Web Services for Radio Resource Control

Evelina Pencheva(✉) and Ivaylo Atanasov

Technical University of Sofia, Sofia, Bulgaria
{enp,iia}@tu-sofia.bg

Abstract. Mobile Edge Computing (MEC) supports network function virtualization and it brings network service intelligence close to the network edge. MEC services provide low level radio and network information to authorized applications. Communication between MEC services and applications is according to the principles of Service–oriented Architecture (SOA). In this paper, we propose an approach to design application programming interfaces for MEC Web Services that may be used by RAN analytics applications to adapt content delivery in real–time improving quality of experience to the end users. Web Service interfaces are mapped onto network protocols.

Keywords: Mobile Edge Computing · 5G · Radio Resource Control · Radio Network Information Services · Service oriented Architecture · Behavioral models

1 Introduction

Mobile Edge Computing (MEC) is a hot topic in 5 G. MEC supports network function virtualization and it brings network service intelligence close to the network edge [1]. MEC enables low latency communications, big data analysis close to the point of capture and flexible network management in response to user requirements [2,3]. MEC is required for critical communications which demand processing traffic and delivering applications close to the user [4,5]. MEC provides real-time network data such as radio conditions, network statistics, etc., for authorized applications to offer context-related services that can differentiate end user experience. Some of the promising real-time MEC application scenarios are discussed in [6].

MEC use cases and deployment options are presented in [7]. The European Telecommunications Standards Institute (ETSI) defined MEC reference architecture, where MEC deployment can be inside the base station or at aggregation point within Radio Access Network (RAN) [8]. Minimal latency for many applications can be achieved by integrating MEC server in base station [9,10].

The communications between applications and services in the MEC server are designed according to the principles of Service-oriented Architecture (SOA). The Radio Network Information Services (RNIS) provide information about the mobility and activity of User Equipment (UE) in the RAN. The information includes parameters on the UE context and established E-UTRAN Radio Access

A. Longo et al. (Eds.): IISSC 2017/CN4IoT 2017, LNICST 189, pp. 188–198, 2018.
https://doi.org/10.1007/978-3-319-67636-4_20

Bearer (E-RAB), such as Quality of Service (QoS), Cell ID, UE identities, etc. This information is available based on the network protocols like Radio Resource Control (RRC), S1 Application Protocol (S1-AP), and X2 Application Protocol (X2-AP) [11].

ETSI standards just identified the required MEC service functionality, but do not define Web Service application programming interfaces (APIs). As far as our knowledge there is a lack of research on MEC service APIs, and the related works consider MEC applications that may use MEC services. In this paper we propose an approach to design APIs of SOA based Web Services for access to radio network information.

The paper is structured as follows. Section 2 provides a detailed Web Service description including definitions of data structure, interfaces, interface operation and use cases. Section 3 describes functionality required for mapping of Web Service interfaces onto network protocols. Device state models are described and formally verified. The conclusion summarizes the authors' contributions and highlights the benefits of the proposed approach.

2 Detailed Service Description

2.1 Device Context Web Service

Device Context service provides access to the UE context including EPS Mobility Management (EMM) state, EPS Connectivity Management (ECM) state, RRC state, UE identities, and Cell-ID. This information is provided through:

– Request for the UE context of a device;
– Request for the UE context of a group of devices;
– Notification change in the context of a device;
– Notification of device context on a periodic basis.

The response to a request for a group of devices may contain a full or partial set of results. The results are provided based on a number of criteria including number of devices for which the request is made and amount of time required to retrieve the information. Additional requests may be initiated for those devices for which information was not provided.

The EMM states describe mobility management states that result from the Attach and Tracking Area Update (TAU) procedures. The $EMMstatus$ is of enumeration type with values of EMM-Deregistered (device is deregistered and it is not accessible) and EMM-Registered (device is registered to the network). The ECM states describe the signaling connectivity between the device and the core networks. The $ECMstatus$ is of enumeration type with values of ECM-Idle (there is no non-access stratum signaling connection between the device and the network) and ECM-Connected (there is a non-access stratum signaling connection). The RRC states describe the connection between the device and the RAN. The $RRCstatus$ is also of enumeration type with values of RRC-Idle (there is no RRC connection between the device and the network), and RRC-Connected (an RRC connection between the device and the RAN is established). The UE identity information is represented by C-RNTI (Cell Radio Network Temporary Identity) which identified the RRC connection. The Cell-ID uniquely identifies the E-Node B which currently serves the device.

StatusData structure contains the device context information. As this can be related to a query of a group of devices, the *ResultStatus* element is used. It is of enumerated type with values indicating whether the information for the device was retrieved or not, or if an error occurred. Table 1 illustrates the *StatusData* elements.

Table 1. Structure of device status data

Names	Types	Description
DeviceAddress	xsd:anyURI	Address of the device to which the UE context information applies
ReportingStatus	ResultStatus	Status of retrieval for this address
CurrentEMMstatus	EMMstatus	EMM status of the device if the ReportingStatus is equal to Retrieved
CurrentECMstatus	ECMstatus	ECM status of the device if the ReportingStatus is equal to Retrieved
CurrentRRCstatus	RRCstatus	RRC status of the device if the ReportingStatus is equal to Retrieved
CellID	integer	Cell-ID of the ENodeB which serves the device
CRNTI	integer	C-RNTI of the device

The Device Context Web Service supports the following interfaces:

The *UEContext* Interface requests the context information for a device. It supports two operations. The *GetUEContext* operation is intended to retrieve the context for a single device. The *GetUEContextForGroup* operation initiates a retrieval of context data for a group of devices.

The *UEContextNotificationManager* interface may be used to set up notifications about the events related to given device context. The operation *StartPeriodicNotifications* makes periodic notifications available to applications (the operation defines maximum frequency of notifications and the length of time notifications occur for). The *StartTriggeredNotification* operation makes triggered notifications available to applications (the operation defines maximum frequency of notifications, maximum number of notifications, period of time notifications are provided for, and the criteria). The *EndNotifications* operation ends either type notifications.

The interface to which notifications are delivered is *UEContextNotification*. It supports the following operations. The *UEContextNotification* operation is used to notify the application when the context of the monitored device changes. The *UEContextError* operation is used to inform the application that the notifications for a device or a group of devices are cancelled by the Web Service.

The *UEContextEnd* operation informs the application that the notifications have been completed when the duration or count for notifications have been completed. Figure 1 shows the sequence diagram for on demand access to device context information and triggered device context notification.

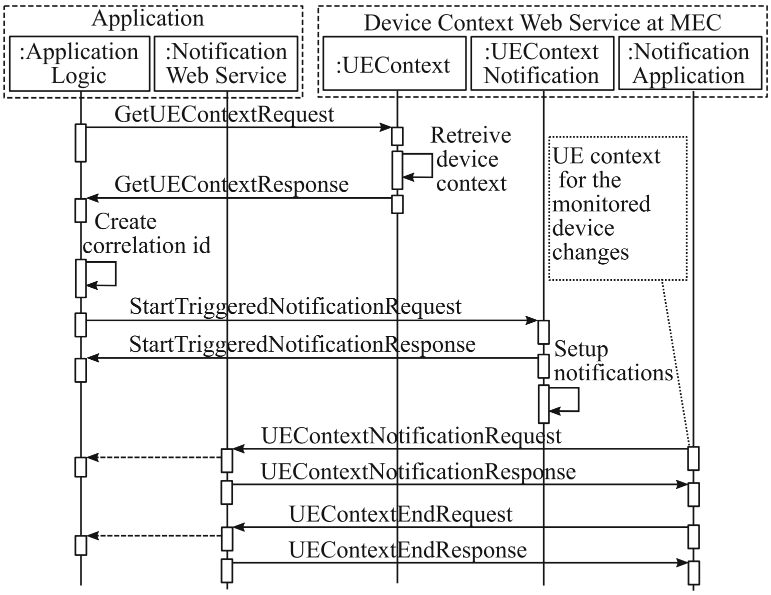
Application
:Application Logic
:Notification Web Service
Device Context Web Service at MEC
:UEContext
:UEContext Notification
:Notification Application
GetUEContextRequest
Retreive device context
GetUEContextResponse
UE context for the monitored device changes
Create correlation id
StartTriggeredNotificationRequest
StartTriggeredNotificationResponse
Setup notifications
UEContextNotificationRequest
UEContextNotificationResponse
UEContextEndRequest
UEContextEndResponse

Fig. 1. Sequence diagram for on demand access to device context and triggered notifications

The Application queries for the device context and receives its context. The Application generates a correlator and starts triggered notifications. The Web Service sets up a notification to monitor changes in the device context. A notification is delivered to the Application when the device context changes. When the notifications are completed, the Application is notified.

2.2 Device Bearer Web Service

Device Bearer service provides access to information about active Radio Access Bearers (RABs) of the device and allows applications to dynamically manipulate device RABs. The information about devices RABs is provided on demand, periodically or upon event occurrence. An authorized Application may request RAB establishment, modification or release.

The Device Bearers Web Service supports the following interfaces:

The *ApplicationBearerControl* interface provides functionality for querying active RABs of the device, applying or modifying the QoS available on device connections. The *GetActiveRAB* operation retrieves the active RABs of the device. The *PutQoS* operation allows the Application to request a temporary QoS feature to be set up on the device connection (the operation may cause an establishment of a new RAB for the device). The *AlterQoS* operation allows the Application to modify the configurable service attributes on active temporary QoS feature. The *PutOffQoS* operation allows the Application to release a temporary QoS feature (this may cause the RAB bearer release). The *Disconnect* operation allows the Application to disconnect the device session (the operation causes release of all active RABs).

The *BearerNotificationManager* interface is used by the Applications to manage their subscriptions to notifications. The *StartPeriodicNotification* operation is used to register the Application interest in receiving notifications periodically. Examples of notifications are bearer establishment/ modification/ release, all bearers are lost, radio link failure. The *StartTriggeredNotification* operation is used to register the Application interest in receiving notifications about bearer related events. The *EndNotifications* operation is used by the Application to cancel any type of notifications.

The *BearerNotification* interface provides operation for notifying the Application about the bearer related events. The *RABNotification* operation reports a network event that has occurred against device active RABs. The *RABError* operation sends an error message to the Application to indicate the notification for a device is cancelled by the Web Service. The *RABEnd* operation informs the Application that the notifications have been completed when the duration expires or the count limit for notifications has been reached.

The *Bearer* interface supports one operation *GetRABAttributes* which allows the Application to query about device's bearer attributes.

3 Implementation Issues

As a mediation point between MEC applications and RAN the MEC server, which provides Web Services, needs to maintain the network and the application views on the device status. These views need to be synchronized. Furthermore, the MEC server needs to translate the Web Service interface operations into respective events in the network and vice versa.

3.1 Device State Models

Figure 2 shows the device state model as seen by the MEC server.

Table 2 provides a mapping between Web Services operations and network events.

Figure 3 shows the device state model as seen by the application.

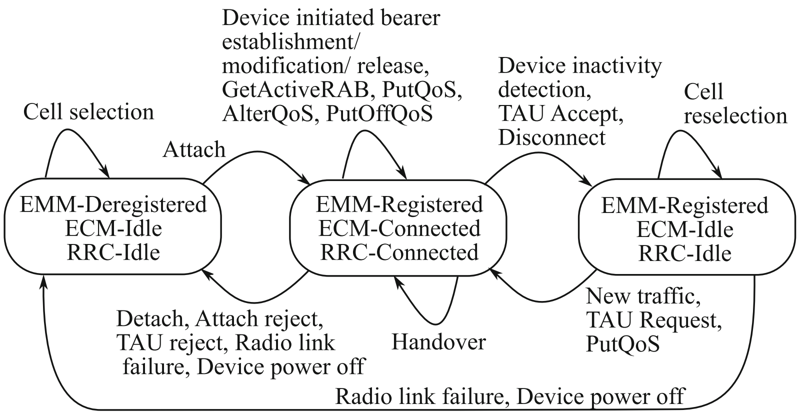
Device initiated bearer establishment/ modification/ release, GetActiveRAB, PutQoS, AlterQoS, PutOffQoS
Cell selection
Attach
Device inactivity detection, TAU Accept, Disconnect
Cell reselection
EMM-Deregistered ECM-Idle RRC-Idle
EMM-Registered ECM-Connected RRC-Connected
EMM-Registered ECM-Idle RRC-Idle
Detach, Attach reject, TAU reject, Radio link failure, Device power off
Handover
New traffic, TAU Request, PutQoS
Radio link failure, Device power off

Fig. 2. Device's state model as seen by the MEC server

Table 2. Structure of device status data

Web service operation	Network events
UEContextNotification	Detach, Attach Reject, TAU Reject, Radio Link Failure, UE Power Off, New Traffic, TAU Request, Cell reselection
RABNotification	Device initiated bearer establishment/release/modification
PutQoS	Application initiated bearer establishment
AlterQoS	Application initiated bearer modification
PutOffQoS	Application initiated bearer release
Disconnect	Application initiated data session release

The proposed state model representing the Application view on the device state includes the following states.

In *AppDeregistered* state, the device is not registered to the network. The respective states in the network are RRC-idle, EMM-deregistered and ECM-idle. During attachment to the network (*Attach*), the network notifies the Application about the change in the device context ($N_{UEcontext}$), the device moves to *AppActive* state and the respective network states are RRC-connected, EMM-registered and ECM-connected. After successful mobility management event without data transfer activity e.g. TAU_{Accept}, the network notifies the Application about the change in device context ($N_{UEcontext}$), and the Application considers the device being in *AppIdle* state, where the respective network states are RRC-idle, EMM-registered and ECM-idle. In case of unsuccessful mobility management event (e.g. $Attach_{Reject}$, TAU_{Reject}), the Application is notified (N_{MMrej}) and the device moves to *AppDeregistered* state.

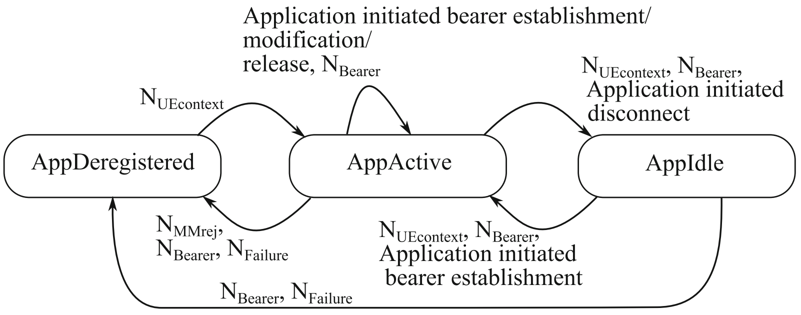
Application initiated bearer establishment/ modification/ release, N_{Bearer}
$N_{UEcontext}$
$N_{UEcontext}$, N_{Bearer}, Application initiated disconnect
AppDeregistered
AppActive
AppIdle
N_{MMrej}, N_{Bearer}, $N_{Failure}$
$N_{UEcontext}$, N_{Bearer}, Application initiated bearer establishment
N_{Bearer}, $N_{Failure}$

Fig. 3. Device's state model as seen by the MEC application

Application may decide to apply specific QoS ($App_{BearerEst}$) and it invokes the *PutQoS* operation. In *AppActive* state, the Application may decide to modify the established bearer ($App_{BearerMod}$) or to release it ($App_{BearerRel}$) and in these cases it invokes *AlterQoS* or *PutOffQoS* operations respectively. In *AppActive* state, the Application may decide to release all bearers i.e. to disconnect the device (App_{Discon}) and in this case it invokes *Disconnect* operation. The Application may also receive notifications about bearer related events (N_{Bearer}), e.g. notifications about device-initiated bearer establishment ($Device_{BearerEst}$), device-initiated bearer modification ($Device_{BearerMod}$), and device-initiated bearer release ($Device_{BearerRel}$).

During active data transfer, the device is in *AppActive* state. In this state, the Application may decide to apply specific QoS ($App_{BearerEst}$) and it invokes the *PutQoS* operation. In *AppActive* state, the Application may decide to modify the established bearer ($App_{BearerMod}$) or to release it ($App_{BearerRel}$) and in these cases it invokes *AlterQoS* or *PutOffQoS* operations respectively. In *AppActive* state, the Application may decide to release all bearers i.e. to disconnect the device (App_{Discon}) and in this case it invokes *Disconnect* operation. The Application may also receive notifications about bearer related events (N_{Bearer}), e.g. notifications about device-initiated bearer establishment ($Device_{BearerEst}$), device-initiated bearer modification ($Device_{BearerMod}$), and device-initiated bearer release ($Device_{BearerRel}$).

If the Application is notified that all bearers are lost ($N_{Bearers}$), it may initiate data session release on behalf of the device (*Disconnect*) and the device state becomes *AppIdle*. In *AppActive* state, when the network detects device inactivity, the Application is notified ($N_{UEcontext}$) and it considers the device being in *AppIdle* state.

In case the device detach (*Detach*) or device power is off ($Device_{PowerOff}$), the Application is notified ($N_{UEcontext}$) and it considers the device being in *AppDeregistered* state. In case of radio link failure, the network notifies the Application ($N_{Failure}$) and the device moves to *AppDeregistered* state.

3.2 Formal Verification of Device State Models

We use the mathematical formalism of Labeled Transition Systems (LTSs) to describe the device state models. An LTS is defined as a quadruple of set of states, set of inputs, set of transitions, and an initial state.

By $D_{App} = \left(S_{App}, Inp_{App}, \rightarrow_{App}, s_0^{App}\right)$ it is denoted an LTS representing the Application's view on device state where:

$$\begin{aligned}
S_{App} &= \{AppDeregistered, AppActive, AppIdle\}\,;\\
Inp_{App} &= \{N_{MMrej}, N_{Bearer}, N_{Bearers}, N_{Failure}, N_{UEcontext}, App_{BearerEst},\\
&\quad App_{BearerMod}, App_{BearerRel}, App_{Discon}\}\,;\\
\rightarrow_{App} &= \{(AppDeregistered\, N_{UEcontext}\, AppActive)\left[\tau_1^A\right], (AppActive\\
&\quad App_{BearerEst}\, AppActive)\left[\tau_2^A\right], (AppActive\, App_{BearerMod}\\
&\quad AppActive)\left[\tau_3^A\right], (AppActive\, App_{BearerRel}\, AppActive)\left[\tau_4^A\right],\\
&\quad (AppActive\, N_{Bearer}\, AppActive)\left[\tau_5^A\right], (AppActive\, N_{MMrej},\\
&\quad AppDeregistered)\left[\tau_6^A\right], (AppActive\, N_{Bearers}\, AppDeregistered)\\
&\quad \left[\tau_7^A\right], (AppActive\, N_{Failure}\, AppDeregistered)\left[\tau_8^A\right], (AppActive\\
&\quad N_{UEcontext}\, AppIdle)\left[\tau_9^A\right], (AppActive\, N_{Bearer}\, AppIdle)\left[\tau_{10}^A\right],\\
&\quad (AppActive\, App_{Discon}\, AppIdle)\left[\tau_{11}^A\right], (AppIdle\, N_{UEcontext}\\
&\quad AppActive)\left[\tau_{12}^A\right], (AppIdle\, N_{Bearer}\, AppActive)\left[\tau_{13}^A\right], (AppIdle\\
&\quad N_{BearerEst}\, AppActive)\left[\tau_{14}^A\right], (AppIdle\, N_{UEcontext}\, AppDeregistered)\\
&\quad \left[\tau_{15}^A\right], (AppIdle\, N_{Failure}\, AppDeregistered)\left[\tau_{16}^A\right]\}\,;\\
s_0^{App} &= \{AppDeregistered\}\,.
\end{aligned}$$

Short notations of the transitions are given in brackets.

Let us denote by *Deregistered* the device states in the network EMM-Deregistered, ECM-Idle, RRC-Idle, by *Connected* the device states in the network EMM-Connected, ECM-Connected, RRC-Connected, and by *Idle* the device states in the network EMM-Registered, ECM-Idle, RRC-Idle.

By $D_{MEC} = \left(S_{MEC}, Inp_{MEC}, \rightarrow_{MEC}, s_0^{MEC}\right)$ it is denoted an LTS representing the MEC server's view on device state where:

$$\begin{aligned}
S_{MEC} &= \{Deregistered, Connected, Idle\};\\
Inp_{MEC} &= \{Cell_{Selection}, Attach, Detach, Attach_{Reject}, TAU_{Reject}, TAU_{Accept},\\
&\quad Radio_{LinkFailure}, Device_{PowerOff}, Device_{BearerEst}, Cell_{Reselection},\\
&\quad Device_{BearerMod}, Device_{BearerRel}, PutOffQoS, GetActiveRAB,\\
&\quad AlterQoS, Disconnect, Handover, Device_{NewTraffic}, TAU_{Request},\\
&\quad PutQoS, Device_{Inactivity}\};\\
\rightarrow_{MEC} &= \big\{(Deregistered\, Cell_{Selection}\, Deregistered)\left[\tau_1^{MEC}\right], (Dere-\\
&\quad gistered\, Attach\, Connected)\left[\tau_2^{MEC}\right], (Connected\, Device_{BearerEst}\\
&\quad Connected)\left[\tau_3^{MEC}\right], (Connected\, Device_{BearerMod}\, Connected)\\
&\quad \left[\tau_4^{MEC}\right], (Connected\, Device_{BearerRel}\, Connected)\left[\tau_5^{MEC}\right],\\
&\quad (Connected\, GetActiveRAB\, Connected)\left[\tau_6^{MEC}\right], (Connected\\
&\quad PutQoS\, Connected)\left[\tau_7^{MEC}\right], (Connected\, AlterQoS\, Connected)\\
&\quad \left[\tau_8^{MEC}\right], (Connected\, PutOffQoS\, Connected)\left[\tau_9^{MEC}\right],\\
&\quad (Connected\, Handover\, Connected)\left[\tau_{10}^{MEC}\right], (Connected\, Detach\\
&\quad Deregistered)\left[\tau_{11}^{MEC}\right], (Connected\, Attach_{Reject}\, Deregistered)\\
&\quad \left[\tau_{12}^{MEC}\right], (Connected\, TAU_{Reject}\, Deregistered)\left[\tau_{13}^{MEC}\right],\\
&\quad (Connected\, Radio_{LinkFailure}\, Deregistered)\left[\tau_{14}^{MEC}\right],\\
&\quad (Connected\, Device_{PowerOff}\, Deregistered)\left[\tau_{15}^{MEC}\right], (Connected\\
&\quad Device_{Inactivity}\, Idle)\left[\tau_{16}^{MEC}\right], (Connected\, TAU_{Accept}\, Idle)\left[\tau_{17}^{MEC}\right]\\
&\quad (Idle\, Device_{NewTraffic}\, Connected)\left[\tau_{18}^{MEC}\right], (Idle\, TAU_{Request}\\
&\quad Connected)\left[\tau_{19}^{MEC}\right], (Idle\, PutQoS\, Connected)\left[\tau_{20}^{MEC}\right], (Idle\\
&\quad Cell_{Reselection}\, Idle)\left[\tau_{21}^{MEC}\right], (Idle\, Radio_{LinkFailure}\, Deregistered)\\
&\quad \left[\tau_{22}^{MEC}\right], (Idle\, Device_{PowerOff}\, Deregistered)\left[\tau_{23}^{MEC}\right]\big\};\\
s_0^{MEC} &= \{Deregistered\}.
\end{aligned}$$

We use weak bisimulation to formally verify the suggested models.

Proposition 1. *The systems D_{App} and D_N are weakly bisimilar.*

Proof. As to definition of weak bisimulation, provided in [12], it is necessary to identify a bisimilar relation between the states of both LTSs and to identify respective matching between transitions.

Let the relation U_{MECApp} be defined as $U_{MECApp} = \{(Deregistered, AppDeregistered), (Connected, AppActive), (Idle, AppIdle)\}$, then:

1. In case of attachment, for $Deregistered \exists \left\{\tau_1^{MEC}, \tau_2^{MEC}\right\}$ that leads to *Connected*, and for $AppDeregistered \exists \left\{\tau_1^{A}\right\}$ that leads to *AppActive*.

2. In case of detach, or attach reject, or TAU reject, or radio link failure, or device power off, for *Connected* $\exists \left\{\tau_{11}^{MEC} \vee \tau_{12}^{MEC} \vee \tau_{13}^{MEC} \vee \tau_{14}^{MEC} \vee \tau_{15}^{MEC}\right\}$ that leads to *Deregistered*, and for *AppActive* $\exists \left\{\tau_6^A \vee \tau_7^A \vee \tau_8^A\right\}$ that leads to *AppDeregistered.*
3. In case of device initiated bearer establishment/modification/release, for *Connected* $\exists \left\{\tau_3^{MEC}, \tau_4^{MEC}, \tau_5^{MEC}\right\}$ that leads to *Connected*, and for the state *AppActive* $\exists \left\{\tau_5^A\right\}$ that leads to *AppActive.*
4. In case of *Application* initiated establishment/modification/release, for state *Connected* $\exists \left\{\tau_6^{MEC}, \tau_7^{MEC}, \tau_8^{MEC}, \tau_9^{MEC}\right\}$ that leads to *Connected*, and for *AppActive* $\exists \left\{\tau_2^A, \tau_3^A, \tau_4^A\right\}$ that leads to *AppActive.*
5. In case of handover, for *Connected* $\exists \left\{\tau_{10}^{MEC}\right\}$ that leads to *Connected*, and for *AppActive* $\exists \left\{\tau_5^A\right\}$ that leads to *AppActive.*
6. In case of device inactivity detection or TAU accept or Application initiated disconnect, for *Connected* $\exists \left\{\tau_{15}^{MEC} \vee \tau_{16}^{MEC} \vee \tau_{17}^{MEC}\right\}$ that leads to *Idle*, and for *AppConnected* $\exists \left\{\tau_9^A \vee \tau_{10}^A \vee \tau_{11}^A\right\}$ that leads to *AppIdle.*
7. In case of device initiated new traffic, or TAU request, or Application initiated bearer establishment, for *Idle* $\exists \left\{\tau_{21}^{MEC}, \tau_{18}^{MEC} \vee \tau_{19}^{MEC} \vee \tau_{20}^{MEC}\right\}$ that leads to *Connected*, and for *AppIdle* $\exists \left\{\tau_{12}^A \vee \tau_{13}^A \vee \tau_{14}^A\right\}$ to *AppIdle.*
8. In case of device power off or radio link failure, for *Idle* $\exists \left\{\tau_{22}^{MEC} \vee \tau_{23}^{MEC}\right\}$ leads to *Deregistered*, and for *AppIdle* $\exists \left\{\tau_{15}^A \vee \tau_{16}^A\right\}$ to *AppDeregistered.* Therefore D_{App} and D_N are weakly bisimilar. □

4 Conclusion

In this paper we propose an approach to design APIs of Web Services for MEC. The approach is based on the RNIS provided by the MEC server. Two Web Services are proposed: Device Context Web Service and Device Bearers Web Service. The Device Context Web Service provides applications with information about device connectivity, mobility and data transfer activity. The Device Bearers Web Service provides applications with information about device's active bearer and allows dynamic control on QoS available on device's data sessions. Web Service data structures, interfaces and interface operations are defined. Some issues related to MEC service APIs deployment are presented. The MEC server functionality includes transition between Web Service operations and network events (signaled by respective protocol messages) and maintenance of device state models which has to be synchronized with the Application view on the device state. A method for formal model verification is proposed.

Following the same approach, other Web Services that use radio network information may be designed. Examples include access to appropriate up-to-date radio network information regarding radio network conditions that may be used by applications which minimize round trip time and maximize throughput for optimum quality of experience, access to measurement and statistics information related to the user plane regarding video management applications, etc.

Acknowledgments. The research is in the frame of project DH07/10-2016, funded by National Science Fund, Ministry of Education and Science, Bulgaria.

References

1. Nunna, S., Ganesan, K.: Mobile Edge Computing. In: Thuemmler, C., Bai, C. (eds.) Health 4.0: How Virtualization and Big Data are Revolutionizing Healthcare, pp. 187–203. Springer, Cham (2017). doi:10.1007/978-3-319-47617-9_9
2. Gupta, L., Jain, R., Chan, H.A.: Mobile edge computing - an important ingredient of 5G networks. In: IEEE Softwarization Newsletter (2016)
3. Chen, Y., Ruckenbusch, L.: Mobile edge computing: brings the value back to networks. In: IEEE Software Defined Networks Newsletter (2016)
4. Roman, R., Lopez, J., Mambo, M.: Mobile edge computing, fog et al.: a survey and analysis of security threats and challenges. J. CoRR, abs/1602.00484 (2016)
5. Beck, M.T., Feld, S., Linnhhoff-Popien, C., Pützschler, U.: Mobile edge computing. Informatik-Spektrum **39**(2), 108–114 (2016)
6. Ahmed, A., Ahmed, E.: A survey on mobile edge computing. In: 10th IEEE International Conference on Intelligent Systems and Control (ISCO 2016), pp. 1–8 (2016)
7. Brown, G.: Mobile edge computing use cases and deployment options. In: Juniper White Paper, pp. 1–10 (2016)
8. ETSI GS MEC 003, Mobile Edge Computing (MEC); Framework and Reference Architecture, v1.1.1 (2016)
9. Sarria, D., Park, D., Jo, M.: Recovery for overloaded mobile edge computing. Future Generation Computer Systems, vol. 70, pp. 138–147. Elsevier (2017)
10. Beck, M., Werner, M., Feld, S., Schimper, T.: Mobile edge computing: a taxonomy. In: Sixth International Conference on Advances in Future Internet, pp. 48–54 (2014)
11. 3GPP. TS 36.300 Evolved Universal Terrestrial Radio Access (EUTRA) and Evolved Universal Terrestrial Radio Access Network (E-UTRAN); Overall description; Stage 2, Release 14, v14.0.0 (2016)
12. Fuchun, L., Qiansheng, Z., Xuesong, C.: Bisimilarity control of decentralized nondeterministic discrete-event systems. In: International Control Conference, pp. 3898–3903 (2014)