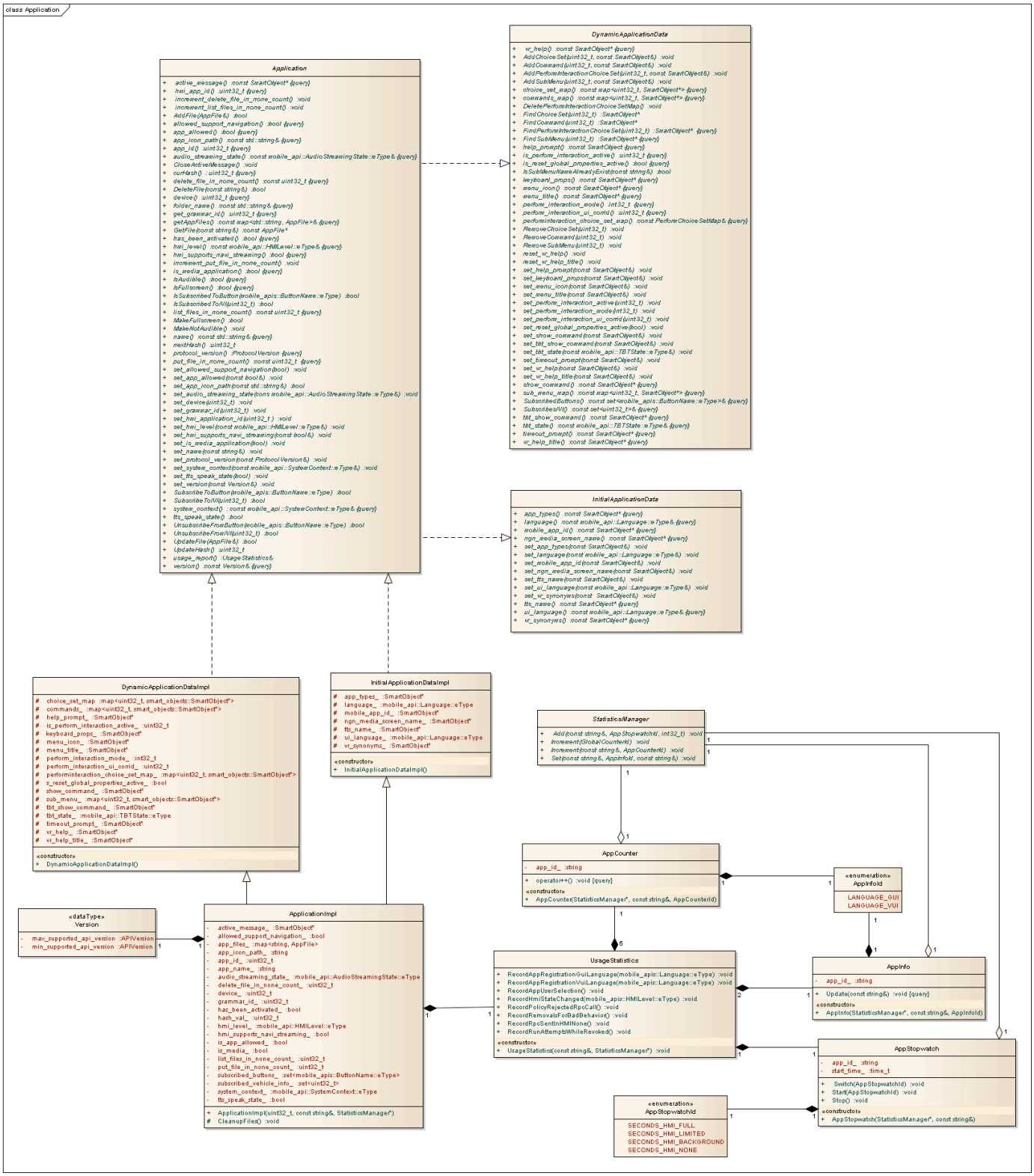
2.2.1. Class Diagram

2.2.2. Design Description

HMICapabilities is used for downloading and saving hmi capabilities from file “hmi_capabilities.json”.

2.3. Application Detailed design

2.3.1. Class Diagram

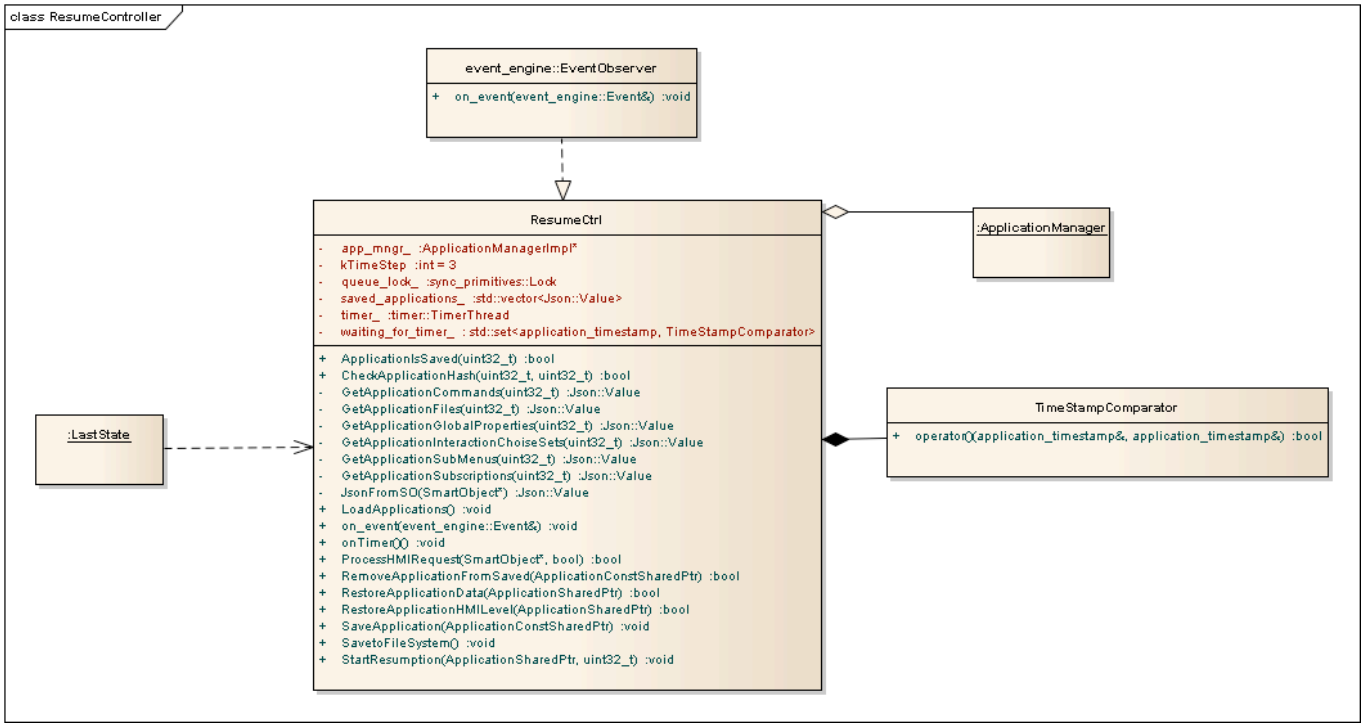


2.3.2. Design Description

Class of application stores data about state of registered applications. Application_manager stores list of registered applications.

2.4. Resume Controller Detailed design

2.4.1. Class Diagram



2.4.2. Design Description

Resume controller is used for storing data and HMI level in case of ignition off or disconnecting application.
Data that is must not be lost throw ignition cycles is:

D1: RegisterAppInterface
D2: AddCommand (VR+Menu) / AddSubMenu
D3: CreateInteractionChoiceSet
D4: SetGlobalProperties
D5: Subscriptions (buttons and vehicle data)
D6: PutFile
D7: Show + MediaClock
D8: Policy data (e.g. consent records / device data; usage and error count)

Resume controller manage D2-D7 data.
Restoring application after disconnecting or IGN_OFF can occur under 2 scenarios:

1) Restore only HMI level and audio streaming state
2) Restore all data and HMI level audio streaming state and D2-D7 data

ResumeCtrl decides to use 1 or 2 scenario depending on hashid in RegisterAppInterface request.
Is hashid's matches will be used 2 scenario, otherwise will be used first one.
Hash id is string constant that updates by SDL after every succeed D2-D7 request, and sends to mobile with notification. Hash allows ResumeCtrl to understand is registering application is the same instance as saved or not.

When application is registering with hashID, and Resume controller stored info about this application, it will initiate restoring D2-D7 data and send RegisterAppInterface response with *resume* = *succes* flag. D2, D3, D5 requests will be sent to HMI. ResumeCtrl is inherited from EventObserver, and will track responses from HMI.

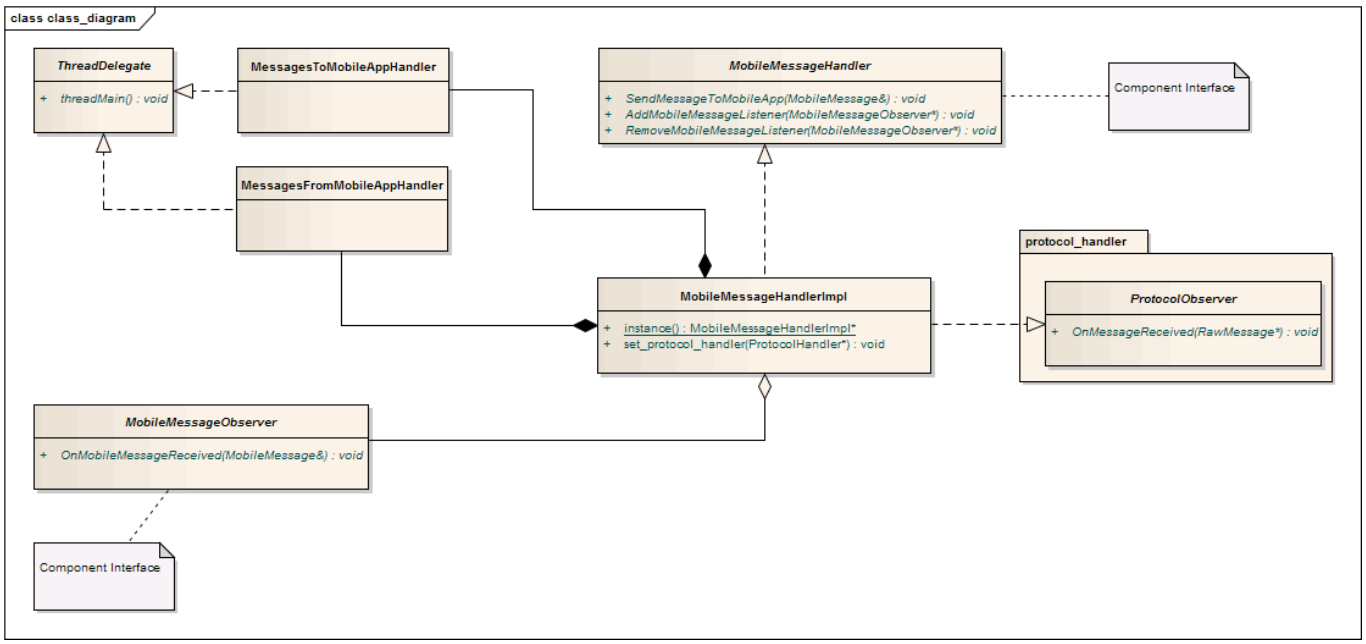
If 2 or more applications are trying to resume at the same time (it means that between registering applications is less 3 seconds) , HMI levels conflicts will be resolve according tables SDLAQ-CRS-2732 SDLAQ-CRS-2731:

Restoring HMI level will occur in 3 seconds after RegisterAppInterface response to handle other application, that need to be resumed.
In case if any application is already registered HMI level will be restored immediately.

TimeStampComparator is used for sorting applications by disconnect timestamp. Privilege for restoring HMI status given to application, that was disconnected later.

2.5. Mobile_message_handler Detailed Design

2.5.1. Class Diagram



2.5.2. Design Description

The component:

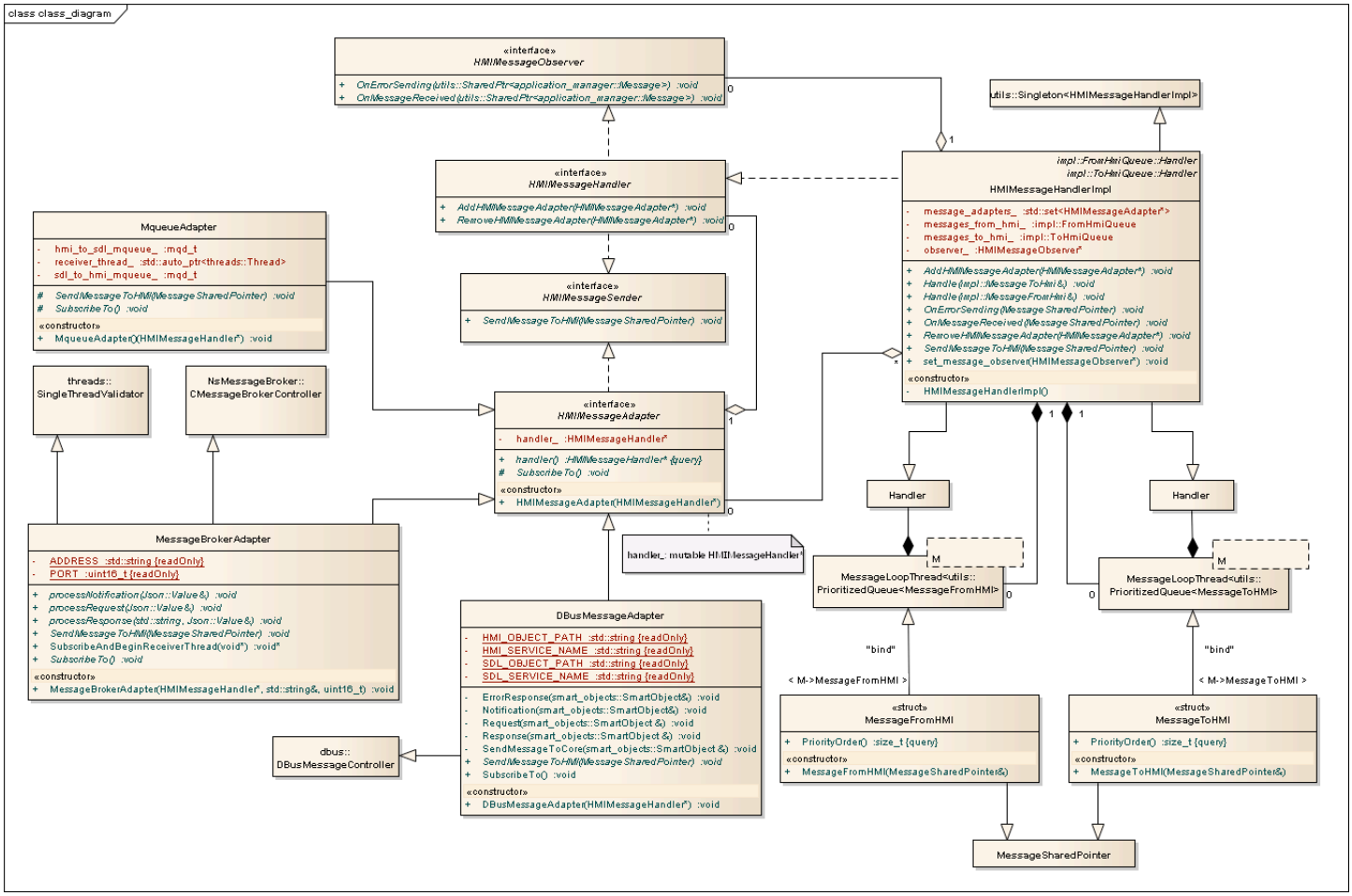
- Is used for data preparing for ApplicationManager.
- Converts the protocol message to known and not depended from protocol or API version message for ApplicationManager.
- Contains two threads for incoming and outgoing data.
- Communicates with ProtocolHandler and ApplicationManager.

Observer pattern is used for processing messages from the mobile application. MobileMessageObserver is implemented by ApplicationManager.

MobileMessageHandler interface is used for processing messages to the mobile application.

2.6. hmi_message_handler Detailed Design

2.6.1. Class Diagram



2.6.2. Detailed Design

Component

- Is a proxy between transport layer for communication with HMI and Application Manager.
- Represents an abstraction layer for communication with HMI and application_manager.
- Different types of transport for communication with HMI will be implemented by HMIMessageAdapter.

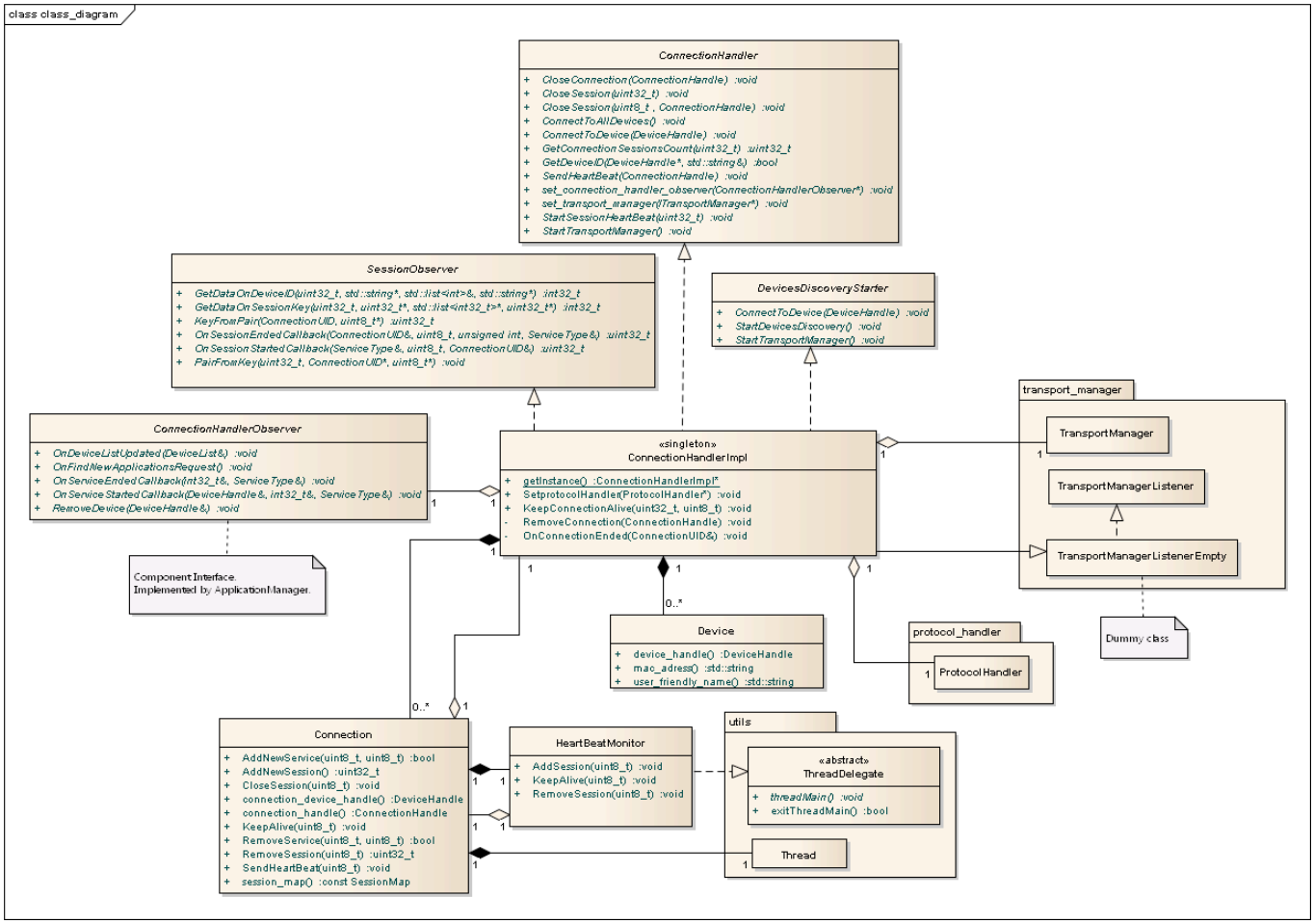
Observer pattern is used for processing messages from HMI. HMIMessageObserver is implemented by application_manager.

HMIMessageHandler is used for message processing to HMI.

Message for HMI placed in queue FIFO MessageLoopThread<utils::PrioritizedQueue<MessageToHmi> >. This queue, read separate thread and sent to HMI message is queue not empty and then remove it. The same process is performed when the message comes from HMI, but queue name MessageLoopThread<utils::PrioritizedQueue <MessageFromHmi> >.

2.7. connection_handler Detailed Design

2.7.1. Class Diagram



2.7.2. Detailed Design

Component

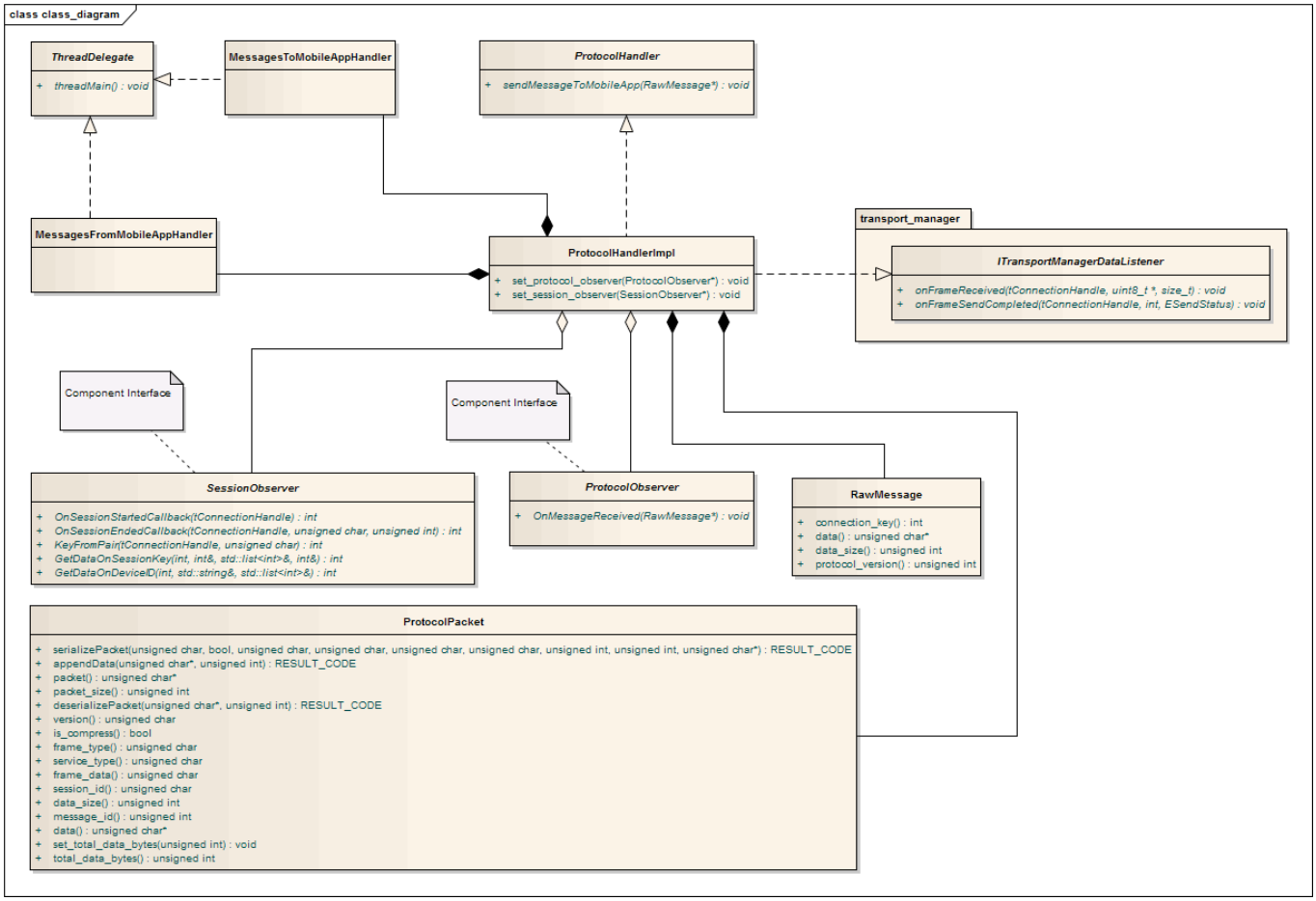
- Is a mid-layer between the low-level connections and Application instances in Application Manager.
- Handles connections and manages applications sessions.
- Communicates with TransportManager, ProtocolHandler, ApplicationManager.

Observer pattern is used for processing messages from mobile application/device. ConnectionHandlerObserver is implemented by ApplicationManager.

CohhectionHandler interface is used for processing messages to mobile application/device.

2.8. protocol_handler Detailed Design

2.8.1. Class Diagram



2.8.2. Detailed Design

Component

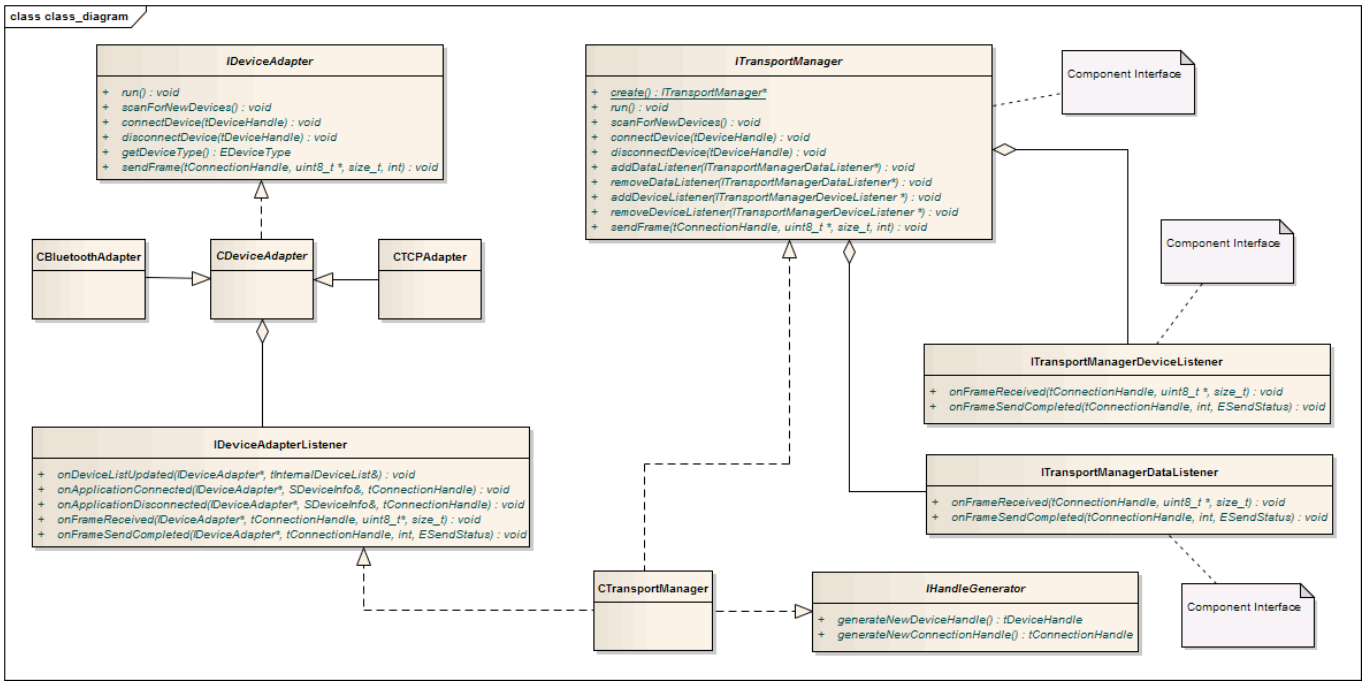
- Is a mid-layer between TransportManager and higher level components.
- Handles the control messages and works with protocol structure defined for communication between Mobile Application and SDL.
- Contains two threads for incoming and outgoing data.
- Communicates with MobileMessageHandler, ConnectionHandler, TransportManager.

Observer pattern is used for processing message from a mobile application. SessionObserver is implemented by ConnectionHandler, ProtocolObserver is implemented by MobileMessageHandler.

ProtocolHandler interface is used for processing messages to mobile application.

2.9. transport_manager Detailed Design

2.9.1. Class Diagram



2.9.2. Design Description

Component

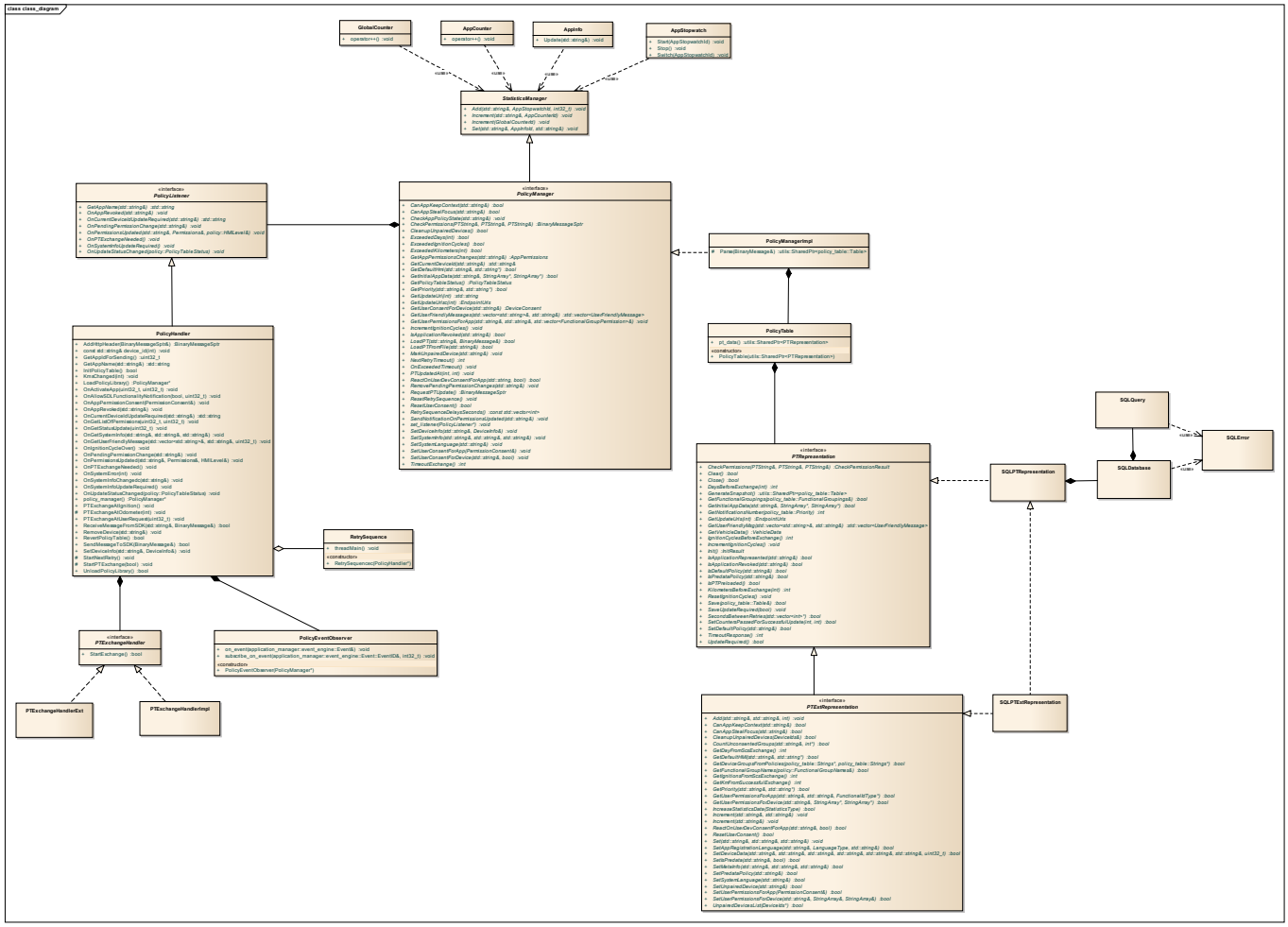
- Is used for handling the low-level transport connections between SDL and Mobile Device.
- Represents an abstraction layer between the low-level transport communication and the protocol layer. Different types of transports will be implemented with IDeviceAdapter.
- Communicates with ProtocolHandler and ConnectionHandler.

Observer pattern is used for processing messages from mobile application. ITransportManagerDataListener is implemented by ProtocolHandler, ITransportManagerDeviceListener is implemented by ConnectionHandler.

ITransportManager interface is used for processing messages to mobile application or scanning for new devices/applications.

2.10. Policy detailed design

2.10.1. Class diagram



2.10.2. Design description

Component is used to:

- enable advanced functionality and protect SDL APIs from unauthorized use
- allow mobile applications to be granted elevated privileges, while providing usage data to a backend server
- remotely manage SDL-enabled apps, including app-specific and device-specific access to system functionality.

Communicates with PolicyHandler class.

Implemented as pluggable shared library object.

3. Process Detailed Design

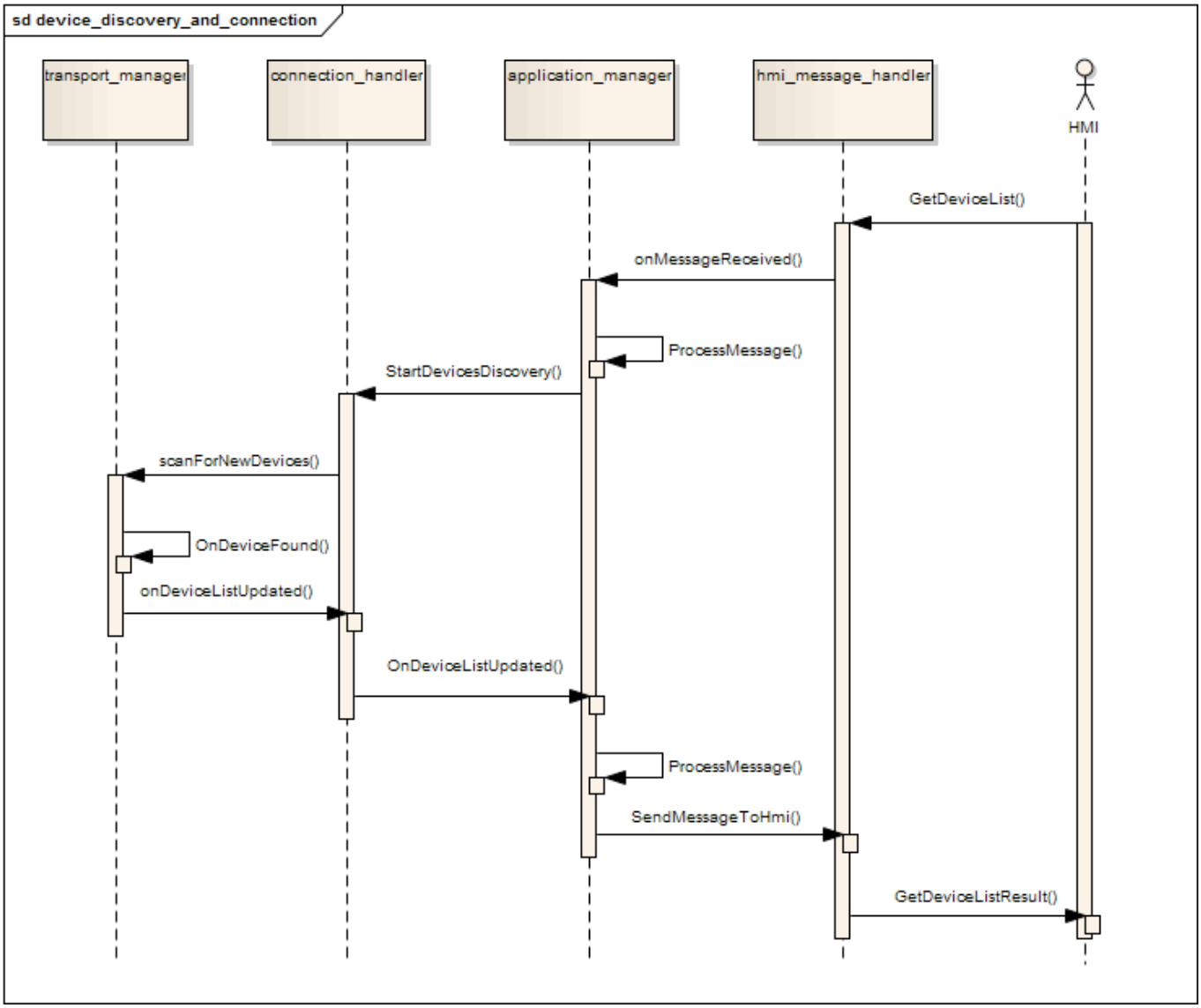
This section provides the system decomposition to the running processes. Evaluate, classify and document the dynamic behavior of the software. The dynamic behavior is determined by operating modes (e.g. start-up, shutdown, normal mode, calibration, diagnosis, etc.), processes, tasks, threads, time slices, interrupts, etc.

The processes detailed design is illustrated with sequence diagrams.

3.1. SDL process detailed design

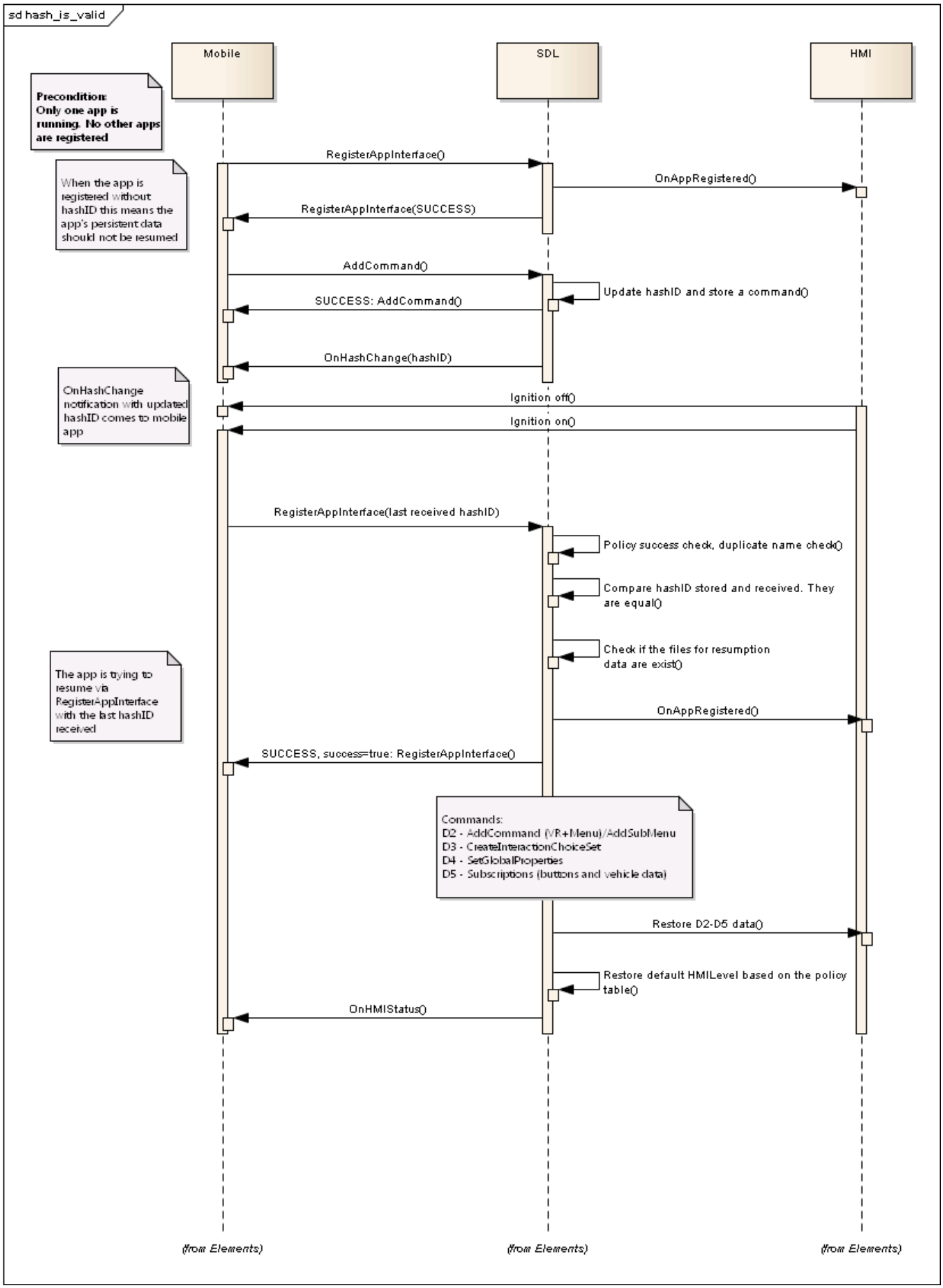
3.1.1. Search device detailed design

The following diagram shows the device discovery. Detailed information about presented components please find below.

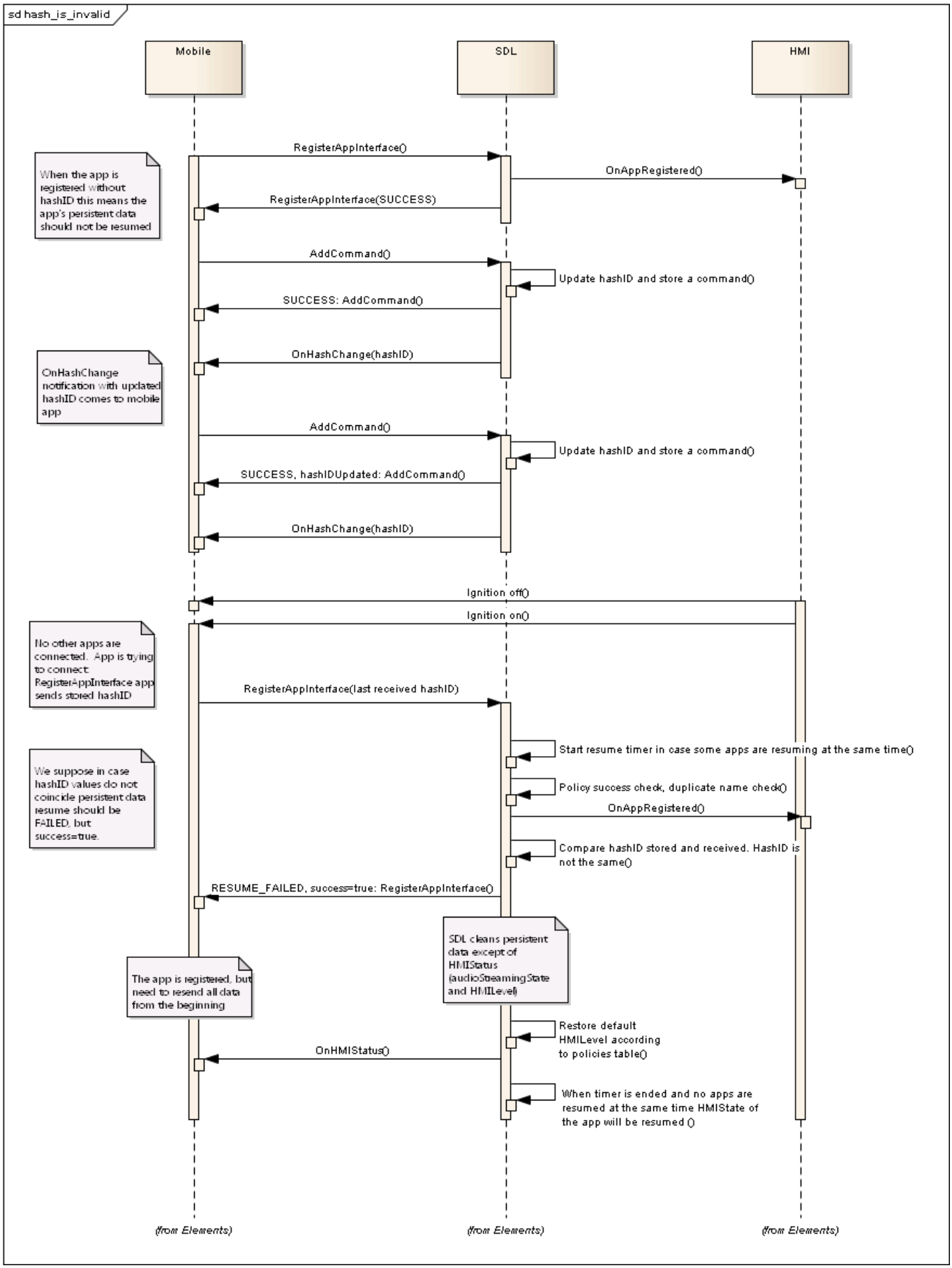


3.1.2. Resume controlled detailed design

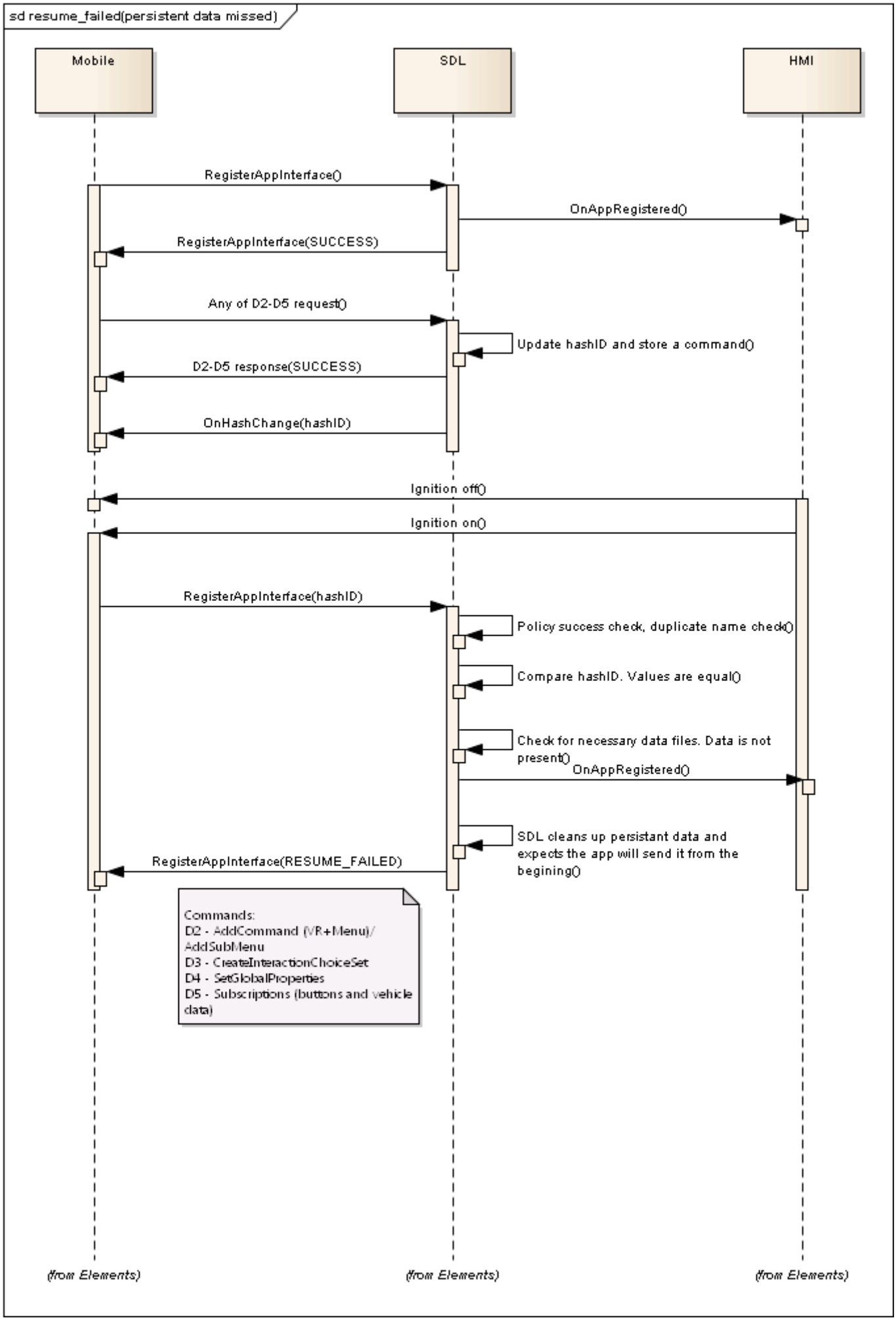
Following diagram shows successes resumption/



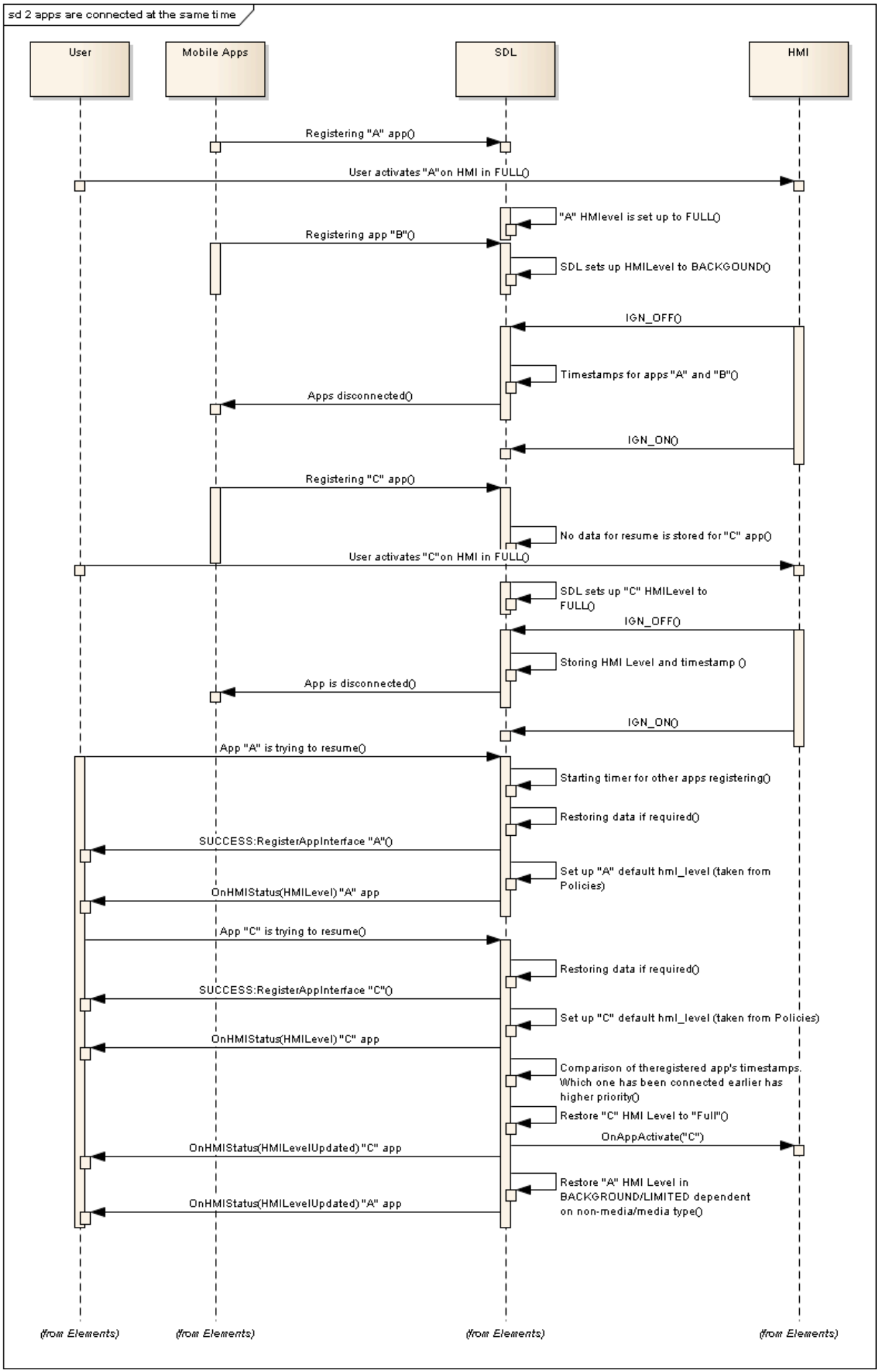
Following diagram shows failed resumption, if hash id's are not matched



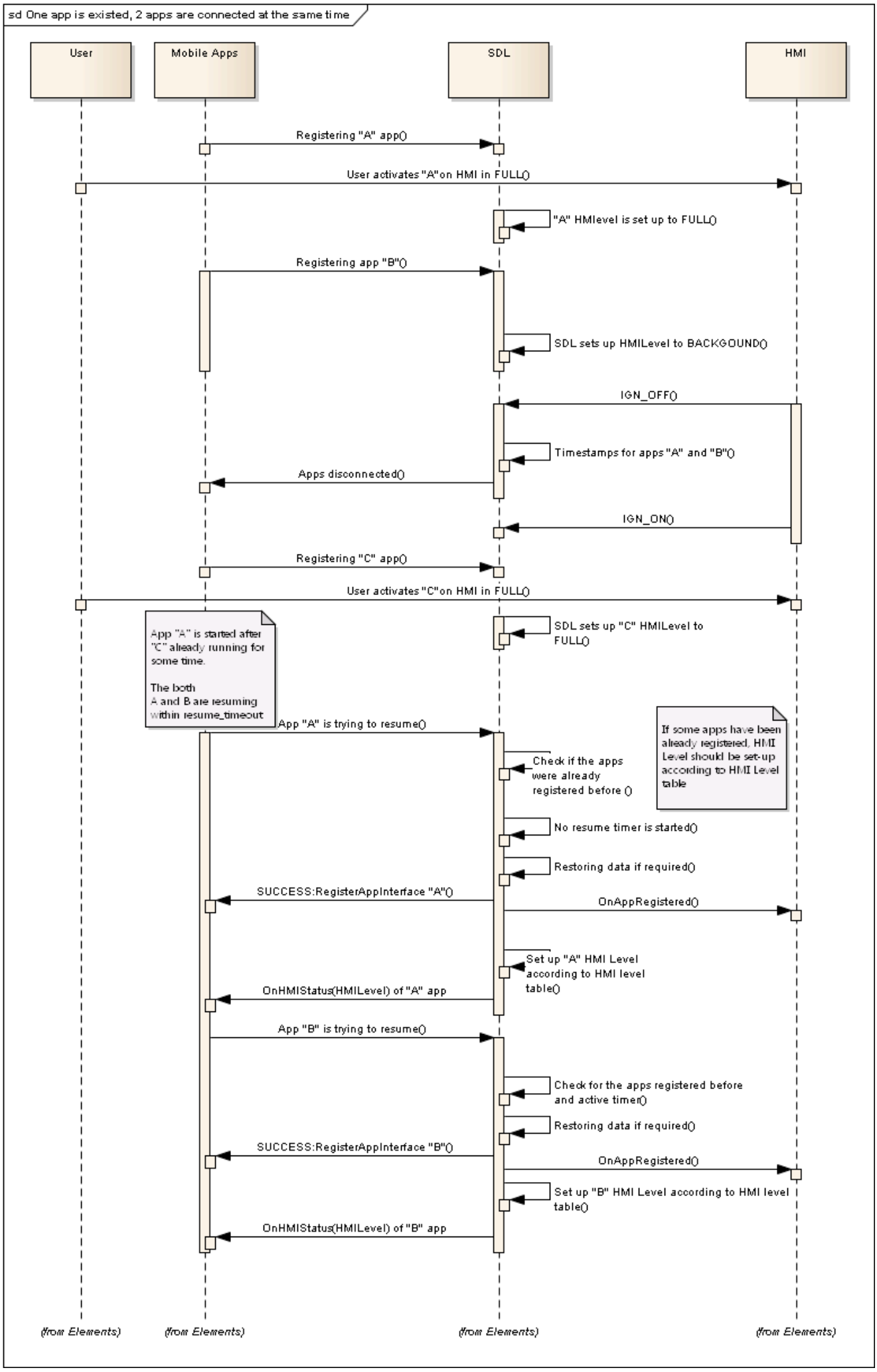
Following diagram shows resume failed sequences is some of persistent data, than need for resumptions is missed



Following diagram shows case if more then one application is trying to resume

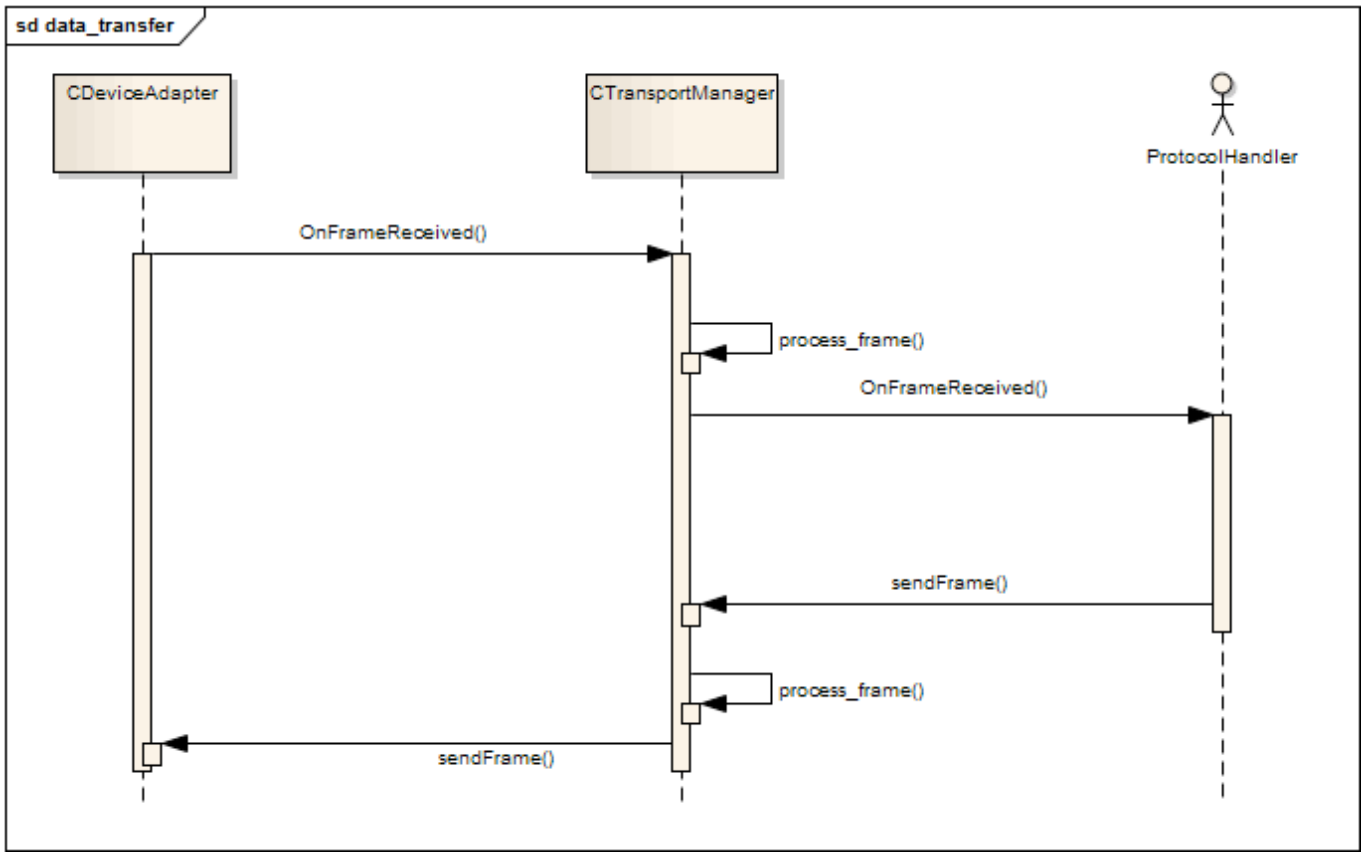


Following diagram shows if application trying to resume, and there are already registered application exists

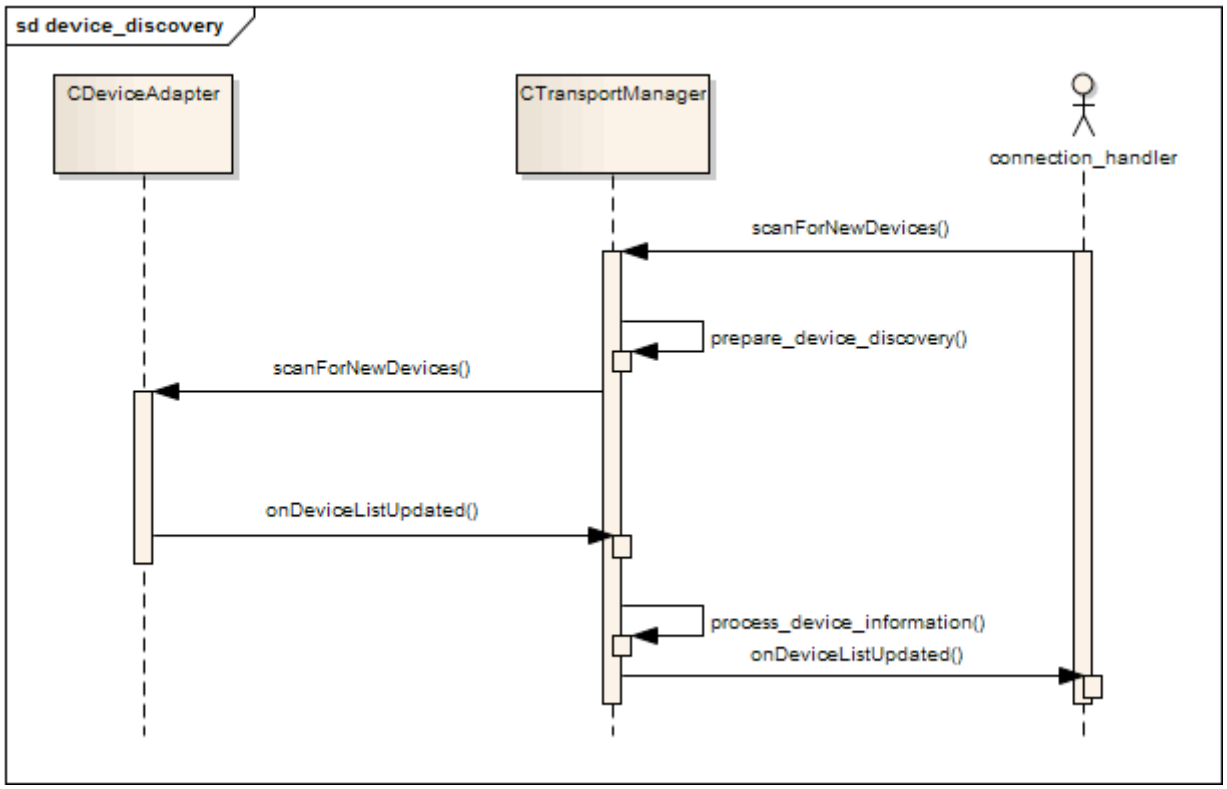

sd One app is existed, 2 apps are connected at the same time
User
Mobile Apps
SDL
HMI
Registering "A" app()
User activates "A"on HMI in FULL()
"A" HMIlevel is set up to FULL()
Registering app "B"()
SDL sets up HMILevel to BACKGOUND()
IGN_OFF()
Timestamps for apps "A" and "B"()
Apps disconnected()
IGN_ON()
Registering "C" app()
User activates "C"on HMI in FULL()
SDL sets up "C" HMILevel to FULL()
App "A" is started after "C" already running for some time.
The both A and B are resuming within resume_timeout
App "A" is trying to resume()
If some apps have been already registered, HMI Level should be set-up according to HMI Level table
Check if the apps were already registered before ()
No resume timer is started()
Restoring data if required()
SUCCESS:RegisterAppInterface "A"()
OnAppRegistered()
Set up "A" HMI Level according to HMI level table()
OnHMIStatus(HMILevel) of "A" app
App "B" is trying to resume()
Check for the apps registered before and active timer()
Restoring data if required()
SUCCESS:RegisterAppInterface "B"()
OnAppRegistered()
Set up "B" HMI Level according to HMI level table()
OnHMIStatus(HMILevel) of "B" app
(from Elements)
(from Elements)
(from Elements)
(from Elements)

3.1.3. transport_manager detailed design

The following diagram shows the device discovery by transport_manager component

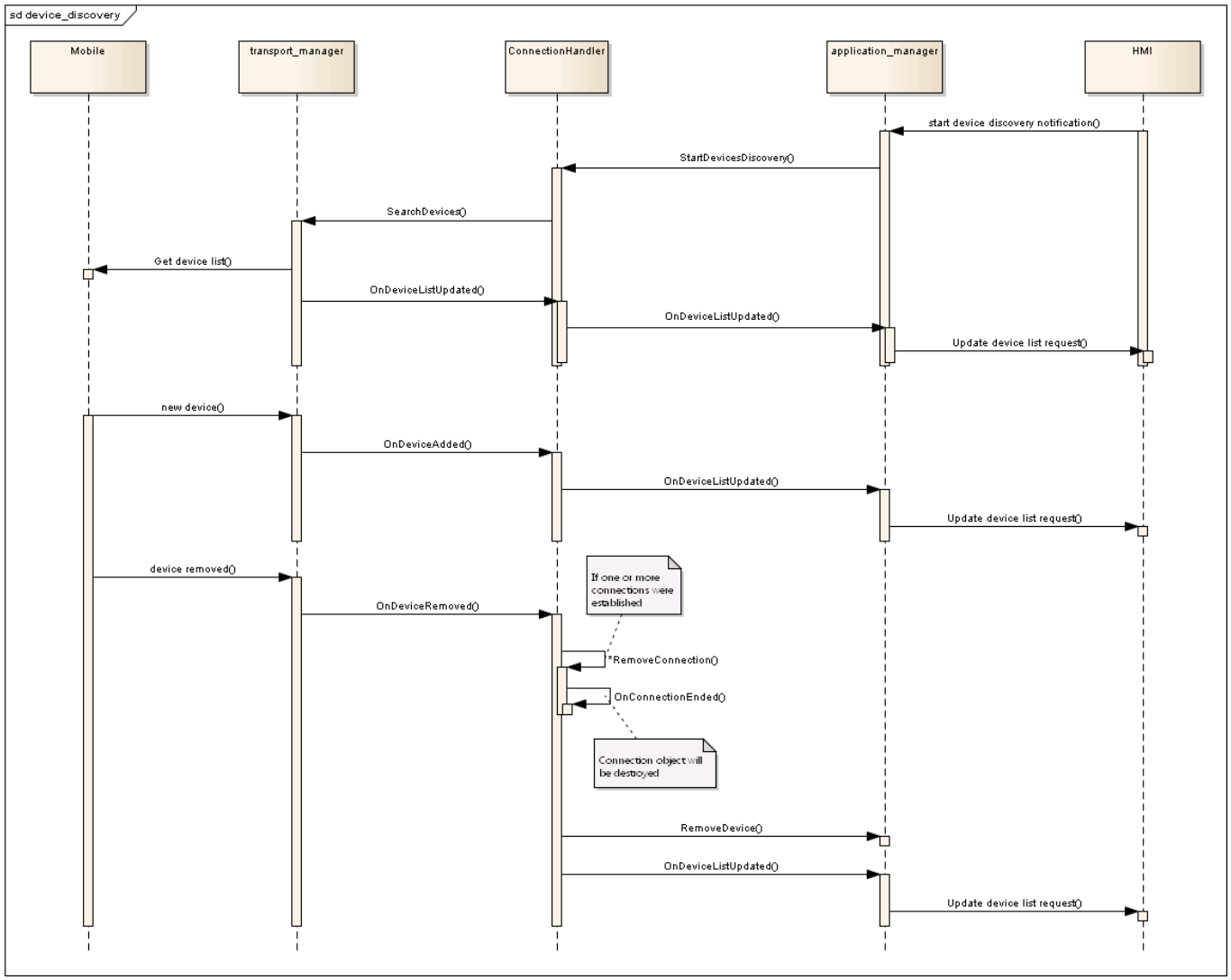


The following diagram shows the data transfer between mobile application and SDL by transport_manager.

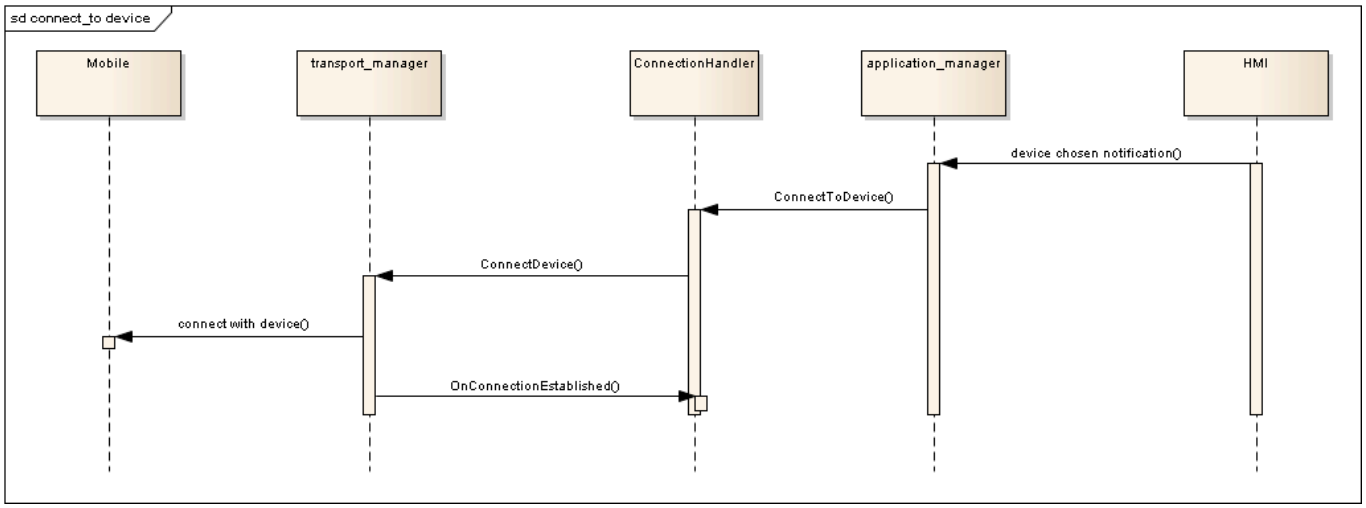


3.1.4. connection_handler detailed design

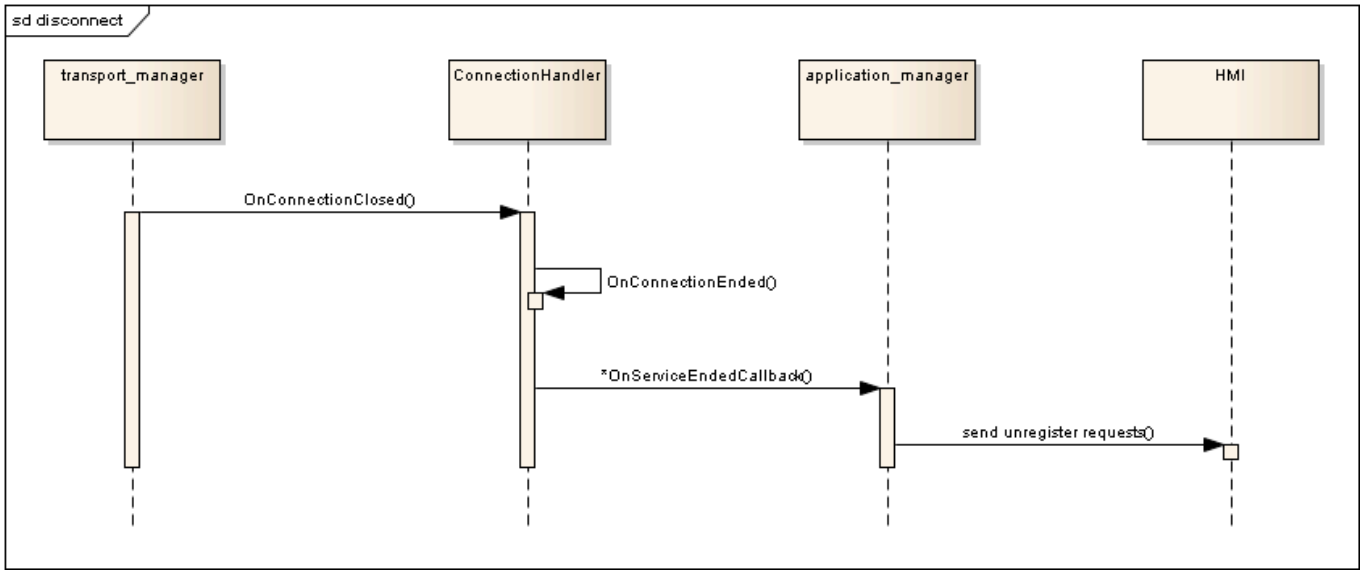
The following diagram shows the device discovery by connection_handler.



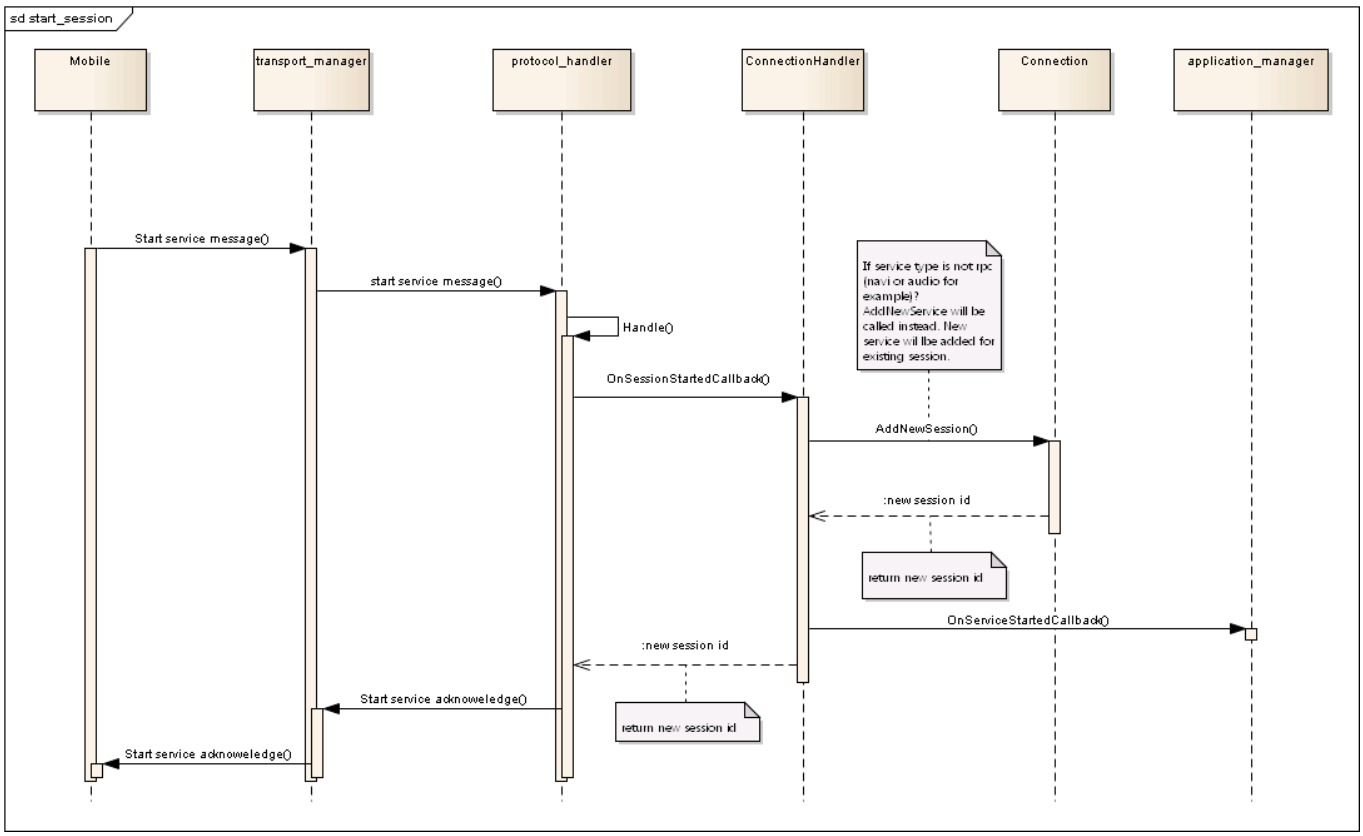
Connect to device



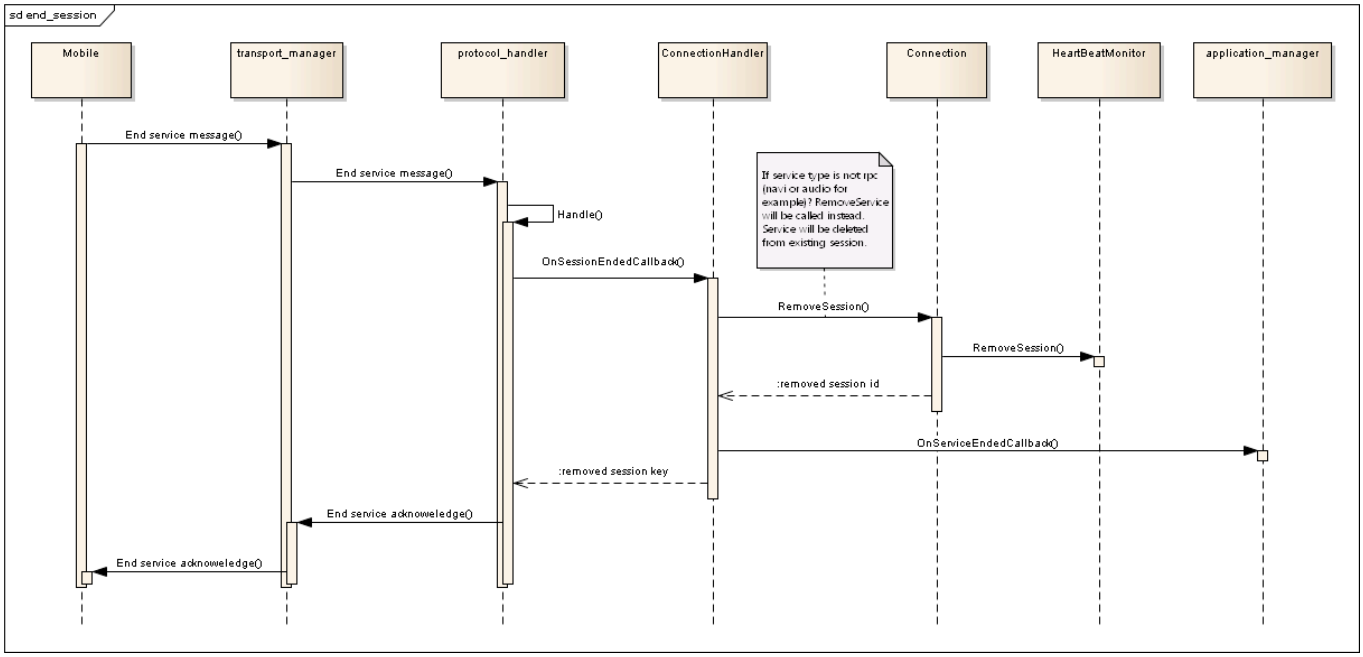
Disconnect sequence



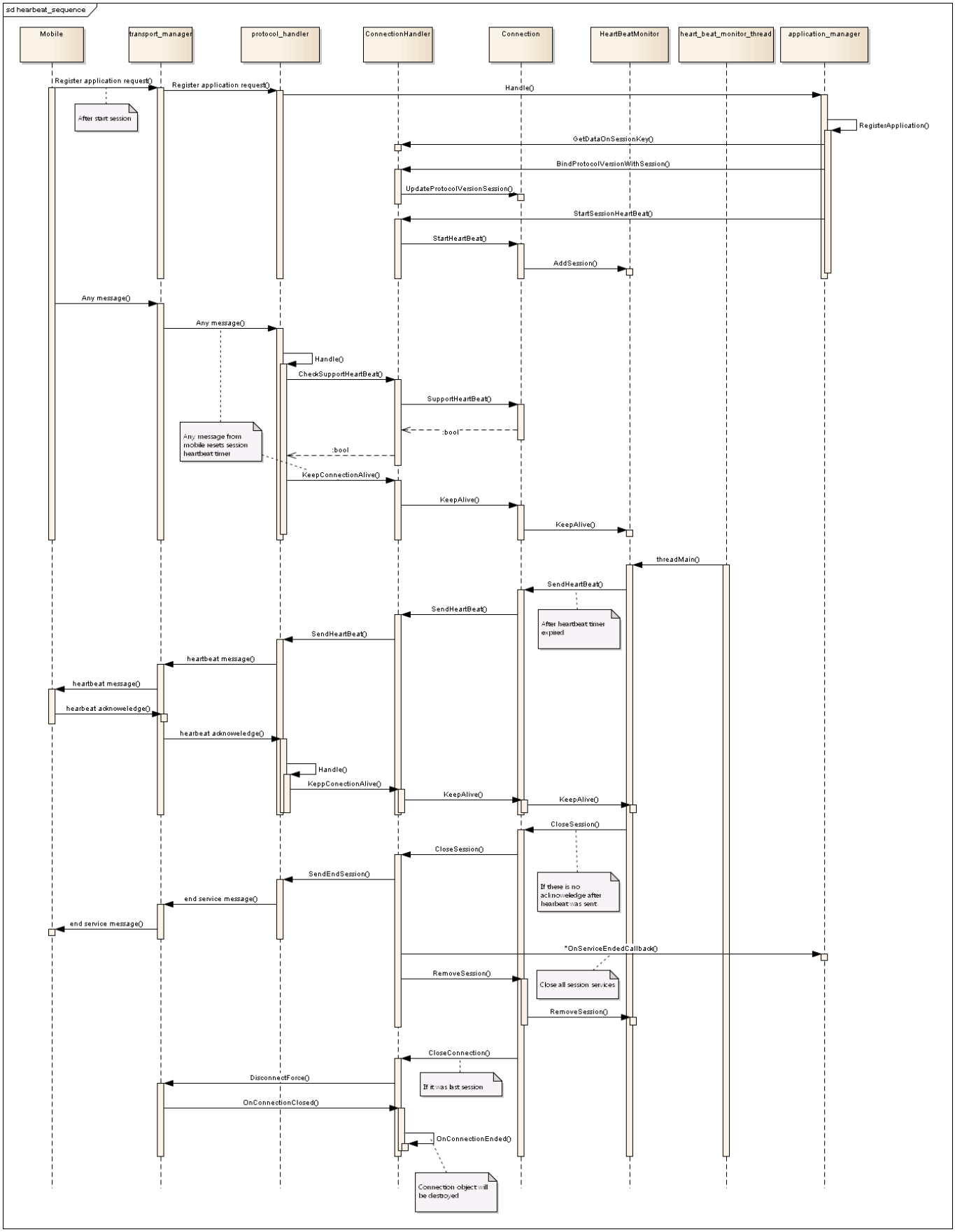
Start session



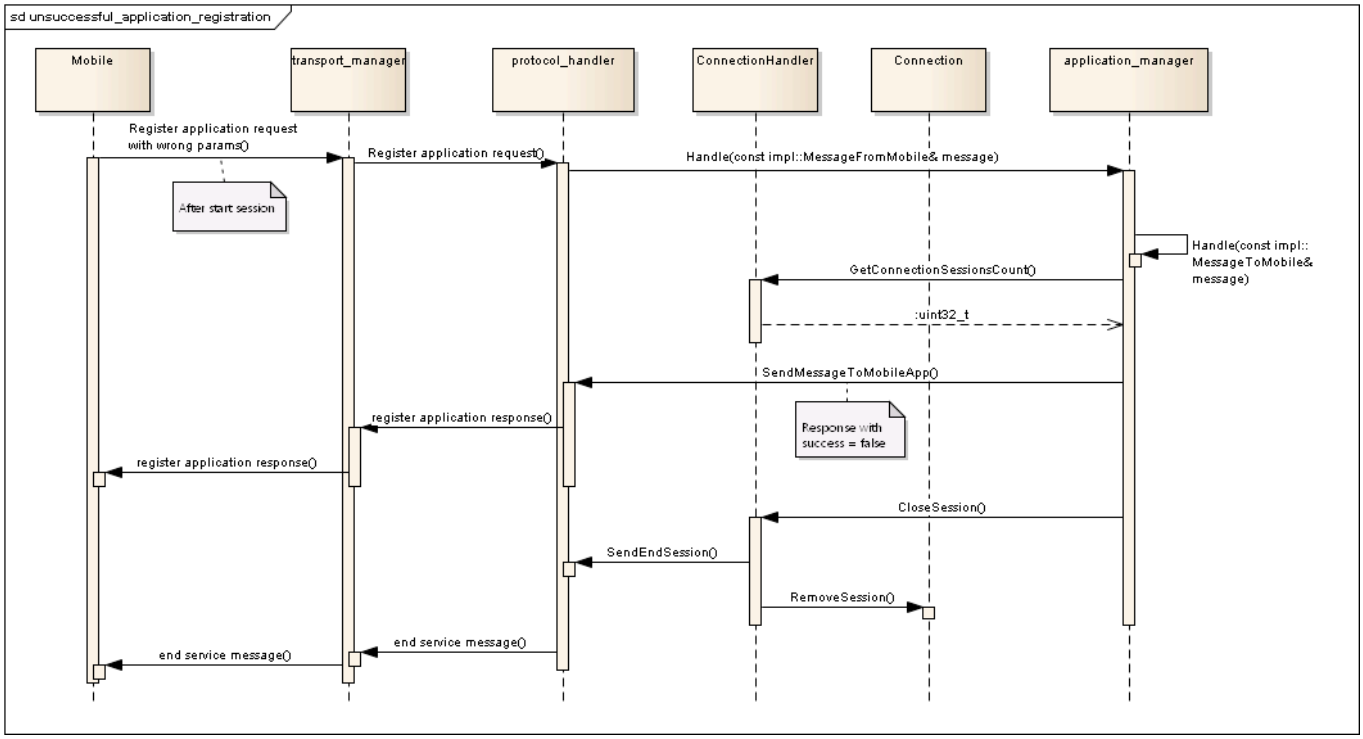
End session


sd end_session
Mobile
transport_manager
protocol_handler
ConnectionHandler
Connection
HeartBeatMonitor
application_manager
End service message()
End service message()
Handle()
If service type is not rpc (navi or audio for example) ? RemoveService will be called instead. Service will be deleted from existing session.
OnSessionEndedCallback()
RemoveSession()
RemoveSession()
:removed session id
OnServiceEndedCallback()
:removed session key
End service acknoweledge()
End service acknoweledge()

Heartbeat sequence

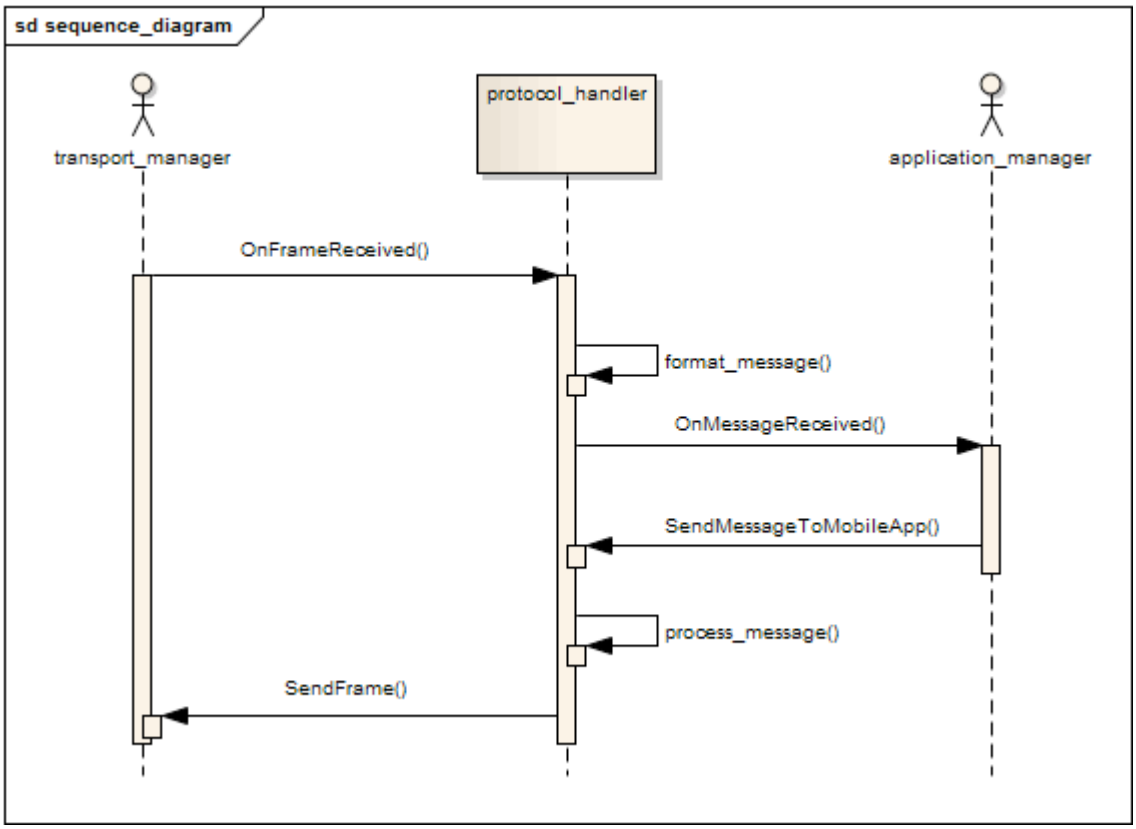


Register application request wit wrong parameters



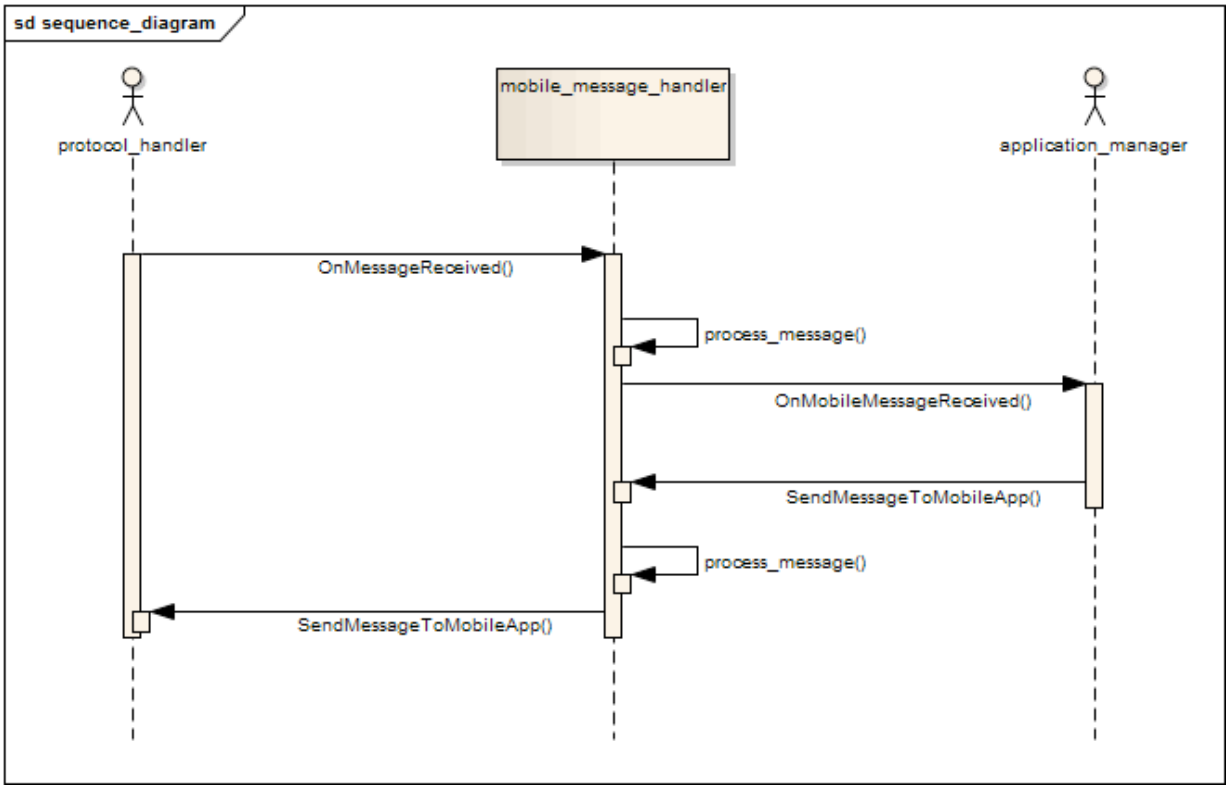
3.1.5. protocol_handler detailed design

The following diagram show message processing by protocol_handler.



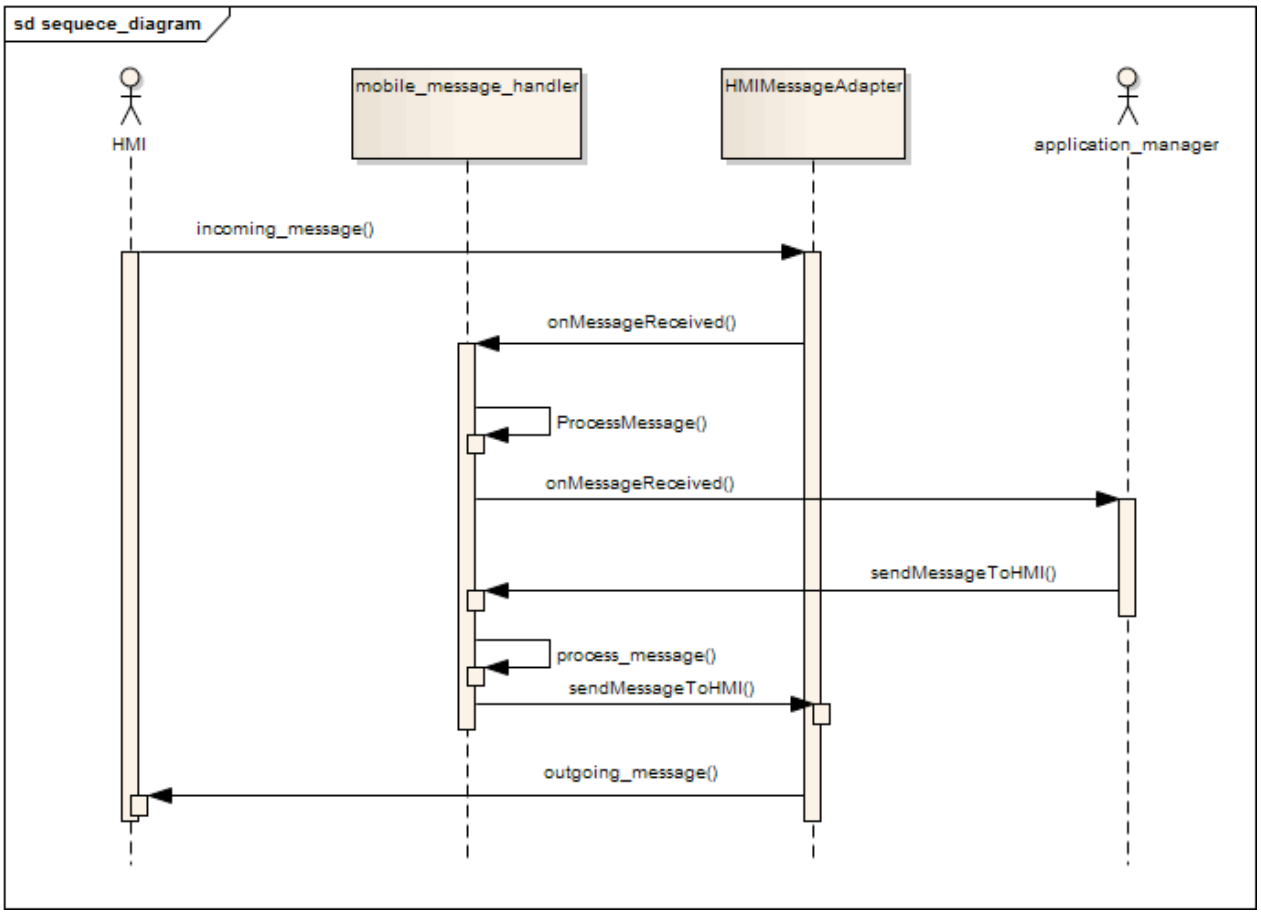
3.1.6. mobile_message_handler detailed design

The following diagram shows the basic message processing from protocol_handler to application_manager.



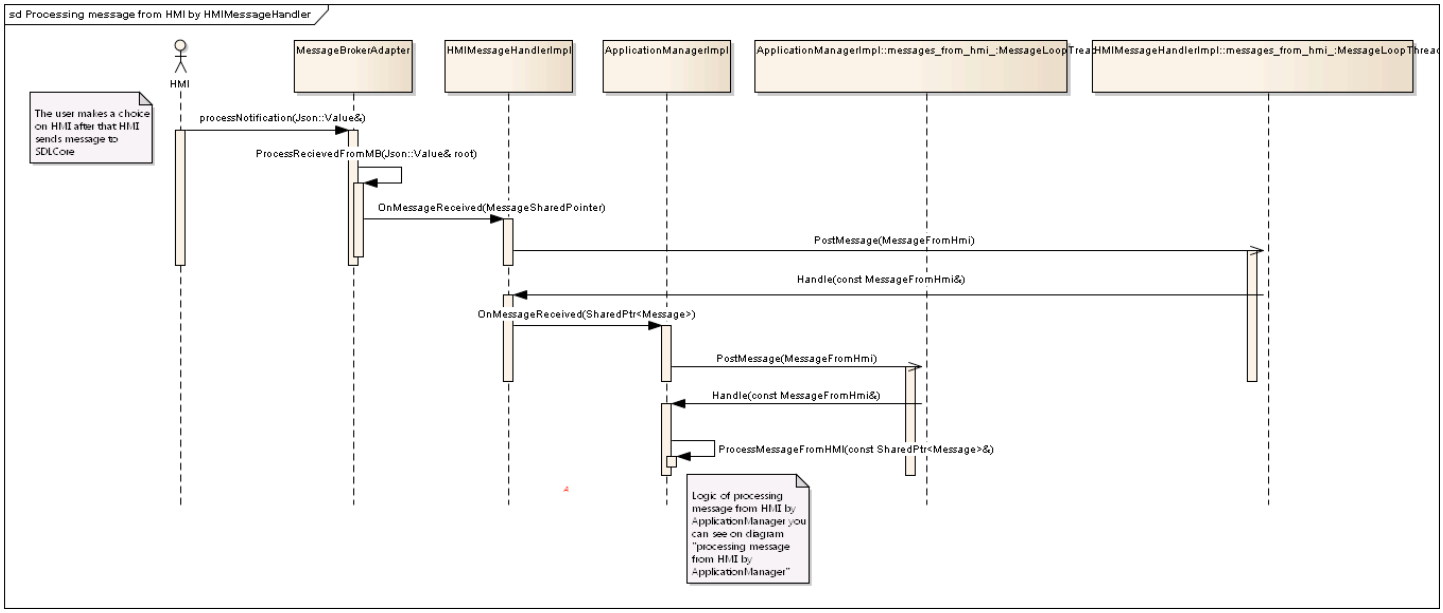
3.1.7. mobile_message_handler detailed design

The following diagram show message processing by mobile_message_handler.

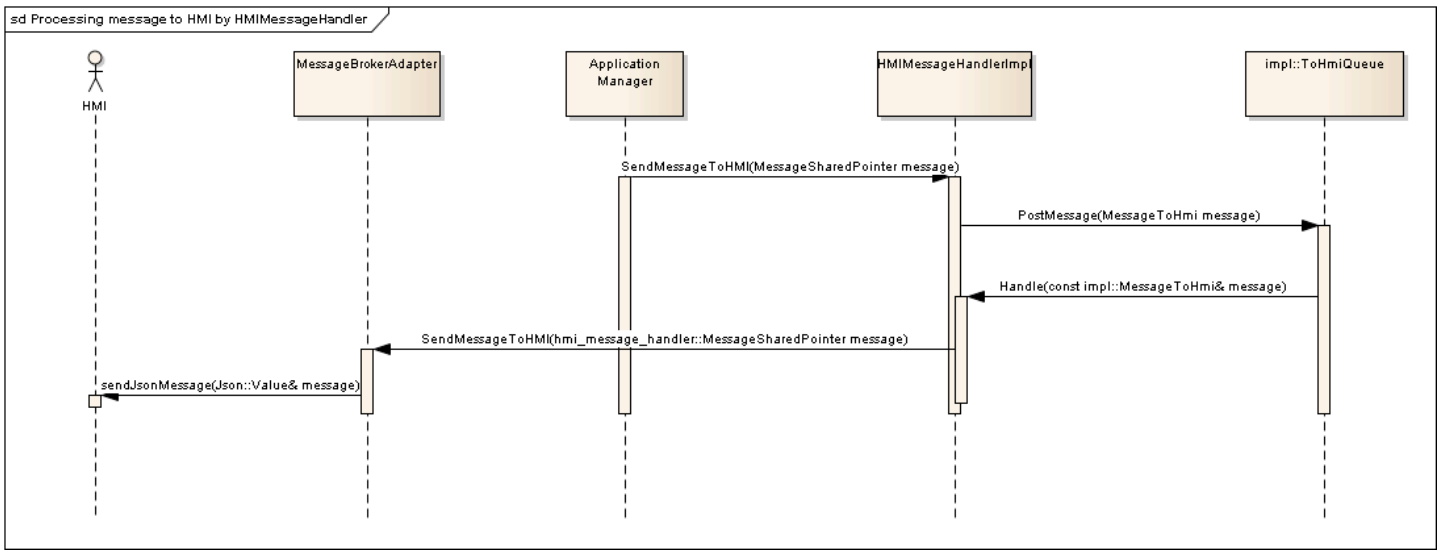


3.1.8. hmi_message_handler

The following diagram show message processing by hmi_message_handler. Message send from HMI

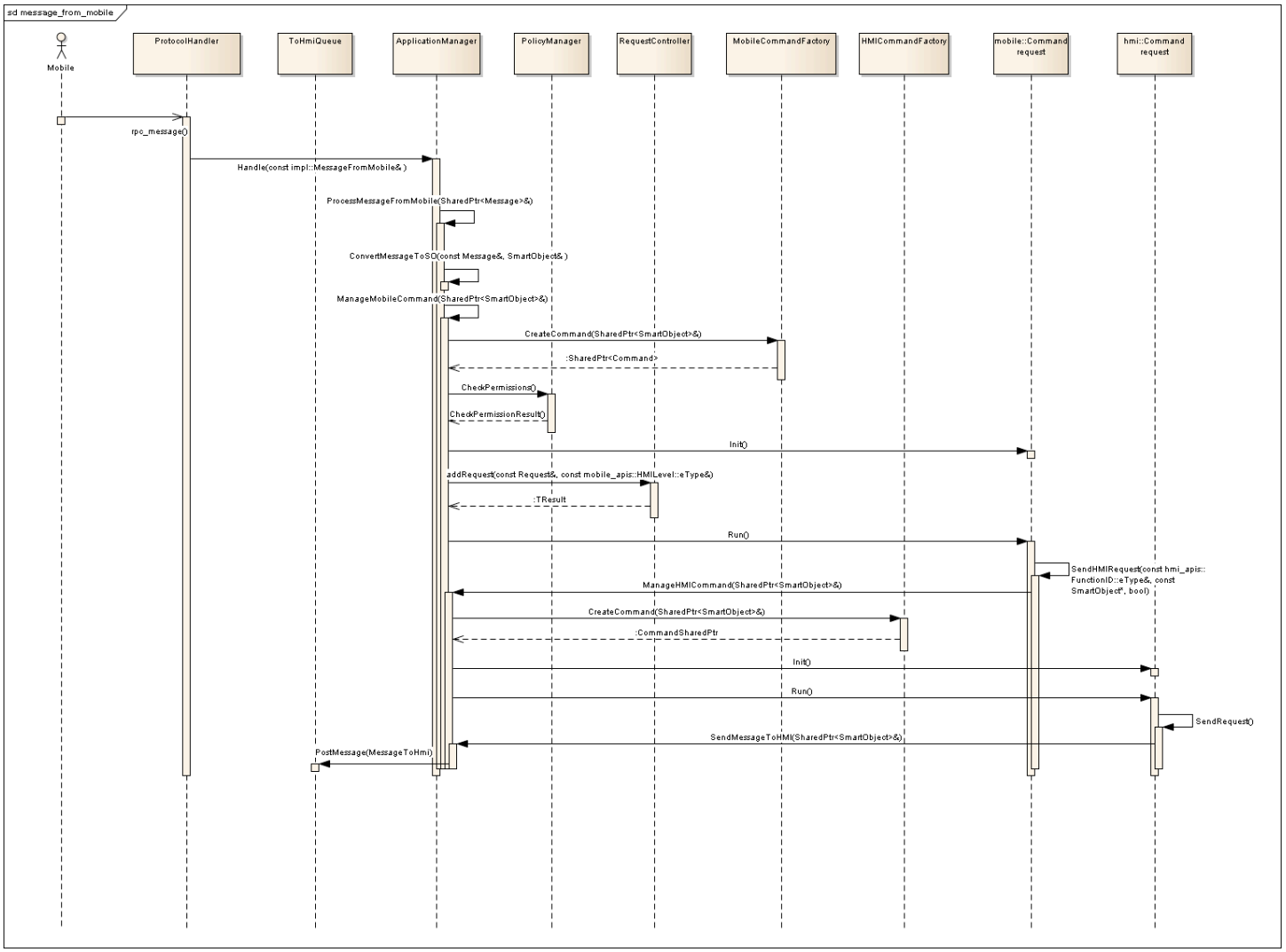


The following diagram show message processing by hmi_message_handler. Message send to HMI

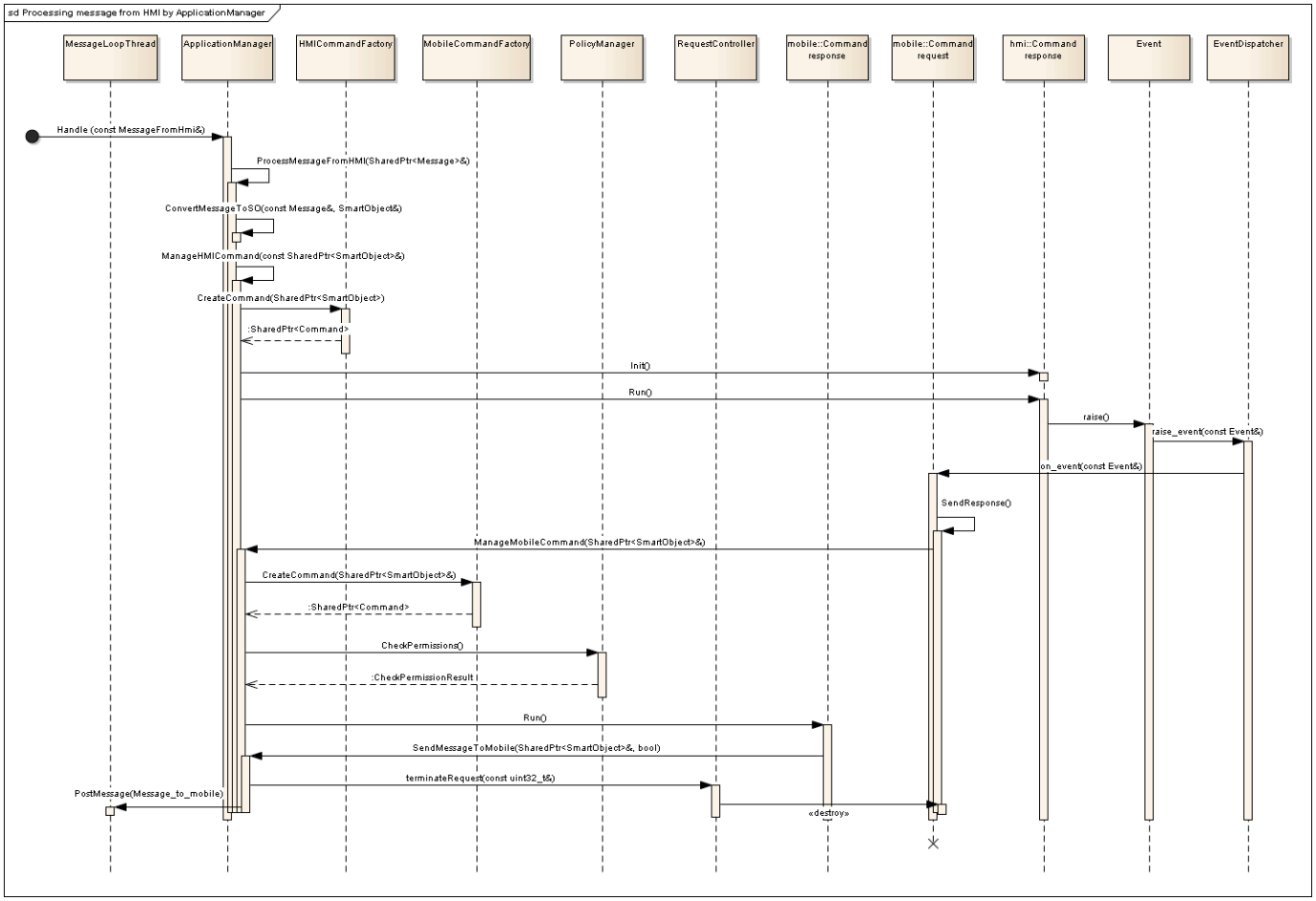


3.1.9. application_manager detailed design

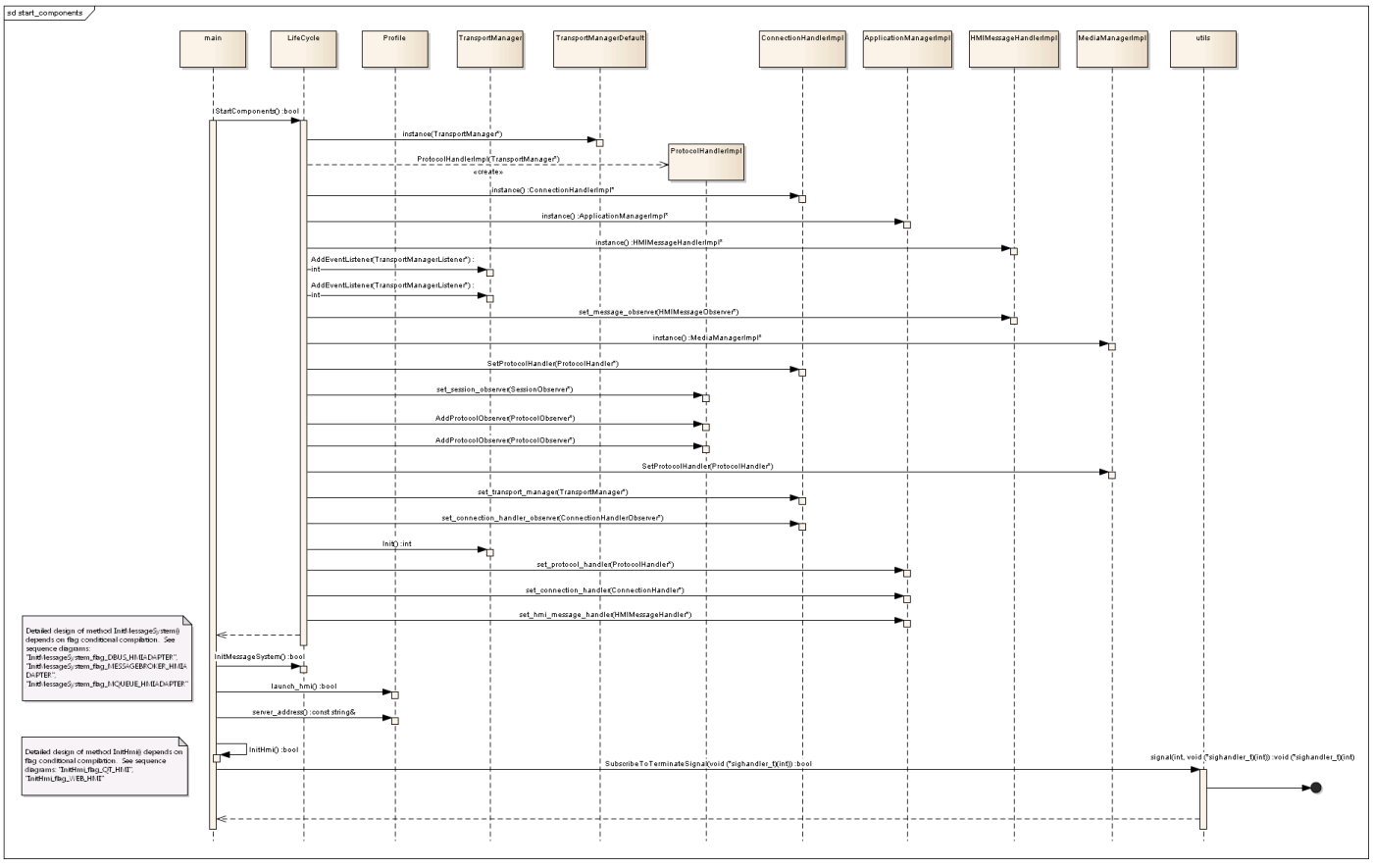
The following diagram shows the RPC message processing from mobile application.



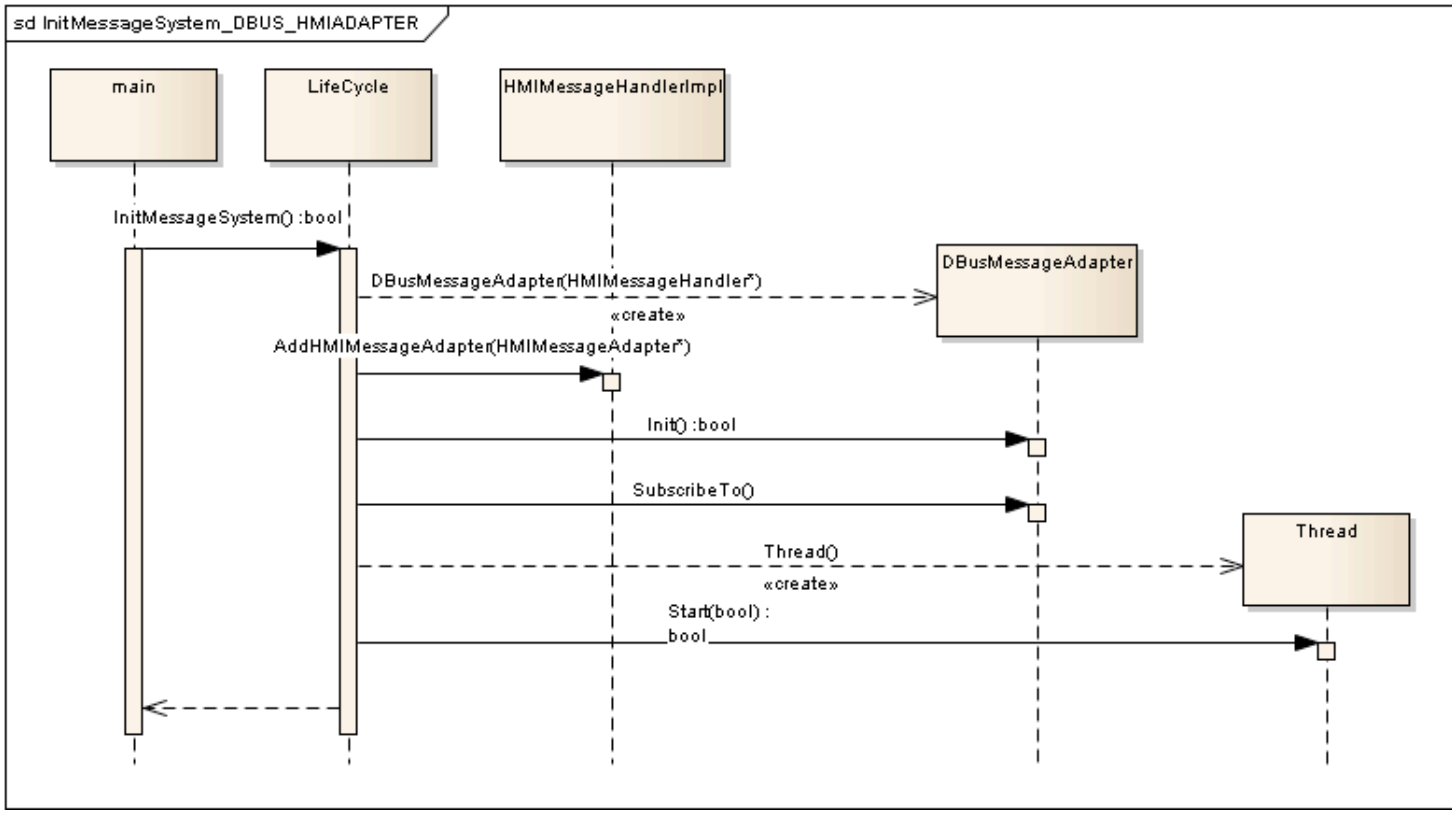
The following diagram shows the RPC message processing from HMI.



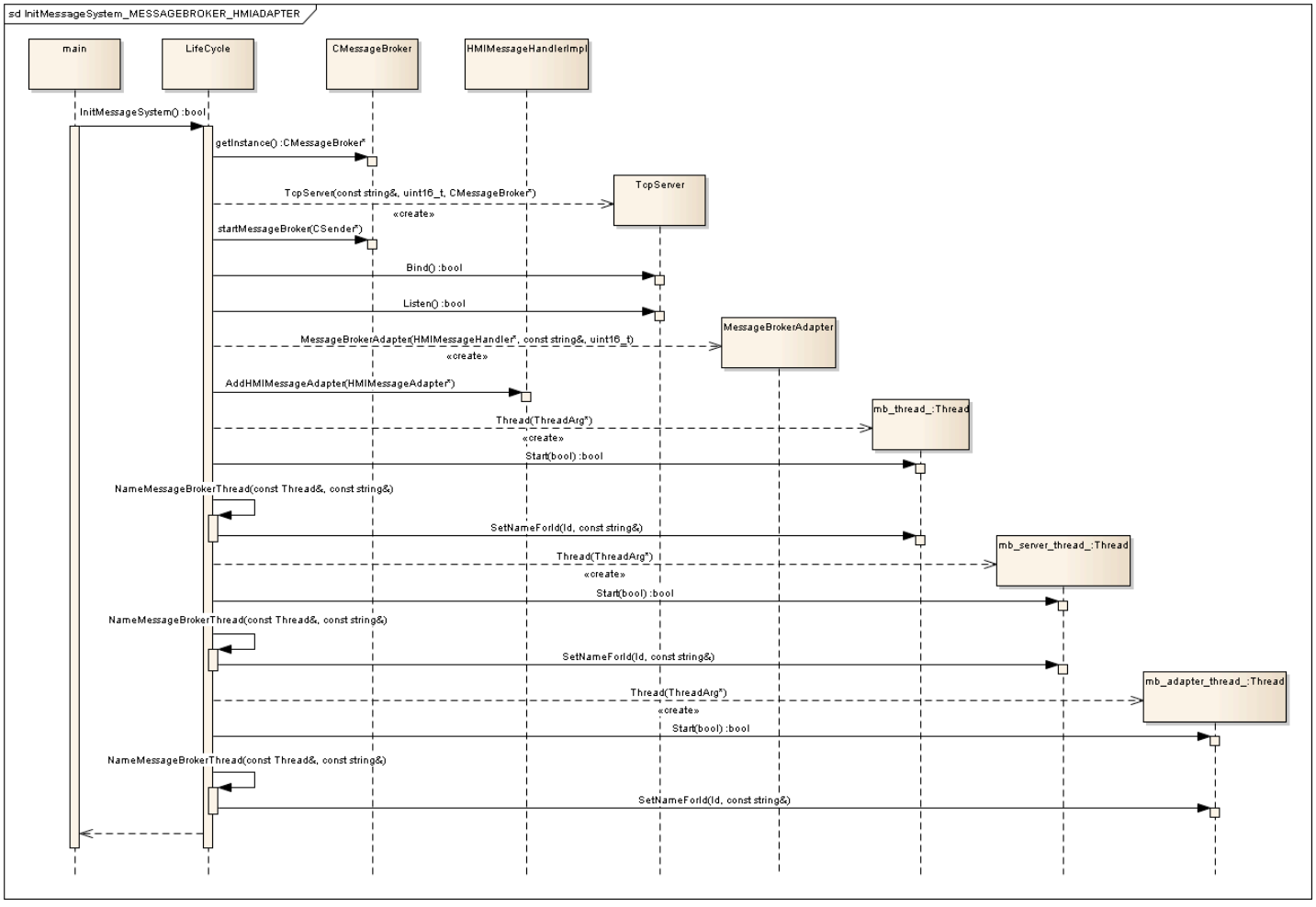
The following diagram shows start components.



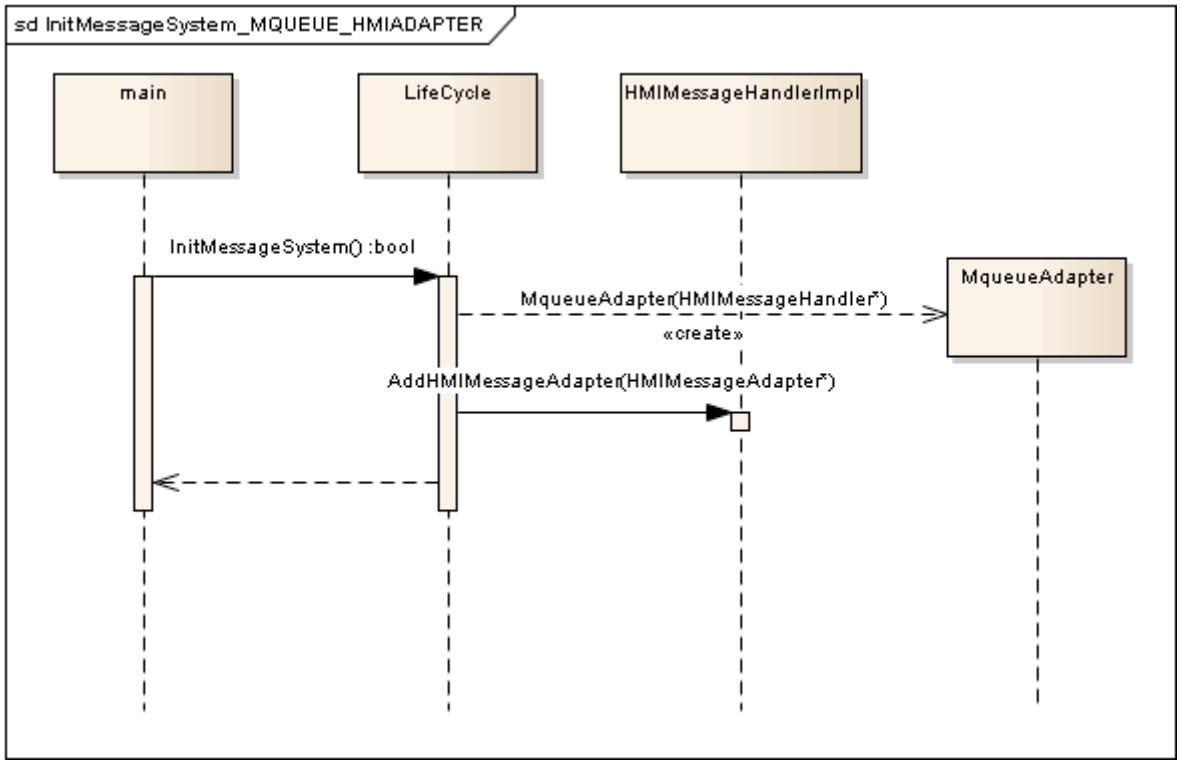
The following diagram shows InitMessageSystem_DBUS_HMIADAPTER.



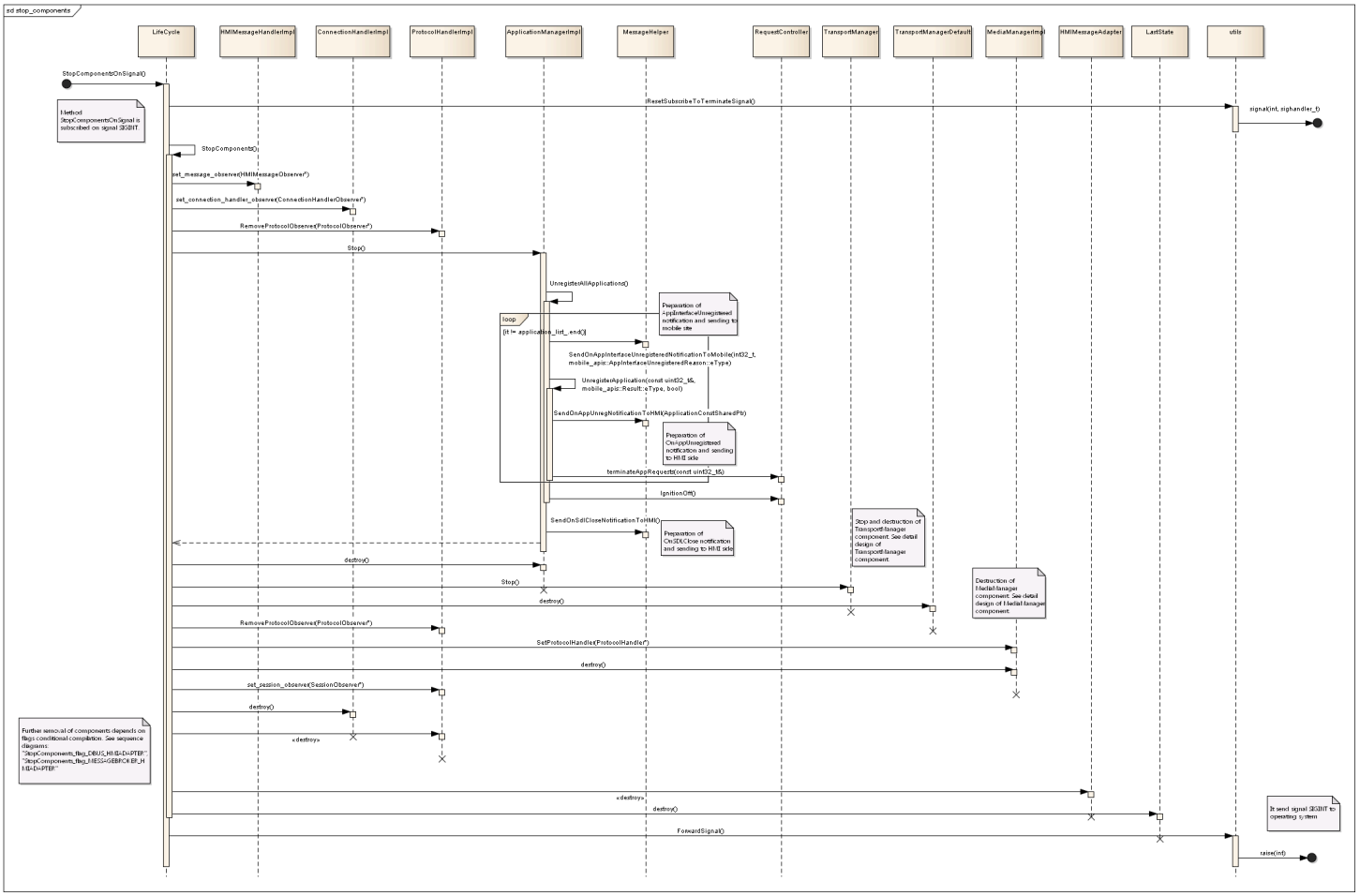
The following diagram shows InitMessageSystem_MESSAGEBROKER_HMIADAPTER.



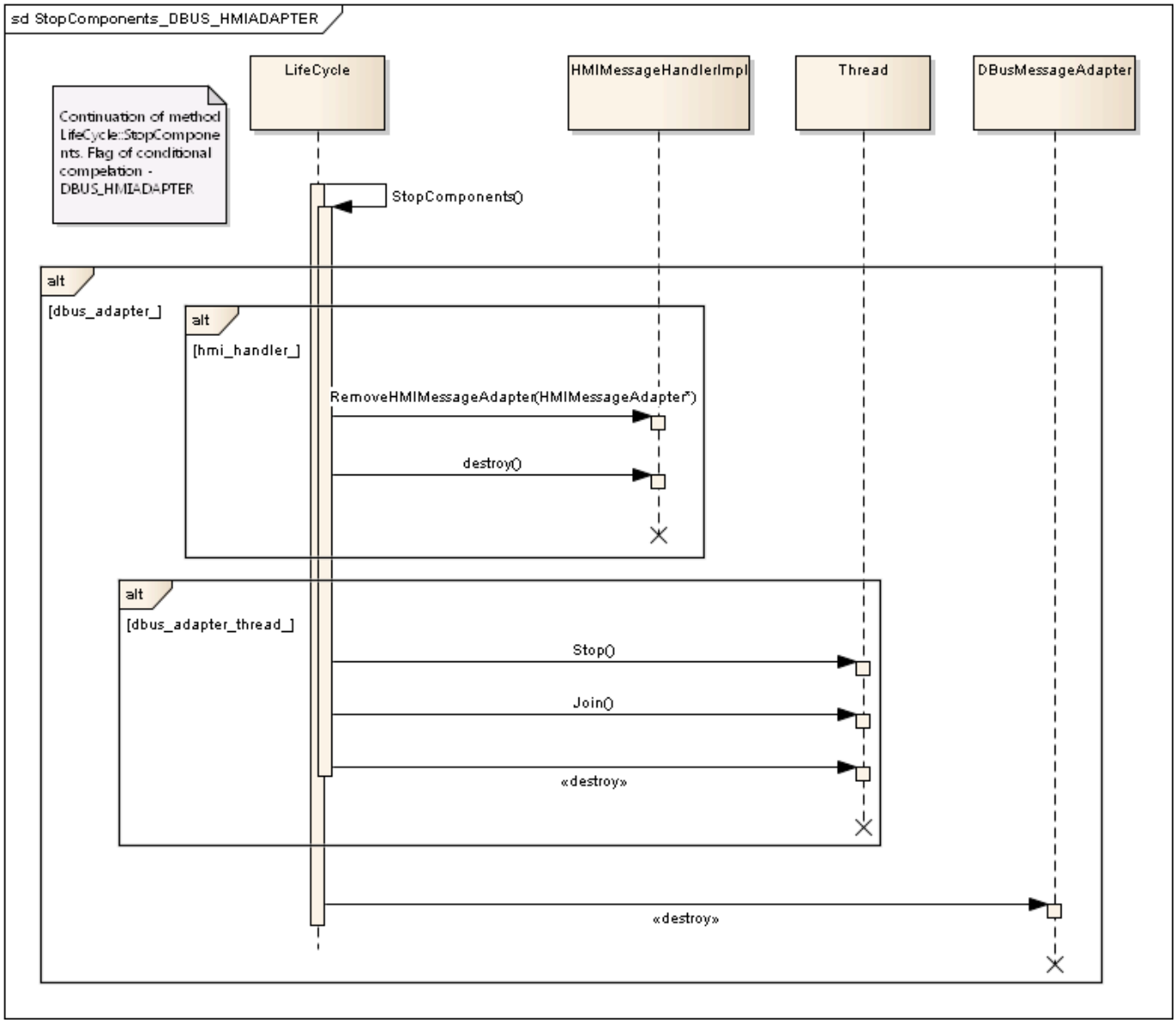
The following diagram shows InitMessageSystem_MQUEUE_HMIADAPTER.



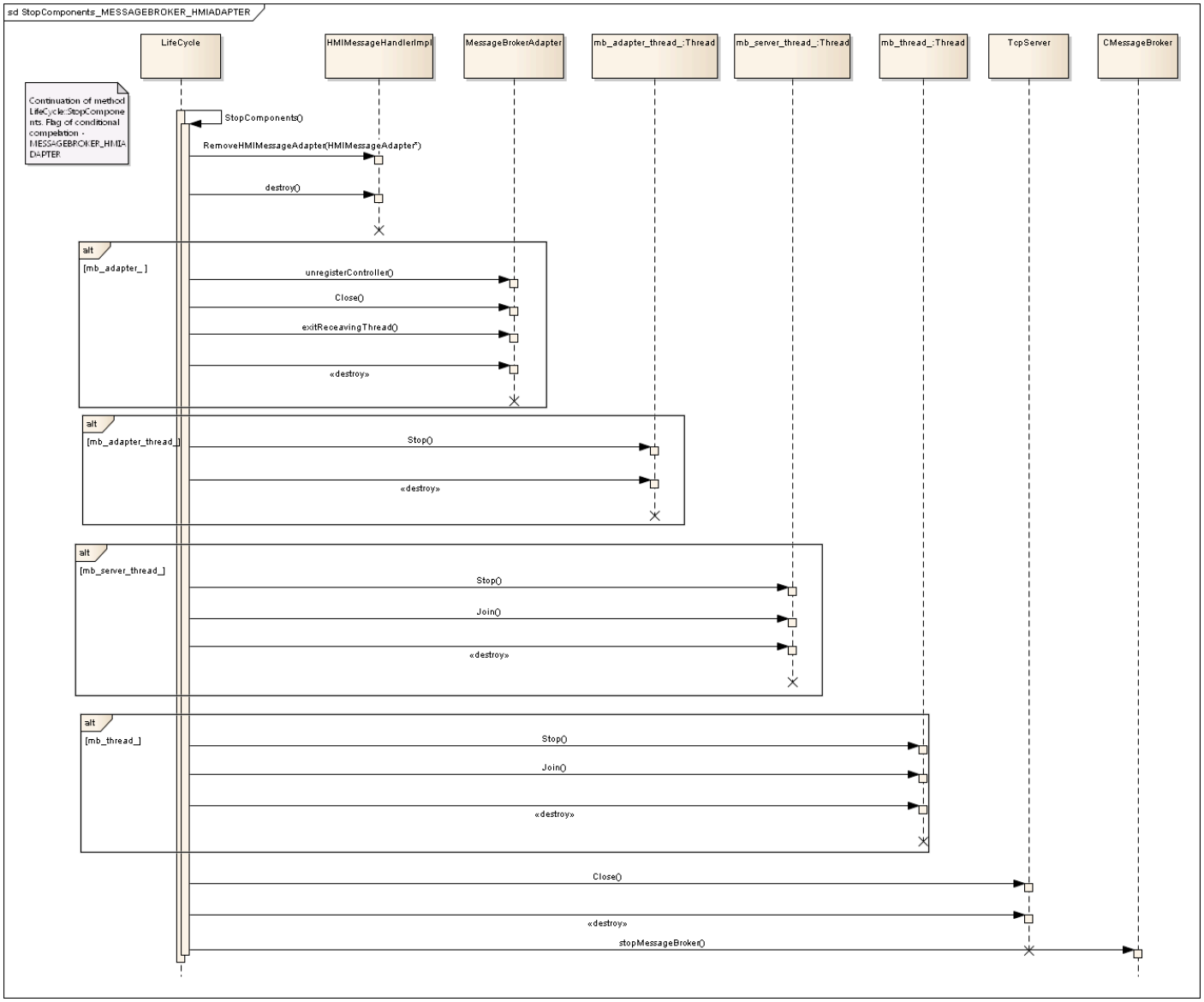
The following diagram shows stop components



The following diagram shows stop components DBUS_HMIADAPTER



The following diagram shows stop components MESSAGEBROKER_HMIADAPTER



3.1.10. Policy detailed design

Diagram for assigning policy permissions on registering of new application:

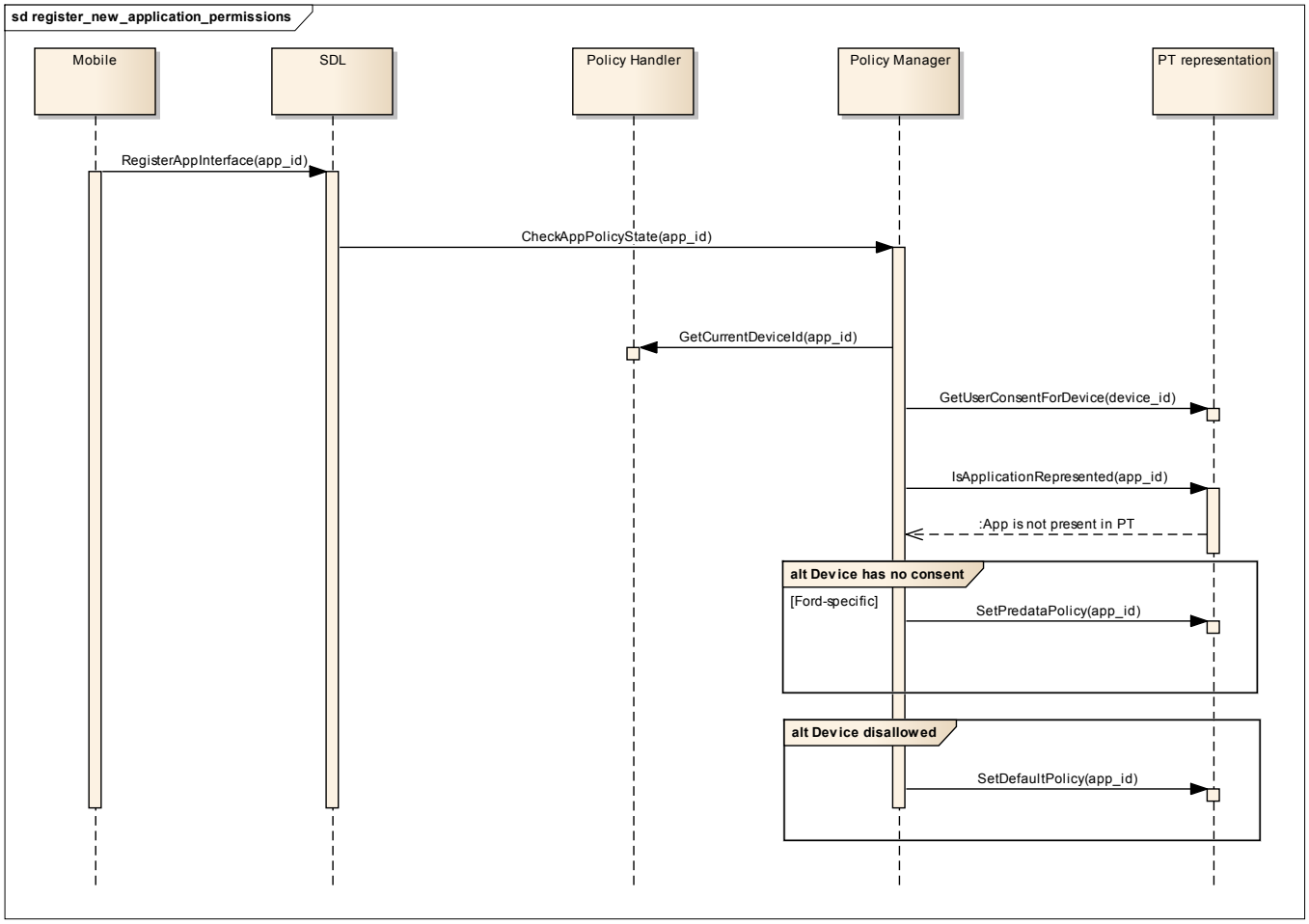


Diagram for **sending** of policy table snapshot to the backend via mobile device (**GENIVI** version):

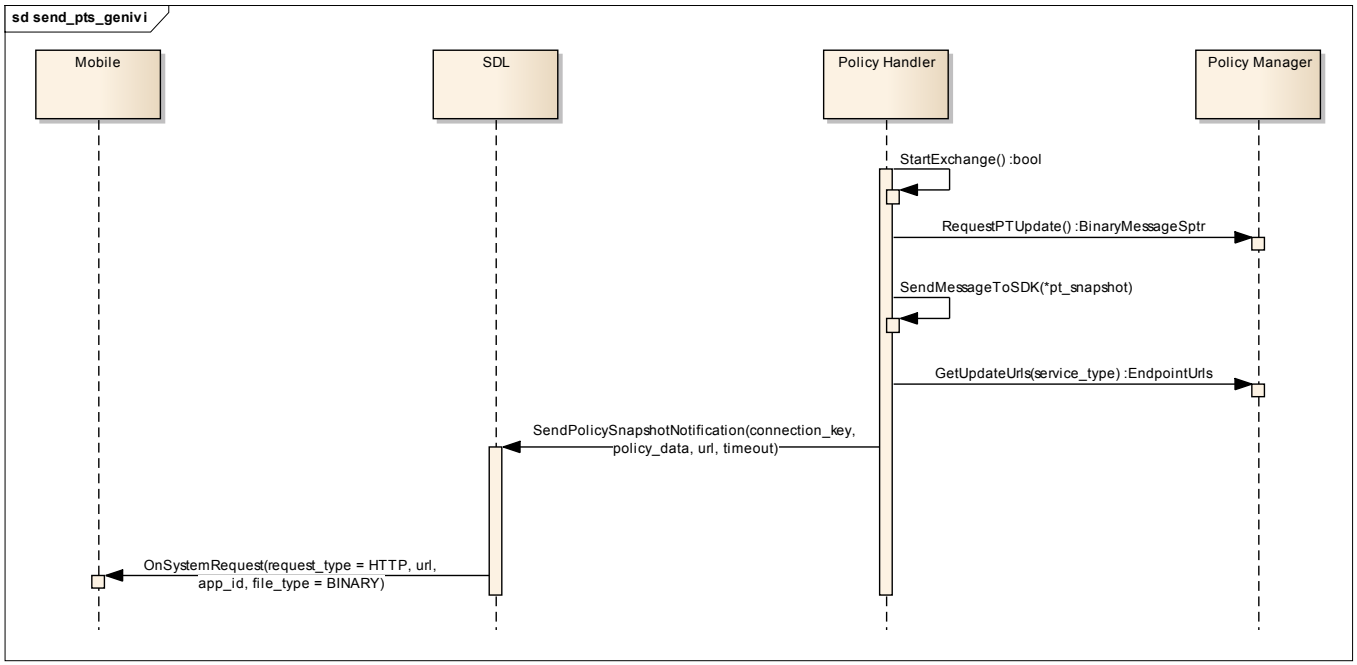


Diagram for **receiving** of policy table update (**GENIVI** version):

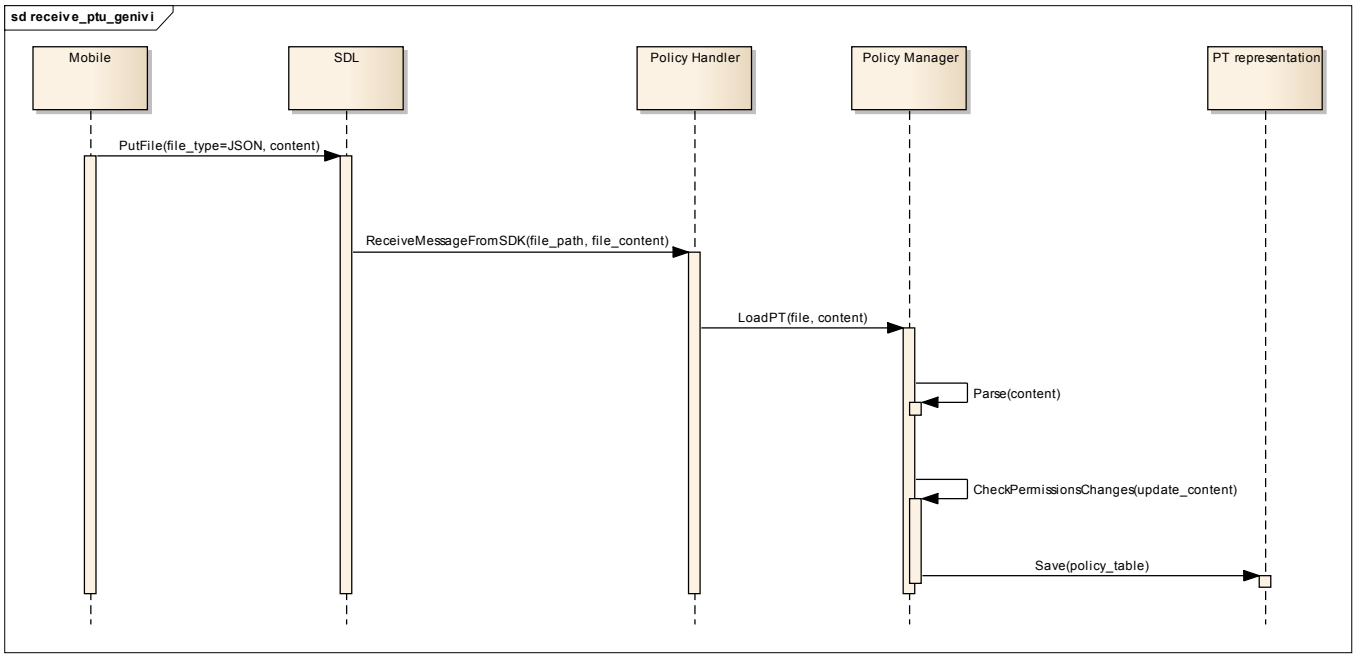


Diagram for **sending** of policy table snapshot to the backend via mobile device (**Ford-specific** version):

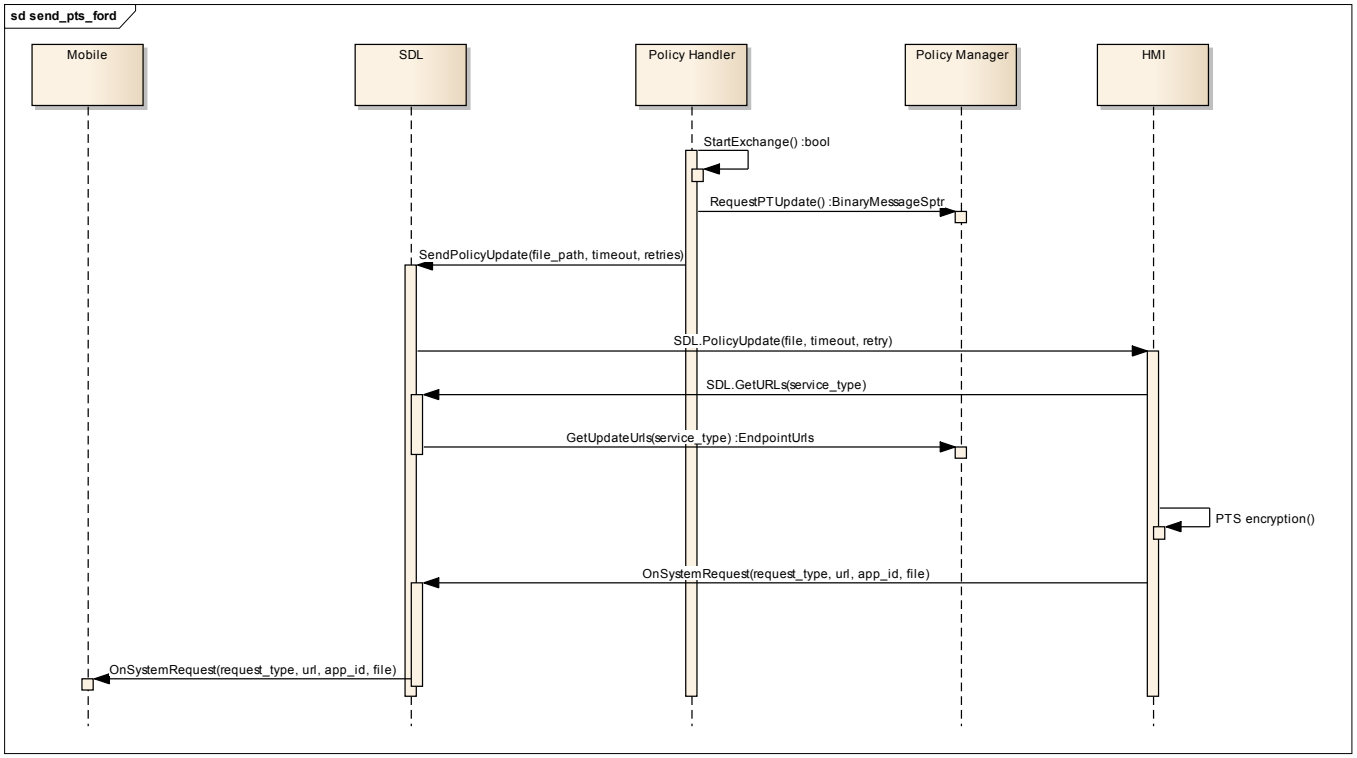
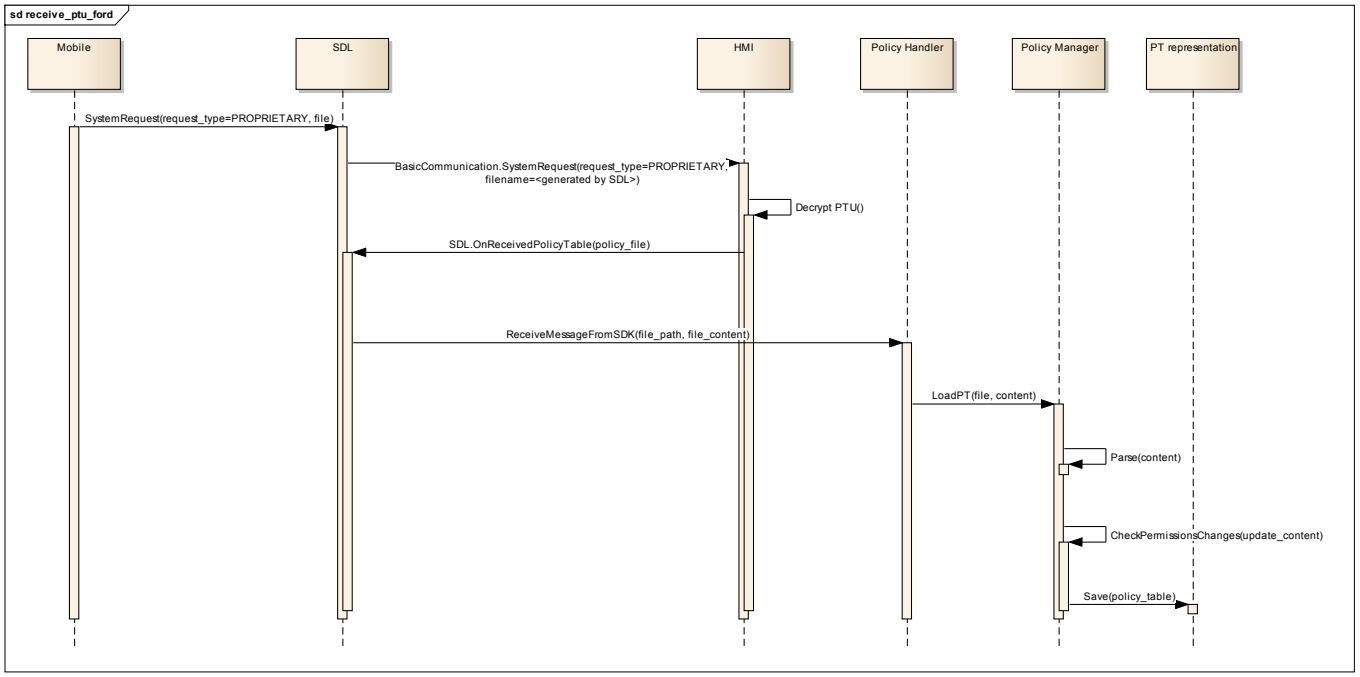


Diagram for **receiving** of policy table update (**Ford-specific** version):



Application selection algorithm described here:
https://adc.luxoft.com/confluence/display/APPLINK/Application+selection+algorithm+for+PTU+request

Application groups check order is here:
https://adc.luxoft.com/confluence/display/APPLINK/App+groups+check+order

Data Detailed Design

3.2. SDL Detailed Design

For detailed information go to following link: SDLAQ-UC-22

4. Dependency Description

4.1. Intermodule dependencies

Interaction mechanisms between the modules of the software are based on using the Observer pattern. In event processing there is a separation between an event publisher object and various event observers that want to be informed of an event occurrence and take their own, presumably different, actions. Current component dependencies please find bellow.

For detailed information go to following link: SDLAQ-UC-13

4.2. Inteprocess dependencies

N/A

4.3. Data dependencies

N/A

Interface Description

This section is divided into the subsections to describe the interface characteristics of the software. It includes both interfaces among the modules and their interfaces with external entities such as systems, configuration items and users.

4.4. Derived interfaces

N/A

4.5. Module interfaces

This section provides the description for the interfaces of the software modules.

4.5.1. Resume Controler description

create – create ResumeCtrl instance. Return pointer to ResumeCtrl.
SaveAllApplications – save applications from application manager to resume controller
SaveApplication – save one application from application manager to resume controller
LoadApplications – Load saved applications from file system
RestoreApplicationHMILevel – restoring application hmi level if application is saved
RestoreApplicationData – restoring application data D2-D7 if application is saved
RemoveApplicationFromSaved – Remove application from list of saved applications
SavetoFileSystem – Save applications info to FileSystem. Used it before ignition off.
StartResumption – trying to start resumption logic for application
StartResumptionOnlyHMILevel – trying to start resumption logic without restoring D2-D7 data.
CheckPersistenceFilesForResumption – check if all data can be processed well. If there are all images exist to restore all D2-D7 data successfully.
CheckApplicationHash – check if application hash is coincides with saved in resume controller.
onTimer – timer callback function, need to restore application HMI level after 3 seconds pause.

4.5.2. ITransportManager description

create – create TransportManager instance. Return pointer to TransportManager.
run – run TransportManager. Must be called on startup, after all references to external components are initialized, to start transport manager.
scanForNewDevices – start scanning for new devices. List of new devices will be provided in onDeviceListUpdated callback.
connectDevice – connect to all applications discovered on device.
disconnectDevice – disconnect from all applications connected on device.
addDataListener – add data listener to the data-related events.
removeDataListener – remove data listener to the data-related events.
addDeviceListener – add device listener to device-related events.
removeDeviceListener – remove device listener to device-related events.
sendFrame – send frame with data to mobile application.

4.5.3. ITransportManagerDataListener description

onFrameReceived – frame received callback.
onFrameSendCompleted – send frame complete callback.

4.5.4. ITransportManagerDeviceListener description

onDeviceListUpdated – available devices list updated callback.
onApplicationConnected – application connected callback.
onApplicationDisconnected – application disconnected callback.

4.5.5. ProtocolObserver description

OnMessageReceived - callback function which is used by ProtocolHandler when new message is received from Mobile Application.

4.5.6. ConnectionHandler description

void set_connection_handler_observer(ConnectionHandlerObserver* observer)
Sets observer pointer for ConnectionHandler.

void set_transport_manager(transport_manager::TransportManager* transport_manager)
Sets pointer to TransportManager.

void ConnectToDevice(connection_handler::DeviceHandle device_handle)
Connects to all services of device.

void ConnectToAllDevices()
Connect to all available device.

void StartTransportManager()
Turns on visibility of SDL to mobile devices. After that mobile devices are able to connect.

void CloseConnection(ConnectionHandle connection_handle)
Close all associated sessions and close the connection pointed by handle.

uint32_t GetConnectionSessionsCount(uint32_t connection_key)
Return count of session for specified connection.

bool GetDeviceID(const std::string& mac_address, DeviceHandle* device_handle)
Gets device id by mac address.

void CloseSession(uint32_t key)
Close session associated with the key.

void CloseSession(ConnectionHandle connection_handle, uint8_t session_id)
Close session by connection and session ids.

void SendHeartBeat(ConnectionHandle connection_handle, uint8_t session_id)
Send heartbeat to mobile app.

void StartSessionHeartBeat(uint32_t connection_key)
Start heartbeat for specified session

4.5.7. SessionObserver description

uint32_t OnSessionStartedCallback(
const transport_manager::ConnectionUID& connection_handle,
const uint8_t session_id,
const ServiceType& service_type)
Callback function used by ProtocolHandler when Mobile Application initiates start of new session.

uint32_t OnSessionEndedCallback(
const transport_manager::ConnectionUID& connection_handle,
const uint8_t session_id,

const uint32_t& hash_code,
const ServiceType& service_type)
Callback function used by ProtocolHandler when Mobile Application initiates session ending.

uint32_t KeyFromPair(
transport_manager::ConnectionUID connection_handle,
uint8_t session_id)
Returns connection identifier and session number from given session key.

void PairFromKey(
uint32_t key,
transport_manager::ConnectionUID* connection_handle,
uint8_t* session_id)
Returns connection identifier and session number from given session key.

int32_t GetDataOnSessionKey (uint32_t key,
uint32_t* app_id,
std::list<int32_t>* sessions_list,
uint32_t* device_id)
Information about given connection key.

int32_t GetDataOnDeviceID(
uint32_t device_handle,
std::string* device_name,
std::list<uint32_t>* applications_list,
std::string* mac_address)
Information about given device handle.

4.5.8. ConnectionHandlerObserver description

void OnDeviceListUpdated(const connection_handler::DeviceMap& device_list)
Available devices list updated callback.

void OnFindNewApplicationsRequest()
Reaction to "Find new applications" request

void RemoveDevice(const connection_handler::DeviceHandle& device_handle)
Called when device has been removed from a list.

bool OnServiceStartedCallback(
const connection_handler::DeviceHandle& device_handle,
const int32_t& session_key,
const protocol_handler::ServiceType& type)
Callback function used by connection_handler when Mobile Application initiates start of new service.

void OnServiceEndedCallback(
const int32_t& session_key,
const protocol_handler::ServiceType& type)
Callback function used by connection_handler when Mobile Application initiates service ending.

4.5.9. MobileMessageObserver description

OnMobileMessageReceived – message received from mobile application callback.

4.5.10. MobileMessageHandler description

SendMessageToMobileApp – send message to mobile application.

AddMobileMessageListener – add listener to message-related events.

RemoveMobileMessageListener – remove listener to message-related events.

4.5.11. ApplicationManager description

ApplicationSharedPtr application(int32_t app_id)*const*

Method finds pointer to the application by application ID in the stored list. Method returns pointer to application. The parameter is application id.

ApplicationSharedPtr application_by_policy_id(const string& policy_app_id)const

Method finds pointer to the application by policy ID in the stored list. Method returns pointer to application. The parameter contains policy id.

const set<ApplicationSharedPtr>& applications()const

Method returns stored list of pointer to application.

ApplicationSharedPtr active_application()const

Method returns pointer to application which are in full-screen mode on HMI.

vector<ApplicationSharedPtr> applications_by_button(uint32_t button)

Method finds pointer to the application by button ID and generates a list of these pointers. Method returns list of pointer to applications which are subscribed on button event. The parameter is button id.

vector<ApplicationSha*redPtr*> applications_with_navi()

Method finds pointer to the application which supports navigation and generates a list of these pointers. Method returns this list.

vector<SharedPtr<Application>> IviInfoUpdated(VehicleDataType vehicle_info, int value)

Method notifies all components interested in Vehicle Data update. Method generates a list of pointers applications which contain a specific vehicle info type. The parameter vehicle_info contains vehicle info type and is used in order to generate list.

HMICapabilities& hmi_capabilities ()

Returns a reference to the object HMICapabilities which store in application_manager.

SetTimeMetricObserver(AMMetricObserver* observer)

Setup observer for time metric. Parameter observer is - pointer to observer

ApplicationSharedPtr RegisterApplication(const SharedPtr<smart_objects::SmartObject>& request_for_registration)

Method creates application, fills information in the application, adds application to list of applications and returns application is registered. Parameter request_for_registration contains the data received during request register_app_interface from mobile side.

UnregisterApplication(const uint32_t& app_id,mobile_apis::Result::eType reason, bool is_resuming = false)

Method finds application by app id and closes this application. After that method removes application from list of applications and calls “resume controller” in order to save application data. If audio passthru process is active method finishes it. Parameter “app_id” is used for search pointer to application from application list. Parameter “reason” contains reason of reboot HU. If parameter “is_resuming” = TRUE method calls “resume controller”.

SetUnregisterAllApplicationsReason(mobile_api::AppInterfaceUnregisteredReason::eType reason)

Method sets unregister reason for closing all registered applications during HU switching off. Parameter “reason” contains reason of reboot HU.

HeadUnitReset(mobile_api::AppInterfaceUnregisteredReason::eType reason)

Called on Master_reset or Factory_defaults when User chooses to reset HU. Resets Policy Table if applicable. Parameter “reason” contains reason of reboot HU.

UnregisterAllApplications()

Method sends data for preparation “OnAppInterfaceUnregistered” notification for mobile side. Method calls UnregisterApplication.

bool ActivateApplication(ApplicationSharedPtr app)

Method activates application, if success returns TRUE, if fail returns FALSE. Parameter contains pointer to application which is activated by application_manager.

mobile_api::HMILevel::eType PutApplicationInLimited(ApplicationSharedPtr app)

Method puts application in Limited HMI Level if possible, otherwise put application other HMI level. Does not send any notifications to mobile side. Parameter app, application, that need to be put in LIMITED. Returns set HMI Level.

mobile_api::HMILevel::eType PutApplicationInFull(ApplicationSharedPtr app)

Method puts application in FULL HMI Level if possible, otherwise put application other HMI level. Does not send any notifications to mobile side. Parameter app, application, that need to be put in LIMITED. Returns set HMI Level.

DeactivateApplication(ApplicationSharedPtr app)

Method sends data for preparation SetGlobalProperties request for HMI side and preparation OnAppInterfaceUnregistered notification for mobile side. Parameter app contains pointer to application for which are sent prepared messages.

ConnectToDevice(uint32_t id)

Method sends device id to connection handler component. Parameter id contains device id.

OnHMIStartedCooperation()

Method sends to HMI requests: “VR_IsReady”, “TTS_IsReady”, “UI_IsReady”, “Navigation_IsReady”, “VehicleInfo_IsReady”, “Buttons_GetCapabilities”. Method is called when core receives “Ready” notification from HMI.

uint32_t GetNextHMICorrelationID()

Method returns unique correlation ID for HMI request.

bool begin_audio_pass_thru()

Method saves state of audio pass thru. If audio pass thru is active method returns FALSE otherwise method changes state and returns TRUE.

bool end_audio_pass_thru()

Method saves state of audio pass thru. If audio pass thru is active method changes state and returns TRUE, otherwise method returns FALSE.

bool driver_distraction()const

Method retrieves driver distraction state.

set_driver_distraction (bool is_distracting)

Method sets state for driver distraction. Parameter “is_distracting” contains state.

bool vr_session_started()

Method retrieves state of VR component on HU.

set_vr_session_started(const bool& state)

Method saves state of VR component on HU. Parameter “state” contains state of VR component.

bool all_apps_allowed()

Method retrieves SDL access to all mobile apps.Method returns currently active state of the access.

set_all_apps_allowed(const bool& allowed)

Method saves state of SDL access to all mobile apps. Parameter “allowed” contains this state.

StartAudioPassThruThread(int32_t session_key, int32_t correlation_id, int32_t max_duration, int32_t sampling_rate, int32_t bits_per_sample, int32_t audio_type)

Method starts audio pass thru thread and activates of recording from microphone(if this opportunity is supported).
Parameters: “session_key” session key of connection for mobile side;
“correlation_id” correlation id for response for mobile side;
“max_duration” max duration of audio recording in milliseconds;
“sapling_rate” value for rate(8, 16, 22, 44 kHz);
“bitsh_per_sample” the quality the audio is recorded;
“audio_type” type of audio data.

StopAudioPassThru(int32_t application_key)

Method terminates audio pass thru thread and stops of recording from microphone(if this opportunity is supported).
Parameter “application_key” is Id of application for which audio pass thru should be stopped.

SendAudioPassThroughNotification(uint32_t session_key, vector<uint8_t> binaryData)

Method prepares “OnAudioPassThru” notification to mobile side. Notification contains binary data from microphone.

string GetDeviceName(DeviceHandle handle)

Method receives device name by device id from ConnectionHandler component. Parameter “handle” contains device id. Method returns device name.

set_hmi_message_handler(HMIMessageHandler* handler)

Methods sets pointer to component HMIMessageHandler. Parameter “handler” contains this pointer.

set_connection_handler(ConnectionHandler* handler)

Methods sets pointer to component ConnectionHandler. Parameter “handler” contains this pointer.

set_protocol_handler(ProtocolHandler* handler)

Methods sets pointer to component ProtocolHandler. Parameter “handler” contains this pointer.

StartDevicesDiscovery()

Methods call “StartDevicesDiscovery” from component ConnectionHandlerImpl.

SendMessageToMobile(const SharedPtr<SmartObject>& message, bool final_message = false)

Methods put message to the queue to be sent to mobile. Parameter “message” contains data for sending to mobile side. If “final_message” parameter = true connection to mobile will be closed after processing this message.

ManageMobileCommand(const SharedPtr<SmartObject>& message)

Methods processes message from or to mobile side, creates mobile command. Command is verified by PolicyManager. Command is executed if the check is completed successfully. Otherwise method creates BlockedByPolicies response and sends to mobile side. Parameter “message” contains sent (or received) data to(or from) mobile side.

SendMessageToHMI(const SharedPtr<SmartObject>& message)

Methods puts message to the queue to be sent to HMI. Parameter “message” contains data for sending to HMI side.

ManageHMICommand(const SharedPtr<SmartObject>& message)

Methods processes message from or to HMI side, creates HMI command. Parameter “message” contains sent (or received) data to(or from) HMI side.

OnMessageReceived(const RawMessagePtr& message)

Method is overridden of ProtocolObserver. Converts raw message to message and puts message to the “message from mobile” queue. Method is called if message has received from mobile side. Parameter message contains data from mobile side.

OnMobileMessageSent(const RawMessagePtr& message)

Method is overridden of ProtocolObserver. Method is called if message has sent to mobile side.
Parameter message contains data to mobile side.

OnMessageReceived(MessageSharedPointer message)

Method is overridden of HMIMessageObserver. Method is called if message has received from HMI. Puts message to the queue. Parameter message contains data from HMI.

OnErrorSending(MessageSharedPointer message)

Method is overriden of HMIMessageObserver.

OnDeviceListUpdated(const DeviceList& device_list)

Method is overridden of ConnectionHandlerObserver. Method is called when device list was updated. Method prepares UpdateDeviceList request to HMI. Parameter device_list contains updated information about devices.

RemoveDevice(const DeviceHandle& device_handle)

Method is overridden of ConnectionHandlerObserver. Method is called when transportManager send “OnDeviceRemoved” notification to ConnectionHandler. Parameter “device_handle” contains device id.

bool OnServiceStartedCallback(constDeviceHandle& device_handle, const int32_t& session_key, const ServiceType& type)

Method is overridden of ConnectionHandlerObserver. Method starts video or audio streaming depending on service type. Parameter “device_handle” contains device id. If the method worked correctly returns TRUE otherwise returns FALSE.

OnServiceEndedCallback(const int32_t& session_key, const ServiceType& type)

Method is overridden of ConnectionHandlerObserver. If parameter “type” contains kMobileNav or kAudio method stops video or audio streaming respectively. If parameter “type” contains kRpc method calls UnregisterApplication.

addNotification(const CommandSharedPtr& ptr)

Method adds notification to collection. Parameter “ptr” reference to shared pointer that point on hmi notification

removeNotification(const CommandSharedPtr& ptr)

Method removes notification from collection. Parameter “ptr” reference to shared pointer that point on hmi notification.

updateRequestTimeout(uint32_t connection_key, uint32_t mobile_correlation_id, uint32_t new_timeout_value)

Method updates request timeout. Parameter “connection_key” is connection key of application.Parameter “mobile_correlation_id” is correlation ID of the mobile request. Parameter “new_timeout_value” is new timeout to be set.

application_id(const int32_t correlation_id)

Method finds application id associated whith correlation id. Parameter “correlation_id” is correlation ID of the mobile request. Method returns application id.

set_application_id(const int32_t correlation_id, const uint32_t app_id)

Method saves application id associated with correlation id. Parameter “correlation_id” is correlation ID of the mobile request. Parameter “app_id” is ID of registered application.

Mute(VRTTSSessionChanging changing_state)

Method changes AudioStreamingState for all application according to system audio-mixing capabilities (NOT_AUDIBLE/ATTENUATED) and send notification for this changes. If parameter “changing_state” =kVRSessionChanging function is used by on_vr_started_notification, if “changing_state”=kTTSSessionChanging function is used by on_tts_started_notification.

Unmute(VRTTSSessionChanging changing_state)

Method changes AudioStreamingState for all application to AUDIBLE and send notification for this changes. If parameter “changing_state” =kVRSessionChanging function is used by on_vr_started_notification, if “changing_state”=kTTSSessionChanging function is used by on_tts_started_notification.

bool IsAudioStreamingAllowed(uint32_t connection_key) const

Method checks HMI level. Parameter “connection_key” is connection key of application. Returns TRUE if HMI level = HMI_FULL or HMI_LIMITED.

bool IsVideoStreamingAllowed(uint32_t connection_key) const

Method checks HMI level and video streaming is allowed. Parameter “connection_key” is connection key of application. Returns TRUE if HMI level = HMI_FULL and video streaming is allowed.

ResumeCtrl& resume_controller()

Method returns component “Resume Controller”.
uint32_t GenerateGrammarID()
Method returns a randomly generated grammar ID.
uint32_t GenerateNewHMIAppID()
Method generates new HMI application ID and return it.
ReplaceMobileByHMIAppId(SmartObject& message)
Method parses parameter “message” and replace mobile app Id by HMI app ID. Parameter “message” contains data to HMI.
ReplaceHMIByMobileAppId(SmartObject& message) –
Method parses parameter “message” and replace HMI app ID by mobile app Id. Parameter “message” contains data from HMI.
mobile_apis::Result::eType SaveBinary(const vector<uint8_t>& binary_data, const string& file_path, const string& file_name, const uint32_t offset)
Method saves binary data to specified directory.
Parameter “binary data” contains data for saving in directory
Parameter “path” contains path for saving data.
Parameter “file_name” contains name of file.
Parameter “offset” for saving data to existing file with offset. If offset is 0 - create new file (overrite existing)
Returns SUCCESS if file was saved, other code otherwise.
uint32_t GetAvailableSpaceForApp(const string& name)
Method calculates how much space is available for data of application. Parameter “name” is name of application.

4.5.11. HMICapabilities description

bool is_hmi_capabilities_initialized()const
Method checks if all HMI capabilities received. Return TRUE if all information received, otherwise FALSE.
bool VerifyImageType(int32_t image_type)
Method checks is image type(Static/Dynamic) requested by Mobile Device is supported on current HMI.
Parameters “image_type” recieved type of image from Enum.Returns TRUE if supported otherwise returns FALSE.
bool is_vr_cooperating()const
Method returns TRUE if VR component is available on HMI, otherwise FALSE.
set_is_vr_cooperating(bool value)
Method sets attribute “is_vr_cooperating”. If parameter “value” = TRUE sends to HMI “VR_GetLanguage”, “VR_GetSupportedLanguages”, “VR_GetCapabilities” requests. Parameter “value” is TRUE if VR component is available on HMI otherwise FALSE.
bool is_tts_cooperating()const
Method returns TRUE if TTS component is available on HMI, otherwise FALSE.
set_is_tts_cooperating(bool value)
Method sets member “is_tts_cooperating”. If parameter “value” = TRUE sends to HMI “TTS_GetLanguage”, “TTS_GetSupportedLanguages”, “TTS_GetCapabilities” requests. Parameter “value” is TRUE if TTS component is available on HMI otherwise FALSE.
bool is_ui_cooperating()const
Method returns TRUE if UI component is available on HMI, otherwise FALSE.
set_is_ui_cooperating(bool value)
Methods sets member “is_ui_cooperating”. If value = TRUE sends to HMI “UI_GetLanguage”, “UI_GetSupportedLanguages”, “UI_GetCapabilities” requests. Parameter “value” is TRUE if UI component is available on HMI otherwise FALSE.
bool is_navi_cooperating()const
Method returns TRUE if NAVI component is available on HMI, otherwise FALSE.
set_is_navi_cooperating(bool value)
Method sets member “is_navi _cooperating”. Parameter “value” is TRUE if NAVI component is available on HMI otherwise FALSE.
bool is_ivi_cooperating()const
Method returns TRUE if IVI component is available on HMI, otherwise FALSE.
set_is_ivi_cooperating(bool value)
Method sets member “is_ivi_cooperating”. If is_ivi_cooperating = TRUE sends to HMI VehicleInfo_GetVehicleType request. Parameter “value” is TRUE if IVI component is available on HMI otherwise FALSE.
bool attenuated_supported()
Method retrieves if mixing audio is supported by HMI (ie recording TTS command and playing audio). Returns current state of the mixing audio flag.
set_attenuated_supported(bool state)
Method sets state for mixing audio. Parameter “state” is received from “MixingAudioSupported” response from HMI
const hmi_apis::Common_Language::eType& active_ui_language()const
Method retrieves currently active UI language.
set_active_ui_language(const hmi_apis::Common_Language::eType& language)

Method sets current active UI language. Parameter “language” received from “OnUILanguageChange” notification or from “UIGetLanguage” response from HMI. If UI component is not available on HMI, parameter UI language is loaded from hmi_capabilities.json.

SmartObject ui_supported_languages()const

Method retrieves UI supported languages.

set_ui_supported_languages(const SmartObject& supported_languages)

Method sets supported UI languages. Parameter “supported_languages” is received from UIGetSupportedLanguages response from HMI. If UI component is not available on HMI, parameter UI supported languages is loaded from hmi_capabilities.json.

hmi_apis::Common_Language::eType& active_vr_language()const

Method retrieves current active VR language.

set_active_vr_language(const hmi_apis::Common_Language::eType& language)

Method sets current active VR language. Parameter “language” is received from TTSLanguageChange notification or VRLanguageChange notification, or VRGetLanguage response from HMI . If VR component is not available on HMI, parameter VR language is loaded from hmi_capabilities.json.

const smart_objects::SmartObject* vr_supported_languages()const

Method retrieves VR supported languages.

set_vr_supported_languages(const SmartObject& supported_languages)

Method sets supported VR languages. Parameter “supported_languages” is received from VRGetSupportedLanguages response from HMI. If VR component is not available on HMI, parameter VR supported languages is loaded from hmi_capabilities.json.

const hmi_apis::Common_Language::eType& active_tts_language()const

Method retrieves current active TTS language.

set_active_tts_language(const hmi_apis::Common_Language::eType& language)

Method sets current active TTS language. Parameter “language” is received from OnTTSLanguageChange notification or TTSGetLanguage response from HMI. If TTS component is not available on HMI, parameter language is loaded from hmi_capabilities.json.

const SmartObject* tts_supported_languages()const

Method retrieves TTS supported languages.

set_tts_supported_languages(const SmartObject& supported_languages)

Method sets supported TTS languages. Parameter “supported_languages” is received from TTSGetSupportedLanguages response from HMI. If TTS component is not available on HMI, parameter supported languages is loaded from hmi_capabilities.json.

const SmartObject* display_capabilities()const

Method retrieves information about the display capabilities.

set_display_capabilities(const SmartObject& display_capabilities)

Method sets supported display capabilities. Parameter “display_capabilities” is received from UIGetCapabilities response or UISetDisplayLayout response from HMI. If UI component is not available on HMI, parameter display capabilities is loaded from hmi_capabilities.json.

const SmartObject* hmi_zone_capabilities()const

Method retrieves information about the HMI zone capabilities.

set_hmi_zone_capabilities(const SmartObject& hmi_zone_capabilities)

Method sets supported HMI zone capabilities. Parameter “hmi_zone_capabilities”” is received from UIGetCapabilities response from HMI. If UI component is not available on HMI, parameter hmi zone capabilities is loaded from hmi_capabilities.json.

const SmartObject* soft_button_capabilities()const

Method retrieves information about the SoftButton's capabilities.

set_soft_button_capabilities(const SmartObject& soft_button_capabilities)

Method sets supported SoftButton's capabilities. Parameter “soft_button_capabilities” is received from UIGetCapabilities response or UISetDisplayLayout response from HMI. If UI component is not available on HMI, parameter softbutton_capabilities is loaded from hmi_capabilities.json.

const SmartObject* button_capabilities()const

Method retrieves information about the Button's capabilities.

set_button_capabilities(const smart_objects::SmartObject& button_capabilities)

Method sets supported Button's capabilities. Parameter “soft_button_capabilities” is received from ButtonGetCapabilities response from HMI or is loaded from hmi_capabilities.json.

set_speech_capabilities(const SmartObject& speech_capabilities)

Method sets supported speech capabilities. Parameter “speech_capabilities” is received from TTSGetCapabilities response from HMI. If TTS component is not available on HMI, parameter speech_capabilities is loaded from hmi_capabilities.json.

const SmartObject* speech_capabilities()const

Method retrieves information about the speech capabilities.

set_vr_capabilities(const SmartObject& vr_capabilities**)**

Method sets VR capabilities. Parameter “vr_capabilities” is received from VRGetCapabilities from HMI. If VR component is not available on HMI, parameter vr capabilities is loaded from hmi_capabilities.json.

const smart SmartObject* vr_capabilities()

Method retrieves information about the VR capabilities.

set_audio_pass_thru_capabilities(const SmartObject& audio_pass_thru_capabilities)

Method sets supported audio_pass_thru capabilities. Parameter audio_pass_thru_capabilities is received from UIGetCapabilities response. If UI component is not available on HMI, parameter audio_pass_thru_capabilities is loaded from hmi_capabilities.json.
const SmartObject* audio_pass_thru_capabilities()const
Method retrieves information about the audio_pass_thru capabilities.
const SmartObject preset_bank_capabilities()const
Method retrieves information about the preset bank capabilities.
set_preset_bank_capabilities(const SmartObject& preset_bank_capabilities)
Method sets supported preset bank capabilities. Parameter “preset_bank_capabilities” is received from ButtonGetCapabilities response from HMI or is loaded from hmi_capabilities.json.
set_vehicle_type(const SmartObject& vehicle_type)
Method sets vehicle information(make, model, modelYear). Parameter “vehicle_type” is received from VIGetVehicleType response from HMI If VI component is not available on HMI, parameter “vehicle_type” is loaded from hmi_capabilities.json.
const SmartObject* vehicle_type()const
Method retrieves vehicle information(make, model, modelYear).
const SmartObject* prerecorded_speech()const
Method retrieves information about the prerecorded speech.
set_prerecorded_speech(const SmartObject& prerecorded_speech)
Method sets supported prerecorded speech. Parameter “prerecorded_speech” is received from TTSGetCapabilities response from HMI.

4.5.12. Application description

const smart_objects::SmartObject* active_message()const
Method returns message belonging to the application that is currently executed (i.e. on HMI).
CloseActiveMessage()
Method removes message belonging to the application that is currently executed (i.e. on HMI).
bool IsFullscreen()const
Method returns TRUE if hmi level equals HMI_FULL, otherwise returns FALSE.
bool MakeFullscreen()
Method changes attributes of application. Method changes following attributes: hmi_level_ = HMI_FULL; audio_streaming_state_ = AUDIBLE; system_context_ = SYSCTXT_MAIN; has_been_activated = true. Returns TRUE.
bool IsAudible()const
Method returns TRUE if attribute hmi_level_ = HMI_FULL or hmi_level_ = HMI_LIMITED otherwise returns FALSE.
MakeNotAudible()
Method changes attributes of application. Method changes following attributes: hmi_level_ = HMI_BACKGROUND; audio_streaming_state_= NOT_AUDIBLE.
bool allowed_support_navigation()const
Method returns TRUE if application supports navigation, otherwise returns FALSE.
set_allowed_support_navigation(bool allow)
Method sets attribute “allowed_support_navigation_”. Parameter “allow” defines support navigation.
bool hmi_supports_navi_streaming()const
Method returns attribute “hmi_supports_navi_streaming_”. If returns true HMI supports streaming of navigation.
set_hmi_supports_navi_streaming(const bool& supports)
Method sets attribute “hmi_supports_navi_streaming_”. Parameter “supports” defines support streaming of navigation.
bool app_allowed()const
Method returns attribute value “is_app_allowed_”.
bool has_been_activated()const
Method returns attribute value “has_been_activated_”.
const Version& version()const
Method returns a supported version of api.
set_hmi_application_id(uint32_t hmi_app_id)
Method sets “hmi_app_id_”. Parameter “hmi_app_id” transmits the value set.
uint32_t hmi_app_id()const
Method returns attribute value “hmi_app_id_”.
uint32_t app_id()const
Method returns attribute value “app_id_”.
const string& name()const
Method returns name of registered application.
const string folder_name()const
Method returns name of folder where store data of application.
bool is_media_application()const
Method returns TRUE if application supports media, otherwise returns FALSE.
const mobile_api::HMILevel::eType& hmi_level()const
Method returns current HMI level for application.

const uint32_t put_file_in_none_count()const
Method returns amount of “PutFileRequest” which was sent when HMI level of application was equal to HMI_NONE.
const uint32_t delete_file_in_none_count()const
Method returns amount of “DeleteFileRequest” which was sent when HMI level of application was equal to HMI_NONE.
const uint32_t list_files_in_none_count()const
Method returns amount of “ListFilesRequest” which was sent when HMI level was equal to HMI_NONE.
const mobile_api::SystemContext::eType& system_context()const
Method returns current system context for application.
const mobile_api::AudioStreamingState::eType& audio_streaming_state()const
Method returns current audio streaming state.
const std::string& app_icon_path()const
Method returns path to application icon.
connection_handler::DeviceHandle device()const
Method returns id of device is running application.
set_tts_speak_state(bool state_tts_speak)
Method sets attribute tts_speak_state_ if tts or speak module is active on HMI tts_speak_state_ will be set TRUE. Parameter “state_tts_speak” contains state of tts or speak module.
bool tts_speak_state()
Method returns TRUE if tts or spek module is active on HMI otherwise returns FALSE.
set_version(const Version& ver)
Method sets a supported version of api. Parameter “ver” contains structure with supported version.
set_name(const std::string& name)
Method sets name of registered application. Parameter “name” contains name of registered application.
set_is_media_application(bool is_media)
Method sets attribute “is_media_” if application supports media. Parameter “is_media” is TRUE if media is supported.
set_hmi_level(const mobile_api::HMILevel::eType& hmi_level)
Method sets current HMI level. Parameter hmi_level contains current HMI level.
increment_put_file_in_none_count()
Method increases the amount of “PutFileRequest” which was sent when HMI level was equal to HMI_NONE.
increment_delete_file_in_none_count()
Method increases the amount of “DeleteFileRequest” which was sent when HMI level was equal to HMI_NONE.
increment_list_files_in_none_count()
Method increases the amount of “ListFilesRequest” which was sent when HMI level was equal to HMI_NONE.
set_system_context(const mobile_api::SystemContext::eType& system_context)
Method sets current system context for application. Parameter “system_context” contains current system context for application.
set_audio_streaming_state(const mobile_api::AudioStreamingState::eType& state)
Method sets current audio streaming state for application. Parameter “state” contains current audio streaming state.
bool set_app_icon_path(const std::string& path)
Method sets path to application icon. Parameter “path” contains path to application icon.
set_app_allowed(const bool& allowed)
Method sets attribute “is_app_allowed_”.
set_device(connection_handler::DeviceHandle device)
Method sets id of device is running application. Parameter “device” contains device id.
uint_32t get_grammar_id()const
Method returns a randomly generated value.
set_grammar_id(uint_32t value)
Method sets attribute “grammar_id_”. Parameter “value” contains randomly generated value.
set_protocol_version(const ProtocolVersion& protocol_version)
Method sets protocol version of registered application. Parameter “protocol_version” contains current version of protocol.
ProtocolVersion protocol_version()const
Method returns protocol of registered application.
bool AddFile(AppFile& File)
Method retains information about the file which was downloaded by “PutFileRequest”. Returns TRUE if file does not exist, otherwise returns FALSE. Parameter “file” contains information about file.
bool UpdateFile(AppFile& File)
Method updates information about the file which was downloaded by “PutFileRequest”. Returns TRUE if file exist, otherwise returns FALSE. Parameter “file” contains information about file.
bool DeleteFile(const std::string& file_name)
Method removes information about the file which was downloaded by “PutFileRequest”. Returns TRUE if file was removed success, otherwise returns FALSE. Parameter “file_name” contains name of the deleted file.
const AppFilesMap& getAppFiles()const
Method returns information about the files stored by application.

const AppFile* GetFile(const std::string& file_name)
Method returns information about the file stored by application. Parameter “file_name” contains name file. If file does not exists method returns NULL.
bool SubscribeToButton(mobile_apis::ButtonName::eType btn_name)
Method retains name of button. Returns TRUE if the name is stored successfully. Parameter “btn_name” contains name of button.
bool IsSubscribedToButton(mobile_apis::ButtonName::eType btn_name)
Method returns TRUE if list of name of buttons contains a specific name. Parameter “btn_name” contains name of button.
bool UnsubscribeFromButton(mobile_apis::ButtonName::eType btn_name)
Method removes name of button from list. Returns TRUE if the name is removed successfully. Parameter “btn_name” contains name of button.
bool SubscribeToIVI(uint32_t vehicle_info_type_)
Method retains vehicle info type. Parameter contains vehicle info type. Returns TRUE if data is stored successfully
bool IsSubscribedToIVI(uint32_t vehicle_info_type_)
Method returns TRUE if list of vehicle info type contains a specific type. Parameter “vehicle_info_type” contains vehicle info type.
bool UnsubscribeFromIVI(uint32_t vehicle_info_type_)
Method removes vehicle info type from list. Returns TRUE if the vehicle info type is removed successfully. Parameter “vehicle_info_type” contains vehicle info type.
const std::set<mobile_apis::ButtonName::eType>& SubscribedButtons()const
Method returns the list of names of buttons that have been saved by application.
const std::set<uint32_t>& SubscribesIVI()const
Method returns the list of vehicle info type that have been saved by application.
uint32_t nextHash()
Method returns a randomly generated value.
uint32_t curHash()
Method returns attribute “hash_val_”.
uint32_t UpdateHash()

Method changes Hash for current application and send notification to mobile. Returns updated_hash.

bool alert_in_background()const
Method returns attribute alert_in_background_. Returns TRUE if application runs alert request from background level, otherwise returns FALSE
set_alert_in_background(bool state_of_alert)
Method sets TRUE if application activates alert in background level otherwise sets FALSE. Parameter contains current state of alert.

4.5.14. HMIMessageObserver description

OnMessageReceived – message received from HMI callback.

OnErrorSending – error report received from HMI callback.

4.5.15. HMIMessageHandler description

AddHMIMessageAdapter – store pointer to message adapter.

RemoveHMIMessageAdapter – remove HMIMessageAdapter by pointer.

4.5.16. HMIMessageSender description

SendMessageToHMI – send message to HMI from application_manager

4.5.17. HMIMessageAdapter description

handler() – return handler_.

SubscribeTo - Each class implementing interface should use it according to standarts of transport for which it is to be an adapter. For example, Adapter for MessageBroker will use it to subscribe to notifications from HMI.

4.5.18. HMIMessageHandlerImpl description

OnMessageReceived(MessageSharedPointer message) – message received from HMI. Message is set to stack “receive message from HMI”.

SendMessageToHMI(MessageSharedPointer message) – send message to HMI from application_manager. Message is set to stack “receive message to HMI”.

set_message_observer(HMIMessageObserver* observer) – set HMI message observer.
OnErrorSending(MessageSharedPointer message) – error report received from HMI.
AddHMIMessageAdapter(HMIMessageAdapter* adapter) – store pointer to message adapter.
RemoveHMIMessageAdapter(HMIMessageAdapter* adapter) – remove HMIMessageAdapter by pointer.
Handle(const impl::MessageFromHmi& message) – is called by thread that checks the stack "receive message from HMI". Sends message to HMIMessageObserver.
Handle(const impl::MessageToHmi& message) – is called by thread that checks the stack "receive message to HMI". Send message to HMIMessageAdapter -s.

4.5.19. PolicyManager description

void set_listener(PolicyListener* listener) – Set listener instance to be able to call its interface methods.
bool LoadPTFromFile(const std::string& file_name) - Loads Policy Table from json file. Used for loading Preloaded Policy Table or after Master Reset of the system. Returns success of operation.
bool LoadPT(const std::string& file, const BinaryMessage& pt_content) - Updates Policy Table from binary message received from mobile device. Saves to Policy Table diff between Policy Table sent in snapshot and received Policy Table. Return success of operation.
std::string GetUpdateUrl(int service_type) - Gets URL for sending PTS to from PT itself. Returns URL.
EndpointUrls GetUpdateUrls(int service_type) - Gets all URLs for sending PTS to from PT itself. Returns vector of URLs.
BinaryMessageSptr RequestPTUpdate() - PTU is needed, for this PTS has to be formed and sent. Returns pointer to PTS as binary message.
CheckPermissionResult CheckPermissions(const PTString& app_id, const PTString& hmi_level, const PTString& rpc) - Check if specified RPC for specified application has permission to be executed in specified HMI Level and also its permitted params. Returns struct with permission and parameters, if any.
bool ResetUserConsent() - Clear all record of user consents. Used during Factory Reset. Returns success of operation.
void CheckAppPolicyState(const std::string& application_id) - Checks, if policy update is necessary for application.
PolicyTableStatus GetPolicyTableStatus() - Returns current status of policy table for HMI. Returns current status of policy table.
bool ExceededIgnitionCycles() - Checks is PT exceeded IgnitionCycles. True, if exceeded.
bool ExceededDays(int days) - Checks is PT exceeded days. True, if exceeded.
bool ExceededKilometers(int kilometers) - Checks is PT exceeded kilometers. True, if exceeded.
void IncrementIgnitionCycles() - Increments counter of ignition cycles.
void ResetRetrySequence() - Resets retry sequence.
int NextRetryTimeout() - Gets timeout to wait before next retry updating PT. If timeout is less or equal to zero then the retry sequence is not need. Returns timeout in seconds.
int TimeoutExchange() - Gets timeout to wait until receive response. Returns timeout in seconds.
const std::vector<int> RetrySequenceDelaysSeconds() - List of timeouts in seconds between retries, when attempt to update PT fails. Returns list of delays between attempts.
void OnExceededTimeout() - Handler of exceeding timeout of exchanging policy table
DeviceConsent GetUserConsentForDevice(const std::string& device_id) - Check user consent for mobile device data connection. Returns status of device consent.
void SetUserConsentForDevice(const std::string& device_id, bool is_allowed) - Set user consent for mobile device data connection.
bool ReactOnUserDevConsentForApp(const std::string app_id, bool is_device_allowed) - Update Application Policies as reaction on user allowing/disallowing device this app is running on. Returns success of operation.
void PTUpdatedAt(int kilometers, int days_after_epoch) - Sets number of kilometers and days after epoch, that passed for receiving PT Update.

bool GetInitialAppData(const std::string& application_id, StringArray* nicknames = NULL, StringArray* app_hmi_types = NULL) - Retrieves data from app_policies about app on its registration. Returns success of operation.

void SetDeviceInfo(const std::string& device_id, const DeviceInfo& device_info) - Stores device parameters received during application registration to policy table.

void SetUserConsentForApp(const PermissionConsent& permissions) - Set user consent for application functional groups.

bool GetDefaultHmi(const std::string& policy_app_id, std::string* default_hmi) - Get default HMI level for application. Returns success of operation.

bool GetPriority(const std::string& policy_app_id, std::string* priority) - Get priority for application. Returns success of operation.

std::vector<UserFriendlyMessage> GetUserFriendlyMessages(const std::vector<std::string>& message_code, const std::string& language) - Get user friendly messages for given RPC messages and language. Returns array of structs with appropriate message parameters.

bool IsApplicationRevoked(const std::string& app_id) const - Checks if the application is revoked. Returns success of operation.

void GetUserPermissionsForApp(const std::string& device_id, const std::string& policy_app_id, std::vector<FunctionalGroupPermission>& permissions) - Get user permissions for application which started on specific device.

AppPermissions GetAppPermissionsChanges(const std::string& app_id) – Gets changes of application permissions from update. Returns appropriate data struct.

void RemovePendingPermissionChanges(const std::string& app_id) – Removes pending permission changes.

std::string& GetCurrentDeviceId(const std::string& policy_app_id) - Return device id, which hosts specific application.

void SetSystemLanguage(const std::string& language) - Set current system language in policy table.

void SetSystemInfo(const std::string& ccpu_version, const std::string& wers_country_code, const std::string& language) - Set data from GetSystemInfo response to policy table.

void SendNotificationOnPermissionsUpdated(const std::string& application_id) - Send OnPermissionsUpdated for choosen application.

void MarkUnpairedDevice(const std::string& device_id) - Marks device as unpaired.

bool CleanupUnpairedDevices() - Removes unpaired device records and related records from DB. Returns success of operation.

bool CanAppKeepContext(const std::string& app_id) - Check if app can keep context. True, if can.

bool CanAppStealFocus(const std::string& app_id) - Check if app can steal focus. True, if can.

4.5.20. PolicyHandler description

PolicyManager* LoadPolicyLibrary()– Load policy shared library. Return pointer to PolicyManager instance, or NULL, if library was not found.

PolicyManager* policy_manager() – Returns pointer to PolicyManager instance.

bool InitPolicyTable() – Initializes policy table from stored file. Returns success of operation.

bool RevertPolicyTable() – Remove user records from policy table. Returns success of operation.

bool SendMessageToSDK(const BinaryMessage& pt_string) – Sends policy table snapshot to get update for policy table. Returns success of operation.

bool ReceiveMessageFromSDK(const std::string& file, const BinaryMessage& pt_string)– Gets and parse received message from application manager. Returns success of operation.

bool UnloadPolicyLibrary() – Unloads policy shared library. Returns success of operation.

void OnPTExchangeNeeded() – Handler method, which is starts necessary actions to receive policy table update

void OnPermissionsUpdated(const std::string& policy_app_id, const Permissions& permissions, const HMILevel& default_hmi)– Sends notification with updated permissions to mobile.

void KmsChanged(int kms) - Lets client to notify PolicyHandler that more kilometers expired
void OnActivateApp(uint32_t connection_key, uint32_t correlation_id) - Gather information for application and sends it to HMI
void OnAllowSDLFunctionalityNotification(bool is_allowed, uint32_t device_id = 0) - Process user consent on mobile data connection access
void OnIgnitionCycleOver() - Increment counter for ignition cycles
void OnAppRevoked(const std::string& policy_app_id) - Send notification to HMI concerning revocation of application
void OnPendingPermissionChange(const std::string& policy_app_id) – Process pending permissions on change
void PTExchangeAtIgnition() - Initializes PT exchange at ignition if need
void PTExchangeAtUserRequest(uint32_t correlation_id) - Initializes PT exchange at user request
void SetDeviceInfo(std::string& device_id, const DeviceInfo& device_info) - Save device info for specific device to policy table
void OnAppPermissionConsent(const PermissionConsent& permissions) - Store user-changed permissions consent to policy table
void OnGetUserFriendlyMessage(const std::vector<std::string>& message_codes, const std::string& language, uint32_t correlation_id) - Get appropriate message parameters and send them with response to HMI
void OnGetListOfPermissions(const uint32_t connection_key, const uint32_t correlation_id) - Get list of permissions for application/device binded to connection key from request and sends response.
void OnGetStatusUpdate(const uint32_t correlation_id) - Get current policy table update state and send response.
void OnUpdateStatusChanged(policy::PolicyTableStatus status) - Send notification to HMI with changed policy update status.
std::string OnCurrentDeviceIdUpdateRequired(const std::string& policy_app_id) - Update currently used device id in policies manager for given application. Returns updated device id.
void OnSystemInfoChanged(const std::string& language) - Set parameters from OnSystemInfoChanged to policy table.
void OnGetSystemInfo(const std::string& ccpu_version, const std::string& wers_country_code, const std::string& language) - Save data from GetSystemInfo request to policy table
void OnSystemInfoUpdateRequired() - Send request to HMI to get update on system parameters
void RemoveDevice(const std::string& device_id) - Removes device record from policy table.
void AddStatisticsInfo(int type) - Adds statistics info to policy table.
void OnSystemError(int code) – Increments system errors counters
uint32_t GetAppIdForSending() - Choose application id to be used for snapshot sending. Returns application id or 0, if there are no applications registered.
std::string GetAppName(const std::string& policy_app_id) – Returns application name for given application id.
BinaryMessageSptr AddHttpHeader(const BinaryMessageSptr& pt_string) - Adds http header (temporary method). Retuns pointer to binary message, which contains PTS with HTTP header.
void StartNextRetry() - Starts next retry exchange policy table.
void PTExchangeAtOdometer(int kilometers) - Initializes PT exchange at odometer if need
void StartPTExchange(bool skip_device_selection = false) - Starts process of updating policy table

4.5.21. PTRepresentation description

CheckPermissionResult CheckPermissions(const PTString& app_id, const PTString& hmi_level, const PTString& rpc) - Check if specified RPC for specified application has permission to be executed in specified HMI Level and also its permitted params. Returns CheckPermissionResult containing flag if HMI Level is allowed and list of allowed params.
bool IsPTPreloaded() - Returns true if Policy Table was not updated yet from preloaded PT file.
int IgnitionCyclesBeforeExchange() - Gets number of ignition cycles before next update policy table.
int KilometersBeforeExchange(int current) - Gets value in kilometers before next update policy table.

bool SetCountersPassedForSuccessfulUpdate(int kilometers, int days_after_epoch) - Sets kilometers and days after epoch, that passed for recieved successful PT Update. Returns success of operation.
int DaysBeforeExchange(int current) - Gets value in days before next update policy table.
void IncrementIgnitionCycles() - Increment number of ignition cycles since last exchange by 1
void ResetIgnitionCycles() - Reset number of ignition cycles since last exchange to 0
int TimeoutResponse() - Returns timeout to wait for a response of PT update
bool SecondsBetweenRetries(std::vector<int>* seconds) - Returns number of seconds between each try of sending PTS. Returns success of operation.
VehicleData GetVehicleData() - Get information about vehicle. Returns appropriate data struct.
std::vector<UserFriendlyMessage> GetUserFriendlyMsg(const std::vector<std::string>& msg_codes, const std::string& language) - Get message text for displaying/pronouncing for user dependent on language and context. Retruns array of appropriate messages parameters.
EndpointUrls GetUpdateUrls(int service_type) - Get list of URL to send PTS to. Returns list for specific service or default.
int GetNotificationsNumber(policy_table::Priority priority) - Get allowed number of notifications depending on application priority.
InitResult Init() - Initialized Policy Table (load). Returns success of operation.
bool Close() - Close policy table storage. Returns success of operation.
bool Clear() - Removes policy table content. Returns success of operation.
utils::SharedPtr<policy_table::Table> GenerateSnapshot() const - Get snapshot of Policy Table including app_policies, functional_groups, device_info, statistics, excluding user messages. Returns pointer to generated structure for obtaining Json string.
bool Save(const policy_table::Table& table) - Saves policy table in storage. Returns success of operation.
bool UpdateRequired() const - Gets flag updateRequired. True, if required.
void SaveUpdateRequired(bool value) - Saves flag updateRequired.
bool GetInitialAppData(const std::string& app_id, StringArray* nicknames = NULL, StringArray* app_types = NULL) - Retrieves data from app_policies about app on its registration. Returns success of operation.
bool IsApplicationRevoked(const std::string& app_id) const - Checks if the application is revoked. True, if revoked.
bool GetFunctionalGroupings(policy_table::FunctionalGroupings& groups) - Get functional groupings from DB. Returns success of operation.
bool IsApplicationRepresented(const std::string& app_id) const - Checks if the application is represented in policy table. True, if represented in policy table.
bool IsDefaultPolicy(const std::string& app_id) const - Checks if the application has default policy. True, if has default policy.applied.
bool IsPredataPolicy(const std::string& app_id) const - Checks if the application has pre_data policy. True, if pre_DataConsent policy applied.
bool SetDefaultPolicy(const std::string& app_id) - Sets default policy for application. Returns success of operation.

4.5.22. PTExtRepresentation description

bool CanAppKeepContext(const std::string& app_id) - Is application allowed to send notifications while in background or limited mode. True, if allowed.
bool CanAppStealFocus(const std::string& app_id) - Is application allowed to move foreground at will. True, if allowed.
bool GetDefaultHMI(const std::string& policy_app_id, std::string* default_hmi) - Get default_hmi for given application. Returns success of operation.
bool GetPriority(const std::string& policy_app_id, std::string* priority) - Get priority for given application. Returns success of operation.
bool ResetUserConsent() - Reset user consent for device data and applications permissions. Returns success of operation.
bool ResetDeviceConsents() - Reset user consent for device data. Returns success of operation.

bool ResetAppConsents() - Reset user consent for applications permissions. Returns success of operation.

bool GetUserPermissionsForDevice(const std::string& device_id, StringArray* consented_groups = NULL, StringArray* disallowed_groups = NULL) - Get user permissions for device data usage. Returns success of operation.

bool GetUserPermissionsForApp(const std::string& device_id, const std::string& policy_app_id, FunctionalIdType* group_types) - Get user permissions for application assigned groups. Returns success of operation.

bool GetDeviceGroupsFromPolicies(policy_table::Strings* groups = NULL, policy_table::Strings* preconsented_groups = NULL) - Get device groups and preconsented groups from policies section. Returns success of operation.

bool SetDeviceData(const std::string& device_id, const std::string& hardware = "", const std::string& firmware = "", const std::string& os = "", const std::string& os_version = "", const std::string& carrier = "", const uint32_t number_of_ports = 0) - Record information about mobile device in Policy Table. Returns success of operation.

bool SetUserPermissionsForDevice(const std::string& device_id, const StringArray& consented_groups = StringArray(), const StringArray& disallowed_gropus = StringArray()) - Sets user consent for particular mobile device, i.e. to use device for exchanging of Policy Table. Returns success of operation.

bool ReactOnUserDevConsentForApp(const std::string& app_id, bool is_device_allowed) - Update Application Policies as reaction on User allowing/disallowing device this app is running on. Returns success of operation.

bool SetUserPermissionsForApp(const PermissionConsent& permissions) - Set user consent on functional groups. Returns success of operation.

bool IncreaseStatisticsData(StatisticsType type) - Counter for statistics information: adds 1 to existing number.

bool SetAppRegistrationLanguage(const std::string& app_id, LanguageType type, const std::string& language) - Records information about what language application tried to register with. Returns success of operation.

bool SetMetaInfo(const std::string& ccpu_version, const std::string& wers_country_code, const std::string& language) - Records information about head unit system to PT. Returns success of operation.

int GetKmFromSuccessfulExchange() – Gets kilometers passed since last successful PT update.

int GetDayFromScsExchange() – Gets days passed since last successful PT update.

int GetIgnitionsFromScsExchange() – Gets number of ignition cycles passed since last successful PT update.

bool SetSystemLanguage(const std::string& language) - Set current system language. Returns success of operation.

void Increment(const std::string& type) const - Increments global counter.

void Increment(const std::string& app_id, const std::string& type) const - Increments counter of application

void Set(const std::string& app_id, const std::string& type, const std::string& value) const - Sets value of application information

void Add(const std::string& app_id, const std::string& type, int seconds) const - Adds value to stopwatch of application.

bool CountUnconsentedGroups(const std::string& policy_app_id, int* result) const - Counts unconsented groups for given application id. Returns success of operation.

bool GetFunctionalGroupNames(policy::FunctionalGroupNames& names) - Gets functional group names and user_consent_prompts, if any. Returns success of operation.

bool SetPredataPolicy(const std::string& app_id) - Set app policy to pre_DataConsented policy. Returns success of operation.

bool SetIsPredata(const std::string& app_id, bool is_pre_data) - Updates application policy to either pre_DataConsented or not. Returns success of operation.

bool CleanupUnpairedDevices(const DeviceIds& device_ids) const - Removes unpaired devices. Returns success of operation.

bool SetUnpairedDevice(const std::string& device_id) const - Sets flag of unpaired device. Returns success of operation.

bool UnpairedDevicesList(DeviceIds* device_ids) const - Gets list of unpaired devices. Returns success of operation.

4.6. Process interfaces

N/A

5. Rationale

N/A

6. Requirements Traceability

SAD Section	SDD Section
TBD	

7. Appendices

N/A

8. References

9. Change History

Version	Date	Status	Change description	Author/Editor
v.0.1	06/14/2013	Initial Draft	Major sections description	A.Kandul
v.0.2	06/25/2013	Draft	Diagrams adding	A.Kandul
v.0.3	07/02/2013	Draft	Ready for internal review	A.Kandul
v.0.4	07/10/2013	Draft	Updating description after internal review.	A.Kandul
v.1.1	25/03/2014	Draft	Add resume controller	A.Kutsan
v.1.2	13/05/2014	Draft	Update application_manager design	D.Trunov
v. 1.3	22/05/2014	Draft	Policy component was added	A.Oleynik
v.1.4	03/06/2014	Draft	Add hmi_message_handler design	V.Slobodyanik
v.1.5	18/07/2014	Draft	Add connection_handler design	V.Semenyuk

10. Approve History

Version	Approval Date	Issue ID	Approved By
1.0	08/11/2013	APPLINK-4043	N. Snitsar