

# **A Unified Data Repository for Rich Communication Services**

Submitted in fulfilment  
of the requirements of the degree  
Masters in Computer Science  
at Rhodes University

Oluwasegun Francis Sogunle

December 2016

## Abstract

Rich Communication Services (RCS) is a framework that defines a set of IP-based services for the delivery of multimedia communications to mobile network subscribers. The framework unifies a set of pre-existing communication services under a single name, and permits network operators to re-use investments in existing network infrastructure, especially the IP Multimedia Subsystem (IMS), which is a core part of a mobile network and also acts as a docking station for RCS services. RCS generates and utilises disparate subscriber data sets during execution, however, it lacks a harmonised repository for the management of such data sets, thus making it difficult to obtain a unified view of heterogeneous subscriber data. This thesis proposes the creation of a unified data repository for RCS which is based on the User Data Convergence (UDC) standard. The standard was proposed by the 3<sup>rd</sup> Generation Partnership Project (3GPP), a major telecommunications standardisation group. UDC provides an approach for consolidating subscriber data into a single logical repository without adversely affecting existing network infrastructure, such as the IMS. Thus, this thesis details the design and development of a prototypical implementation of a unified repository, named Converged Subscriber Data Repository (CSDR). It adopts a polyglot persistence model for the underlying data store and exposes heterogeneous data through the Open Data Protocol (OData), which is a candidate implementation of the Ud interface defined in the UDC architecture. With the introduction of polyglot persistence, multiple data stores can be used within the CSDR and disparate network data sources can access heterogeneous data sets using OData as a standard communications protocol. As the CSDR persistence model becomes more complex due to the inclusion of more storage technologies, polyglot persistence ensures a consistent conceptual view of these data sets through OData. Importantly, the CSDR prototype was integrated into a popular open-source implementation of the core part of an IMS network known as the Open IMS Core. The successful integration of the prototype demonstrates its ability to manage and expose a consolidated view of heterogeneous subscriber data, which are generated and used by different RCS services deployed within IMS.

## **Acknowledgements**

First of all, I wish to express my sincere gratitude to my supervisors Dr. Mosiuoa Tsietsi and Prof. Alfredo Terzoli for their exemplary mentorship. Our meetings and discussions, their patience and unconditional support made this thesis possible. I am grateful to my friends for their support and insightful comments given on this work. Finally, special thanks to my family for always being close to me.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Service Oriented Architecture . . . . .	2
1.2	Network Services . . . . .	3
1.3	IP Multimedia Subsystem . . . . .	4
1.3.1	Session Initiation Protocol . . . . .	6
1.3.2	Diameter Protocol . . . . .	7
1.4	Goal . . . . .	7
1.5	Scope of Study . . . . .	8
1.6	Thesis Outline . . . . .	8
<b>2</b>	<b>Rich Communication Services</b>	<b>10</b>
2.1	Globalisation Efforts . . . . .	11
2.2	Architecture . . . . .	12
2.3	Specifications . . . . .	13
2.4	Clients . . . . .	15
2.5	Mobility . . . . .	16
2.6	Supported Protocols . . . . .	16
2.7	Enabling RCS Services . . . . .	17
2.7.1	Configuration Service . . . . .	17
2.7.2	Presence Service . . . . .	19
2.7.3	Messaging Service . . . . .	20

2.7.4	File Transfer Service . . . . .	21
2.7.5	Content Sharing Service . . . . .	21
2.7.6	Telephony Service . . . . .	22
2.7.7	Geolocation Exchange Service . . . . .	22
2.7.8	Common Message Store (CMS) . . . . .	22
2.7.9	Common File Store (CFS) . . . . .	23
2.7.10	XML Document Management Server (XDMS) . . . . .	23
2.8	Service Interactions . . . . .	24
2.9	Summary . . . . .	25
<b>3</b>	<b>Understanding User Data Convergence</b>	<b>27</b>
3.1	Architecture . . . . .	28
3.1.1	Front End . . . . .	28
3.1.2	Client Applications . . . . .	29
3.1.3	Ud Reference Point . . . . .	29
3.1.4	User Data Repository (UDR) . . . . .	30
3.2	Categories of UDR Subscriber Data . . . . .	30
3.3	Data Modelling Requirements . . . . .	31
3.4	Ud Interface Requirements . . . . .	33
3.4.1	Access Control . . . . .	33
3.4.2	Protocols . . . . .	33
3.4.3	CRUD Messages . . . . .	34
3.4.4	Subscribe/Notify Messages . . . . .	36
3.4.5	Transactions . . . . .	37
3.5	Critique of the UDC Specifications . . . . .	39
3.5.1	Integration of LDAP and SOAP . . . . .	39
3.5.2	Data Storage Technology . . . . .	40

3.6	Summary . . . . .	41
<b>4</b>	<b>An Overview of the Open Data Protocol</b>	<b>42</b>
4.1	OData Architecture . . . . .	43
4.2	Data Modelling . . . . .	44
4.2.1	Entity Data Model . . . . .	44
4.2.2	Common Schema Definition Language . . . . .	46
4.3	Data Exchange . . . . .	47
4.3.1	Messages . . . . .	47
4.3.2	Data Format . . . . .	48
4.3.3	Payload . . . . .	49
4.4	Service Provisioning . . . . .	49
4.4.1	Capability Negotiation . . . . .	49
4.4.2	Service and Metadata Document . . . . .	50
4.5	Data Manipulation Capabilities . . . . .	50
4.5.1	Create . . . . .	51
4.5.2	Update . . . . .	52
4.5.3	Query . . . . .	52
4.5.4	Aggregation . . . . .	54
4.5.5	Delete . . . . .	55
4.5.6	Callback Mechanism . . . . .	55
4.5.7	Action and Function Imports . . . . .	57
4.6	Extending Producer Capabilities . . . . .	57
4.7	Summary . . . . .	58
<b>5</b>	<b>Data Store Design Practices</b>	<b>59</b>
5.1	Common Data Models . . . . .	60
5.1.1	Hierarchical . . . . .	60

5.1.2	Key/Value . . . . .	61
5.1.3	Relational . . . . .	61
5.1.4	Document . . . . .	62
5.1.5	Column-Oriented . . . . .	63
5.1.6	Graph . . . . .	64
5.2	Data Abstraction . . . . .	64
5.3	Schema Management . . . . .	66
5.3.1	Schema-full Model . . . . .	66
5.3.2	Schema-less Model . . . . .	66
5.4	Transaction Management . . . . .	66
5.5	Fault Tolerance Techniques . . . . .	67
5.6	The CAP Theorem . . . . .	68
5.7	Persistence Model . . . . .	68
5.7.1	Single-Store Persistence . . . . .	69
5.7.2	Multi-Store Persistence . . . . .	69
5.8	Summary . . . . .	70
<b>6</b>	<b>Introducing the Converged Subscriber Data Repository</b>	<b>71</b>
6.1	Design Considerations . . . . .	72
6.1.1	Data Protection . . . . .	72
6.1.2	Formalised Access Mechanism . . . . .	72
6.1.3	Interoperability with External Systems . . . . .	73
6.1.4	Coping with Schema Evolution . . . . .	73
6.1.5	Persistence and Transactions . . . . .	73
6.2	System Overview . . . . .	74
6.2.1	Application FE . . . . .	75
6.2.2	Provisioning FE . . . . .	76

6.2.3	Converged Subscriber Data Repository . . . . .	78
6.3	CSDR Messages . . . . .	81
6.3.1	Create . . . . .	81
6.3.2	Read . . . . .	82
6.3.3	Update . . . . .	82
6.3.4	Delete . . . . .	84
6.3.5	Subscribe . . . . .	84
6.3.6	Notify . . . . .	84
6.4	Summary . . . . .	85
<b>7</b>	<b>Constructing a Prototype</b>	<b>87</b>
7.1	Choice of Implementation . . . . .	88
7.1.1	Maven . . . . .	88
7.1.2	Wildfly AS . . . . .	88
7.1.3	Apache Olingo . . . . .	89
7.1.4	Java Persistence API (JPA) . . . . .	89
7.1.5	JavaServer Faces (JSF) . . . . .	89
7.1.6	Data Stores . . . . .	90
7.2	Implementing Polyglot Persistence . . . . .	90
7.2.1	Creation of the Domain Model . . . . .	90
7.2.2	Mapping the Domain Model across Stores . . . . .	93
7.2.3	Configuring the JPA Interface . . . . .	93
7.2.4	Exposing the Data through JPA . . . . .	95
7.3	Providing Schema Management . . . . .	96
7.3.1	Schema Descriptor . . . . .	96
7.3.2	Data Producer . . . . .	97
7.3.3	Backend Interface . . . . .	98



7.4	Implementing the Data Processing Component . . . . .	99
7.4.1	Request Processor . . . . .	99
7.4.2	Query Handler . . . . .	99
7.5	Implementing Access Control . . . . .	100
7.6	Discussion . . . . .	101
7.7	Summary . . . . .	102
<b>8</b>	<b>Integration within an IMS Testbed</b>	<b>103</b>
8.1	Overview . . . . .	104
8.2	HSS FE . . . . .	105
8.2.1	Description . . . . .	105
8.2.2	Implementation . . . . .	106
8.2.3	Results . . . . .	106
8.3	Telephony and Messaging FEs . . . . .	111
8.3.1	Description . . . . .	111
8.3.2	Implementation . . . . .	112
8.3.3	Results . . . . .	112
8.4	XCAP FE . . . . .	115
8.4.1	Description . . . . .	116
8.4.2	Implementation . . . . .	116
8.4.3	Results . . . . .	116
8.5	Engaging the CSDR . . . . .	119
8.6	Summary . . . . .	121
<b>9</b>	<b>Conclusion</b>	<b>122</b>
9.1	Achieved Objectives . . . . .	123
9.1.1	Review of RCS specifications and its data management policies	123
9.1.2	Review of the UDC standard to extract relevant requirements	123

9.1.3	Investigation of data store design practices . . . . .	123
9.1.4	Creating a Converged Repository . . . . .	124
9.1.5	Evaluation of the design . . . . .	124
9.2	Thesis Contributions . . . . .	124
9.3	Future Work . . . . .	125
9.4	Summary . . . . .	126
<b>References</b>		<b>127</b>
<b>A Additional Tables</b>		<b>134</b>
A.1	RCS Device Configuration . . . . .	134
A.2	Device Configuration Parameters . . . . .	135
A.3	EDM Data Types . . . . .	136
A.4	Olingo Processors . . . . .	137
A.5	Diameter Command Codes . . . . .	138
A.5.1	Sh Codes . . . . .	138
A.5.2	Cx Codes . . . . .	139
A.5.3	Sh Data-Reference AVP Codes . . . . .	139
<b>B Additional Figures</b>		<b>141</b>
B.1	OMA XDM Architecture . . . . .	141
B.2	HSS FE DM . . . . .	142
B.3	XCAP DM . . . . .	142
B.4	Provisioning the CSDR . . . . .	143
B.4.1	Registering the FEs on the CSDR . . . . .	143
B.4.2	Creating Subscription Profiles . . . . .	143
B.5	Redis . . . . .	147
<b>C Configuration Files</b>		<b>148</b>

C.1	HSS FE . . . . .	148
C.1.1	Jetty AS . . . . .	148
C.1.2	Deployment Descriptor . . . . .	148
C.2	XCAP FE . . . . .	149
C.3	Telephony FE . . . . .	150
C.4	Messaging FE . . . . .	150
C.5	CSDR . . . . .	151
C.5.1	Deployment Descriptor . . . . .	151
C.5.2	POM File . . . . .	152

# List of Figures

1.1	Overview of core SOA components. Source: [58]. . . . .	3
1.2	Main components of an IMS core network. Source: [31]. . . . .	6
2.1	A simplified view of the RCS architecture. Source: [52] . . . . .	12
2.2	Core RCS Service Enablers. Source: [52]. . . . .	18
2.3	A matrix of Joyn service interactions. . . . .	24
3.1	UDC Architecture. Adapted from: [10]. . . . .	28
3.2	Relationship between the UDC IMs and DMs. Source: [9]. . . . .	31
3.3	Steps involved in creating a CDM. Source: [14]. . . . .	32
3.4	LDAP and SOAP network stacks. Source: [13]. . . . .	34
3.5	Basic call flow for creating user data in the UDR. . . . .	35
3.6	Basic call flow for querying user data from the UDR. . . . .	35
3.7	Basic call flow for updating user data in the UDR. . . . .	35
3.8	Basic call flow for deleting user data in the UDR. . . . .	36
3.9	Procedure for subscribing to change events on user data. . . . .	37
3.10	Procedure for notifying FEs when events occur on user data. . . . .	37
3.11	LDAP and SOAP Integration. Source: [13]. . . . .	39
4.1	Fundamental Components of an OData Architecture. Source: [27]. . .	43
4.2	An overview of the EDM. Source: [73]. . . . .	45
4.3	The elements of a CSDL Document. Source: [75]. . . . .	47

4.4	Description documents exposed by an OData Producer. Source: [100]. . . . .	50
4.5	Example call flow for creating an OData Entity. . . . .	51
4.6	Example call flow for updating an OData Entity. . . . .	52
4.7	Example call flow for retrieving an OData entity. . . . .	54
4.8	Example call flow for deleting an OData entity. . . . .	55
4.9	Using OData Callback. . . . .	56
5.1	Simple Hierarchical DM. . . . .	60
5.2	Simple key/value DM. . . . .	61
5.3	Simple Relational DM. . . . .	62
5.4	Simple Document DM. . . . .	63
5.5	Simple Column-Oriented DM. . . . .	63
5.6	Simple Graph DM. . . . .	64
5.7	Data abstraction levels. Adapted from: [33]. . . . .	65
5.8	Single store persistence example. . . . .	69
5.9	Polyglot persistence example. . . . .	70
6.1	A high level overview of the CSDR architecture. . . . .	74
6.2	Sequence diagram depicting a generic interaction between an RCS client, an AFE and the CSDR. . . . .	76
6.3	Sequence diagram depicting interactions between a Client, the PFE and the CSDR. . . . .	77
6.4	Components of the CSDR. . . . .	78
6.5	CSDR Authentication and Authorisation. . . . .	79
6.6	Defined CSDR behaviour when creating data. . . . .	82
6.7	Defined CSDR behaviour when fetching data. . . . .	83
6.8	Defined CSDR behaviour when updating data. . . . .	83
6.9	Defined CSDR behaviour when deleting data. . . . .	84

6.10	Defined CSDR behaviour when handling subscription. . . . .	85
6.11	Defined CSDR behaviour when handling notification. . . . .	85
7.1	Overview of the prototype. . . . .	88
7.2	The Domain Model as an Entity Relationship Diagram. . . . .	92
7.3	Snapshot of the CSDR Metadata Document. . . . .	96
7.4	Snapshot of the CSDR Service Document. . . . .	97
7.5	Snapshot of the CSDR Backend Interface. . . . .	98
7.6	Class Diagram of the CSDR Request Processors. . . . .	99
7.7	Class Diagram of the CSDR Query Handlers. . . . .	100
8.1	Overview of the experimental setup. . . . .	104
8.2	UAR from S-CSCF to HSS FE. . . . .	107
8.3	HSS FE and CSDR interaction. . . . .	107
8.4	HSS FE response sent to the I-CSCF. . . . .	108
8.5	S-CSCF MAR request sent to HSS FE. . . . .	108
8.6	S-CSCF MAR request sent to HSS FE. . . . .	108
8.7	S-CSCF sends Cx-SAR message to the HSS FE. . . . .	109
8.8	HSS FE sends SAR response to the S-CSCF. . . . .	109
8.9	HSS FE LIR response sent to the I-CSCF. . . . .	110
8.10	UDR message received by HSS FE. . . . .	110
8.11	HSS responds to the messaging service. . . . .	111
8.12	Tessa accepts the voice call. . . . .	112
8.13	Telephony FE adds Mike's recent conversation to the CSDR call log. . . . .	113
8.14	Telephony FE updates Mike's session data in the CSDR. . . . .	113
8.15	Tessa sends an SMS to Mike. . . . .	114
8.16	Messaging FE updates Tessa's session data in the CSDR. . . . .	114
8.17	SIP File Transfer Request from Tessa. . . . .	115

8.18	Telephony FE updates Tessa's session data with the file transfer activity. . . . .	115
8.19	Creating an XCAP document. . . . .	117
8.20	Creating an XCAP document in CSDR. . . . .	117
8.21	Fetching an XCAP document. . . . .	118
8.22	Document sent to the XCAP client. . . . .	118
9.1	OData Service Convergence. . . . .	126
A.1	Sequence Diagram for first time startup of an RCS device. Source: [52]. . . . .	134
B.1	OMA XDM Architecture. Source: [78]. . . . .	141
B.2	Cx Data. . . . .	142
B.3	Sh Data. . . . .	142
B.4	XCAP Document. . . . .	142
B.5	Snapshot of FE registration page. . . . .	143
B.6	Snapshot of Add SCSCF page. . . . .	144
B.7	Snapshot of Add Application Server page. . . . .	144
B.8	Snapshot of Add Trigger Point page. . . . .	144
B.9	Snapshot of Add Service Profile page. . . . .	145
B.10	Snapshot of Add Visited Network page. . . . .	145
B.11	Snapshot of Add Subscription Profile with IMPU page. . . . .	145
B.12	Snapshot of Add Subscription Profile with IMPI page. . . . .	146
B.13	Starting up a Redis Server . . . . .	147
B.14	Session data generated while Tessa was sharing a file. . . . .	147

# List of Tables

2.1	Chronology of RCS specifications. . . . .	13
2.2	RCS Protocols. Adapted from: [52]. . . . .	17
4.1	Common OData message headers. . . . .	48
4.2	Common OData query options. . . . .	53
7.1	Multi-Store JPA Data Model Mapping. . . . .	93
7.2	JPA Handler Classes. . . . .	95
7.3	JPA and EDM Mapping . . . . .	95
7.4	Definition of the CDM elements. . . . .	97
8.1	Interface operations supported by the HSS FE. . . . .	105
A.1	RCS device configuration parameters. Source: [52]. . . . .	135
A.2	EDM Primitive Data Types. . . . .	136
A.3	Olingo EDM Processors. . . . .	137
A.4	Diameter Sh Codes. Source: [12]. . . . .	138
A.5	Diameter Cx Codes. Source: [11]. . . . .	139
A.6	Diameter Sh Data Reference Codes. Source: [12]. . . . .	139



# Listings

7.1	A version of the polyglot persisence.xml file. . . . .	93
7.2	Maven dependencies for the data stores. . . . .	94
7.3	Primefaces configuration . . . . .	98
7.4	Pseudo-code for the AuthFilter Procedure . . . . .	100
8.1	Diameter interfaces in config-server.xml . . . . .	106
C.1	HSS FE jetty.xml configuration file. . . . .	148
C.2	HSS FE web.xml configuration file. . . . .	149
C.3	XCAP FE web.xml configuration file. . . . .	149
C.4	Telephony FE sip.xml configuration file. . . . .	150
C.5	Messaging FE sip.xml configuration file. . . . .	150
C.6	CSDR web.xml configuration file. . . . .	151
C.7	CSDR pom.xml configuration file. . . . .	152

# Glossary of Terms

3G	Third Generation
3GPP	3G Partnership Project
AA	Access Agnostic
AAA	Authentication, Authorisation and Accounting
ADM	Abstract DM
ADSL	Asymmetric DSL
ADV	Application DM View
AFE	Application FE
API	Application Programming Interface
APN	Access Point
AS	Application Server
AtomPub	Atom Publishing Protocol
AVP	Attribute/Value Pair
BASE	Basically Available, Solid state and Eventually consistent
BSON	Binary JSON
CAP	Consistency, Availability and Partition-tolerance
CBIM	Common Baseline IM
CDM	Consolidated DM
CDR	Call Detail Record
CFS	Common File Store
CMS	Common Message Store
CRUD	Create, Read, Update and Delete
CS	Circuit Switched
CSCF	Call Session Control Function
CSDL	Conceptual Schema Domain Language
CSDR	Converged Subscriber Data Repository
DIB	Directory Information Base
DM	Data Model
DSL	Digital Subscriber Line
EAB	Enhanced Address Book
EDMX	EDM Extensions

EM	Entity Manager
FE	Front End
FHoSS	Fokus Fraunhofer HSS
FTLF	File Transfer Localisation Function
GPRS	General Packet Radio Service
GSMA	Global System for Mobile Communications
HSPA	High Speed Packet Access
HSS	Home Subscriber Server
HTTP	Hypertext Transfer Protocol
I-CSCF	Interrogating CSCF
ICSP	Internet Communication Service Provider
IETF	Internet Engineering Task Force
iFC	Initial Filter Criteria
IM	Information Model
IMAP	Internet Message Access Protocol
IMS	IP Multimedia Subsystem
IP	Internet Protocol
IPSec	IP Security
IPX	IP Packet eXchange
IWF	Interworking Function
JSON	JavaScript Object Notation
LDAP	Lightweight Directory Access Protocol
LTE	Long Term Evolution
MIME	Multipurpose Internet Mail Extensions
MMS	Multimedia Message Service
MNO	Mobile Network Operator
MSISDN	Mobile Station International Subscriber Directory Number
MSRP	Message Session Relay Protocol
NNI	Network-to-Network Interface
OASIS	Organisation for the Advancement of Structured Information Standards
OData	Open Data Protocol
OEM	Original Equipment Manufacturer
OMA	Open Mobile Alliance
OMA-DM	OMA Device Management
ONM	Object NoSQL Mapper
ORM	Object Relational Mapper
OS	Operating System
P-CSCF	Proxy CSCF
PC	Personal Computer
PCRF	Policy and Charging Rules Function
PFE	Provisioning FE
PNB	Personal Network Blacklist

PPI	Polyglot Persistence Interface
PPM	Polyglot Persistence Module
QoS	Quality of Service
RCS	Rich Communication Services
RDM	Reference DM
REST	Representational State Transfer
RFC	Request For Comments
RTP	Real-time Transport Protocol
S-CSCF	Serving CSCF
SDP	Session Description Protocol
SIMPLE	SIP for Instant Messaging and Presence Leveraging Extensions
SIP	Session Initiation Protocol
SMS	Short Message Service
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SPI	Social Profile Information
SpIM	Specialized IM
SQL	Structured Query Language
SUPL	Secure User Plane Location
TLS	Transport Layer Security
TS	Technical Specification
UC	Unified Communications
UDC	User Data Convergence
UDDI	Universal Description, Discovery and Integration
UE	User Equipment
UMTS	Universal Mobile Telecommunications System
UNI	User-to-Network Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VoHSPA	Voice over HSPA
VoLTE	Voice over LTE
VVoIP	Voice and Video over IP
WiMAX	Worldwide Interoperability for Microwave Access
WLAN	Wireless Local Area Network
WSDL	Web Services Description Language
XCAP	XML Configuration Access Protocol
XDM	XML Document Management
XDMS	XDM Server
XML	eXtensible Markup Language
XSD	XML Schema Document

# Chapter 1

## Introduction

Decades ago, the term “Unified Messaging” was used to describe technology that combines different forms of communication into a single inbox. This involved the holistic integration of services such as fax, email, and voice calling to reduce communication breakdown between individuals. Today, the growth of the Internet, coupled with the proliferation of mobile devices, have driven the advancement of communication services such as Skype, Whatsapp, Apple FaceTime, and Google Hangouts. These technologies allow individuals to be reached across devices, which can be mobile or fixed. This allows users to choose how to connect with each other while leveraging the Internet to transcend geographic boundaries. Take, for example, interactions between users, which occur through instant messaging services, voice or video calls over the Internet. This approach to communications ensures that users can be reached through any available device. Thus providing the opportunity for a user to have their messages or voice mails, for instance, delivered in an electronic form on their tablet or smartphone. This concept in modern communication and collaboration is known as Unified Communications (UC) [42, 43, 91].

Rich Communications Services (RCS) [47] is a communications framework that aims to achieve UC for mobile network subscribers. It was mainly developed by a consortium of Mobile Network Operators (MNOs) known as the Global System for Mobile communications Association (GSMA). RCS offers a range of IP communication services over IP including voice and video calling, content and file sharing, location exchange and instant messaging services. These services are discussed extensively in Chapter 2. For an MNO to effectively deliver on the promises of RCS, an IP core network infrastructure is required — this is the IP Multimedia Subsystem (IMS). Section 1.3 gives an overview of the IMS core network.

However, as RCS handles different interactions between its service offerings, user data is generated and managed by different data sources across the network. Some of these data sources use distinct data storage and retrieval mechanisms. Simply put, distinct data sources adopt different data structures and access protocols. This leads to the fragmentation of user data across different storage facilities in the network.

For an MNO to gain more insight about their subscribers, deliver subscriber-tailored RCS services, manage evolving RCS service offerings and maintain a simple network topology, a unified view of these disparate data sets is required.

This thesis investigates the potential use of the User Data Convergence (UDC) standard [10] to achieve data consolidation for deployed RCS services. UDC was proposed by the 3rd Generation Partnership Project (3GPP), a telecommunications standardisation body, as a framework for introducing a logically unique repository into a telecommunications network such as IMS. This repository is expected to cater for different types of subscriber data, which are used by network components. These components support the effective and efficient delivery of the various RCS service offerings. This chapter starts out by briefly discussing some fundamental concepts related to the literature reviewed and contributions of the thesis. This is done to provide a distinction between certain concepts, which are used throughout this thesis. What follows next, is a brief discussion on the goal of this study including an outline of the defined objectives. This is followed by the specification of the scope of the research. Finally, a brief overview of the thesis is presented — highlighting the contributions of each chapter.

## 1.1 Service Oriented Architecture

Service Oriented Architecture, or SOA, is a flexible architectural style that integrates business processes by transforming large-scale applications into a set of autonomous services [81]. In other words, SOA consists of a collection of services that perform specific functions. The term *Web service* describes each functional unit within a SOA environment, which facilitate different business processes. More often than not, these processes are the operations that are carried out by distinct services in a network environment.<sup>1</sup> These web services encapsulate different implementations and interact through common interfaces. Interactions may occur through standardised communication protocols or Application Programming Interfaces (APIs), which are platform-independent. This allows SOA implementations to facilitate loose-coupling among web services in a distributed environment [34, 80]. Further, SOA allows for a client-server architecture that separates presentation, processing and data management functions into different components. The main components of the architecture are the Service Provider, Service Requestor, Broker and Description Document. Figure 1.1 shows the relationship between various SOA components.

A *provider* is a web service component that provides a desired functionality to the requestors. The contents of the provider are published through the service description document. This document is stored in the service registry. A *requestor* acts as a client application that consumes the resources exposed by the provider. It seeks to use the resources shared by the provider. For the sake of this thesis, the requestor is referred to as the *consumer* where relevant.

---

<sup>1</sup>See Section 1.2

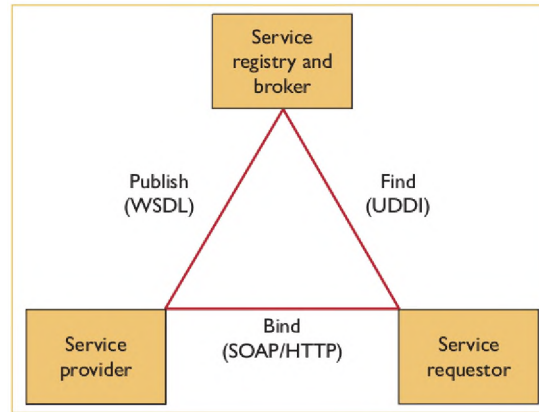


Figure 1.1: Overview of core SOA components. Source: [58].

With the aid of the description document, a *broker* can initiate and manage interactions between a consumer and a provider. For the provider to generate and publish a description document, it uses the Web Service Description Language (WSDL). The broker uses the Universal Description, Discovery and Integration (UDDI) protocol to find and bind providers to requests sent from consumers. The service registry is implemented as an Extended Markup Language (XML) document that contains a list of registered providers on the network. The consumer and provider interact either through the Simple Object Access Protocol (SOAP) or a Representational State Transfer (REST) protocol.

Providers that adopt SOAP, exchange their messages in XML format [62]. A SOAP message structure comprises two distinct parts: a header and body. The header comprises metadata while the body contains the actual data. SOAP can be used with another application layer protocol such as the Hypertext Transfer Protocol (HTTP) protocol. Although REST also adopts the HTTP protocol, it is however, an architectural principle for creating web services [82]. Providers that use REST-based protocols are commonly referred to as RESTful web services. The REST protocol allows consumers to use simple HTTP verbs to access and manipulate data. Examples include POST, GET, and PUT. Chapter 4 gives a comprehensive discussion on these verbs. Thus, consumers interact with RESTful web services through HTTP messages. This differentiates a RESTful service from a SOAP service, which adds an extra XML layer for messaging. This is packaged together with the header, thus forming a SOAP message envelope. Hence, RESTful services provide loose-coupling between HTTP and existing data exchange formats such as plain text and XML, while SOAP does not.

## 1.2 Network Services

In a telecommunications network, different functions and processes are implemented as independent, reusable components, much like web services [65]. This follows from the SOA paradigm as services are loosely-coupled but interact through standardised interfaces. Each service provides a specific functionality with the network. For

example, web services can allow subscribers to place phone calls, and exchange location information, while being charged for using these services. For instance, an example of a network service that can handle the billing of subscribers is the Policy and Charging Rules Function (PCRF). In other words, the PCRF can determine how a subscriber is charged for a corresponding service usage. Within a network, three distinct set of network services can exist — application, data and control services.

An *application service* delivers some added value to enhance communication between network subscribers. Examples of application services include voice mail, instant messaging, file sharing and call barring applications. The MNO can introduce, remove, enable or disable these services based on user-specific needs [60]. These applications can provide additional value to communication services while being independent of each other across the network. Application Server (AS) is a term used to describe the system that enables these services. In essence, the server hosts and executes these applications.

A *data service* is an application service that handles the management of pertinent data. These services expose and provide access control to their data through standardised interfaces [64]. They expose data to other services at different levels within the network. Network components consume data services, which are repositories for user data. A repository for subscriber data is an example of a data service. At the lower level, data services encapsulate distinct storage systems. In addition, an MNO can deploy different data-management functions in the network through such services. Alternatively, these services can provide a centralised repository for frequently used data in the network [22]. Hence, data services can be used to address complex data integration challenges such as heterogeneity and volume of data stored [64].

A *control service* handles the rules and policies, which guide the proper functioning of the network in response to a particular request. The PCRF can be described as a control service. Each control service can also be provisioned by an AS. The network functions that are responsible for initiating and controlling the behaviour of the services rendered to subscribers are themselves control services. Furthermore, packet-based core networks such as IMS, use control services to coordinate how a request is executed to effectively route a message to its required destination.

## 1.3 IP Multimedia Subsystem

IMS is an architectural framework for IP multimedia services deployed on mobile and fixed telecommunication networks [31]. It is standardised and currently maintained by 3GPP. IMS was introduced as part of the standardisation for mobile phones connectivity and service delivery in 3G networks. These can be through fixed, wireless or mobile network access interfaces. Thus, IMS enables the convergence of fixed and mobile networks by being access-agnostic. In other words, a subscriber can connect to IMS through different access networks. Examples include Digital Subscriber Line (DSL) and cable modems for fixed; Wireless Local Area Network (WLAN) and Worldwide Interoperability for Microwave Access



(WiMAX) for wireless; General Packet Radio Service (GPRS) and Universal Mobile Telecommunications System (UMTS) for mobile networks. Consequently, a subscriber can use the application services deployed in the network through multiple access interfaces. This diverges from the traditional approach where a subscriber's device is dependent on a specific network access interface. This allows an MNO to deliver ubiquitous communication services to network subscribers. An access network is integrated within the mobile network through a gateway, which acts as a protocol translator.

Further, IMS represents the portion of the network that controls the core elements necessary for the provisioning and delivery of multimedia application services over IP. These services can integrate the use of audio, video and image data to enhance conversations between subscribers. Section 2.7 discusses some of these IMS offerings which exist in RCS. Mobile network subscribers have the choice of mixing and matching these service offerings during conversations. Hence, the offerings can be added to or removed from a real-time conversation as the subscriber requires.<sup>2</sup> Thus, IMS defines the underlying standards for security, Quality of Service (QoS), resiliency, and inter-operator charging services for these service offerings [25].

More importantly, IMS provides two main functions within a network: session control and service authentication, authorisation and accounting (AAA) [65]. It coordinates interactions between subscribers by establishing, managing and terminating connections with the aid of the Session Initiation Protocol (SIP), a signalling protocol used pervasively throughout the IMS.<sup>3</sup> To achieve session control, IMS provides a set of network services collectively known as the Call Session Control Function (CSCF). There are three types of CSCFs namely: Proxy (P), Interrogating (I) and Serving (S) as shown in Figure 1.2.

- **P-CSCF** is a proxy service that acts as the first point of contact for a subscriber's User Equipment (UE). The P-CSCF inspects all SIP request and response messages to and from the UE. Through this inspection, it ensures that the UE behaves as expected. Although it does not modify the contents of a SIP request, the P-CSCF can enforce the message route in specific conditions. For instance, an emergency registration. For requests that may not be related to registration, the P-CSCF forwards the message directly to the S-CSCF.
- **I-CSCF** determines which S-CSCF should be assigned to a subscriber. It serves as the entry point into the IMS administrative domain. This also applies to external networks with which roaming agreements are established. That is, between the home and visited network of the subscriber. It checks if the subscriber is registered and the location of the subscriber's S-CSCF. Upon confirmation of the agreement, it assigns a S-CSCF to a visiting subscriber. For visited networks, it can forward SIP requests to the home network.
- **S-CSCF** is responsible for service control and user provisioning. It acts as both a registrar and location server. Therefore it handles user registration and

---

<sup>2</sup>See Section 2.8

<sup>3</sup>See Section 1.3.1

saves the location of the user. The S-CSCF is found at the subscriber's home network. It can initiate, manage and terminate a session between the calling and called party. It handles requests from both the P-CSCF and I-CSCF. The S-CSCF downloads a registered subscriber's profile information from the Home Subscriber Server (HSS). It uses this profile information to route request to the appropriate AS through the SIP protocol.

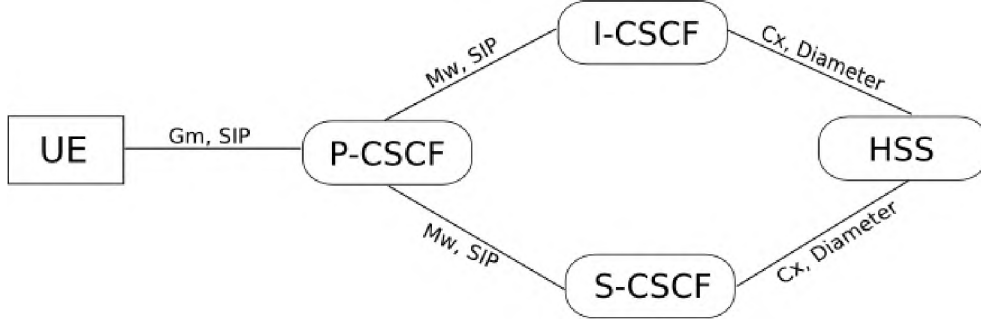


Figure 1.2: Main components of an IMS core network. Source: [31].

Figure 1.2 shows the relationship between the core components of the IMS. The UE is essentially the personal device that the subscriber uses to interact with the network. This can be a wireless or wireline device such as a smartphone, tablet or personal computer (PC). The HSS contains subscription information for subscribed network users. The *Gm* reference point identifies the interface between the UE and the P-CSCF. The *Mw* reference point identifies the interface between the P-CSCF with both I-CSCF and S-CSCF. The *Cx* reference point represents the interface between the I-CSCF and S-CSCF with HSS. Hence, the IMS core network uses two main protocols for signalling and AAA: the SIP and Diameter protocol respectively.

### 1.3.1 Session Initiation Protocol

SIP is an application layer protocol that can establish, modify and terminate multimedia sessions [93]. It is a text-based signalling protocol, which follows a similar model to HTTP. This is evident in its response codes which range from class 100 to 600. It was first standardised in 1999 by the Internet Engineering Task Force (IETF) and has since been adopted as the core signalling protocol of the IMS network [31]. The current version of SIP is version 2.0 which was released in 2002 [93]. The protocol is considered a peer-to-peer protocol but some SIP-enabled devices and network components are capable of functioning as clients, servers or both. SIP can run on most transport layer protocols such as Transport Control Protocol (TCP) and User Datagram Protocol (UDP). It uses the Session Description Protocol (SDP) to negotiate media content, which is used during communication. In essence, the content of an SDP instance is the body of a SIP message.

### 1.3.2 Diameter Protocol

Diameter is used to authenticate and authorise a subscriber to use available application services deployed in the network. The protocol can also be used to account for a subscriber's service usage in the network. This information can be used by billing servers to charge a subscriber for using the available service offerings. Hence, the term AAA protocol. Request and response messages are constructed as a set of attribute/value pairs (AVPs). It provides multiple reference points for different services within IMS to query and manipulate subscriber data. Comprehensive discussions on Diameter protocol can be found in various Technical Specifications (TS) and Request For Comments (RFC). For example, TS 29.228 [11] and TS 29.328 [12] discuss the use of *Cx* and *Sh* reference points within an IMS network. Extensive information on other Diameter reference points and messages can be found in RFC 6733 [44].

## 1.4 Goal

This research attempts to create a unified subscriber data repository for the distinct RCS service offerings. This is to ensure that the various RCS services and other supporting network services have a central facility within the network to query for pertinent subscriber data. Moreover, having a converged view of subscriber data that can be used to support RCS service offerings can greatly reduce the fragmentation of subscriber data across the MNO's network. Thus, the key research question for the thesis is stated as follows:

- *How can a standard-based consolidated data repository be created for RCS?*

However, the following objectives are defined and provide a guideline for what must be accomplished in this thesis:

- i. The review of RCS specifications and its data management policies.
- ii. Review of the UDC standard to extract relevant requirements.
- iii. Investigation of data store design practices, which the converged repository can adopt.
- iv. Designing and prototyping a converged data repository for RCS.
- v. Evaluation of the design by integrating the prototype within an existing IMS network testbed.

## 1.5 Scope of Study

The study focuses on the adoption of UDC to eliminate subscriber data fragmentation, which arises within the RCS platform. Realising UDC implies that subscriber data has to be consolidated into a single view. Thus, achieving this consolidation requires a flexible and consistent way of defining a unique conceptual view for heterogeneous data.

Due to the fact that a mobile network is quite a sizeable construction, and given that the process of ensuring that the solution presented in this thesis works as expected in such an environment would be an exorbitantly lengthy endeavour, there is much need to reduce the scope of the investigation to fit manageable limits. As such, this does not address the following:

- Integration of the repository with network services beyond the IMS core network.
- Data sources existing in circuit-switched (CS) networks.
- Load testing and balancing mechanisms used by the unique repository.
- Service discovery mechanisms adopted by the repository.

## 1.6 Thesis Outline

This thesis spans nine (9) chapters; the remaining chapters are described as follows:

**Chapter 2** discusses attempts to propose the RCS framework as a global standard for next generation communications — as it has been around for some time. The chapter highlights some components of the framework responsible for the delivery of its service offerings. The discussion also presents the various interactions between RCS service offerings, which implicitly rely on data sets hosted on different data repositories. These repositories are presented to illustrate the existence of silo services managing RCS subscriber data.

**Chapter 3** explores the idea of creating a data consolidation solution for the network through a standardised UDC architecture. The various requirements for a centralised and singular repository for heterogeneous subscriber data are discussed in this chapter. It further discusses certain issues, which were not properly addressed by the 3GPP specifications. This leads to a discussion on design spanning the next two chapters.

**Chapter 4** presents an overview of the Open Data Protocol (OData). This thesis proposes the use of OData as an alternative protocol to those suggested by 3GPP, for exposing heterogeneous data. Thus, a unified data repository can be realised, which can provide a converged view of subscriber data and expose it in a common but standardised way.

**Chapter 5** investigates common data storage design practices in order to document different ways in which this repository may be built. To build a data storage system that meets the basic UDC requirements, one should be accustomed to some of the available techniques. This chapter gives a brief overview of some design approaches for data management systems, which can be adopted when building a converged data repository.

**Chapter 6** gives a high level description of the proposed repository. It considers the requirements stated in Chapter 3 and ties together some principles discussed in Chapters 4 and 5.

**Chapter 7** discusses the implementation of a prototypical system that implements the design which is detailed in Chapter 6. It shows how the different components are integrated and explains the method adopted toward the implementation.

**Chapter 8** evaluates the design by integrating the unique repository and its supporting component implementations, within an IMS network testbed. This testbed allows interaction between the distinct components using the various protocols supported by RCS. By so doing, this chapter demonstrates that the design meets the requirements of UDC and is easily extensible to accommodate new RCS service integrations and use-cases.

**Chapter 9** reveals the conclusions culminating from this work and recommendations for further work on unified RCS subscriber data management.

## Chapter 2

# Rich Communication Services

Internet communication services have completely transformed the way in which end users communicate, collaborate and share their daily experiences. They define and offer services, which are available on a myriad of devices including smartphones, tablets, and PCs. Examples of these services include instant messaging, video calling, and social network applications, among others. Some of these services are offered by Internet Communication Service Providers (ICSPs) such as Google, Facebook and Microsoft, which may have no formal contracts with the MNOs. As these Internet services continuously gain popularity, MNOs are seeking new ways to maintain their relevance as core communications providers. RCS, which was previously known as Rich Communication Suite, is one of the technologies that was developed to achieve this goal.

This chapter provides a brief description of the RCS framework, and how its services are constituted. It also highlights the attempts made by MNOs to drive the framework toward global adoption. For a comprehensive discussion on RCS, the reader may refer to specifications found at [49] or relevant literature such as [63, 103]. Therefore, a thorough description of RCS specifications is not presented in this chapter. However, the chapter commences with a discussion that highlights the efforts made by GSMA to encourage the utilisation of RCS on a global scale. This is followed by a description of the components of RCS as a UC service delivery platform. Next, is an account of contributions made to the RCS specifications. The subsequent sections outline the various clients, supported protocols, and mobility mechanisms allowed for RCS implementations. Thereafter, the following section presents the services offered by RCS and their respective enablers. The next part highlights the opportunities for enhancing RCS by leveraging the different service interactions, which occur over the network. The chapter concludes with a summary of the literature discussed.

## 2.1 Globalisation Efforts

GSMA introduced RCS in 2007 and defined it as a “platform that enables the delivery of communication experiences, beyond voice and Short Message Service (SMS), which provides consumers with instant messaging, live video and file sharing across devices on any network” [50]. RCS realises a platform that can create opportunities for better service offerings that give an enriched experience to subscribers through standards-based features. This results from a collaborative effort from leading telecommunications industry players within GSMA, in order to advance the deployment of IMS-based services [57]. Consequently, RCS can work with UEs registered on both fixed and mobile networks.

Further, GSMA is currently working with Original Equipment Manufacturers (OEMs) to enable more devices to support RCS. These collaborations ensure that future smartphones are embedded with RCS clients.<sup>1</sup> This is just as messaging and voice call services have been embedded on mobile phones over the years [36], thus creating a “just there, just works” experience when trying to interact with other subscribers [47]. This also provides a competitive advantage against ICSPs who may not have the benefit of having their applications pre-loaded on mobile phones. The deep integration of the framework with the UE implicitly makes RCS the default mode of communication on the device. Consequently, the range of devices that support RCS services will influence its global outreach — as there are numerous device manufacturers across the globe. However, RCS has enjoyed a rather low level of adoption since its inception compared to the voice and messaging applications of the ICSPs. This has occurred despite the fact that major telecommunication industry players such as Deutsche Telekom and Orange, have been intensively involved with the specification, definition and deployment of RCS services [30].

A study was conducted in South Korea by GSMA in 2011, which involved three companies namely: Korea Telecom, LG Telecom and SK Telecom [48]. They offered a subset of interoperable RCS services to users for free, and this increased the traffic dramatically across the networks. Once charges were introduced for the usage of services such as instant messaging, the number of users and traffic dropped dramatically. The study showed two main factors that can influence the adoption of RCS. First, despite an attractive service suite and appealing user experience, the price is an important factor that determines usage. This means that irrespective of how well structured and effective the service offerings are, most users will still prefer to utilise free services offered by the ICSPs such as Skype, Whatsapp, and Viber. Consequently, effective and efficient business models that can reduce the cost of using RCS services are required, if it is to become pervasive among subscribers. Second, the necessity to have interoperability between different network operators, which is also as important as attractive service offerings. This is the reason for the global outreach of the SMS and voice call applications. Thus, interoperability among operators can have a positive impact when done correctly.

To achieve global interoperability, more MNOs have to commit to the deployment

---

<sup>1</sup>See Section 2.4



of RCS services. For example, a network subscriber can send a message through an RCS messaging application to another subscriber registered on another network. This makes RCS offerings interoperable across mobile telecommunication networks. The GSMA has taken steps to ensure this scale of RCS interoperability by developing a standard specification for MNOs. The document defines the way MNOs inter-work across geographical regions. Thus, GSMA defines the standards-based interactions between MNOs in different regions and countries [50]. Furthermore, since RCS is offered as an interoperable service suite by MNOs, a subscriber can interact with other RCS subscribers with or without using the Open Internet [63], which can occur through any supported service such as messaging. This gives the RCS services a competitive advantage when compared to ICSP offerings, which are heavily dependent on the Open Internet for effective service delivery.

## 2.2 Architecture

RCS provides a service delivery architecture, which leverages the IMS for core network signalling among its distinct services. Thus, IMS coordinates various service interactions and orchestrations for RCS with the aid of the SIP signalling protocol [103]. In essence, IMS performs session control, request routing, and AAA for RCS services. Consequently, IMS brings convergence to the different service offerings, which makes RCS a suitable candidate for delivering next generation UC experience. This is the merit of service deployment over IMS — as it converges different multimedia services. Hence, the architecture allows RCS to provide a common framework for services that are interoperable among MNOs and devices.

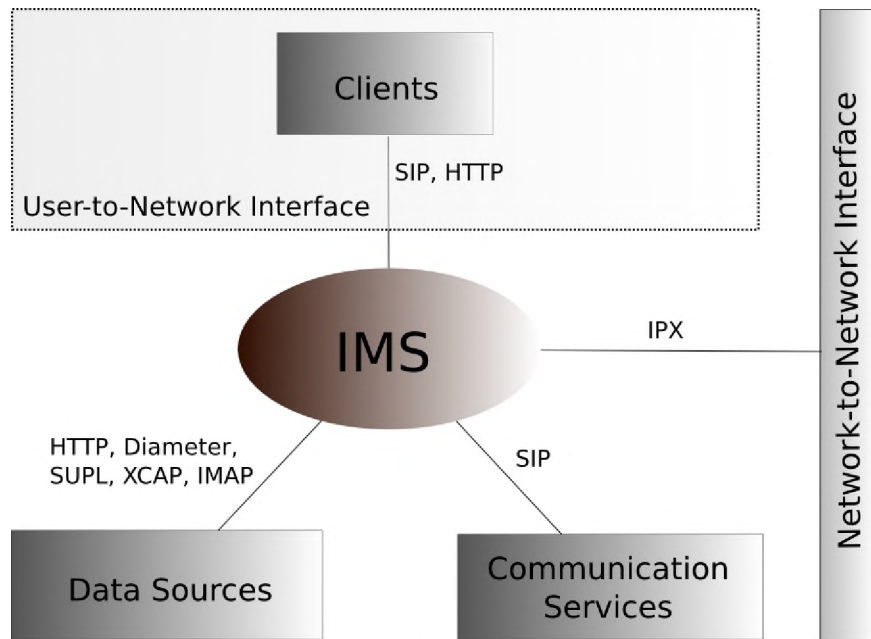


Figure 2.1: A simplified view of the RCS architecture. Source: [52]

Figure 2.1 shows that RCS provides different communication services, adopts



multiple protocols, data sources and supports interoperability among MNO network infrastructures. The clients interact with the IMS using either SIP or HTTP for service interaction and device configuration respectively. The User-to-Network Interface (UNI) represents the demarcation point between the clients and the core network. The Network-to-Network Interface (NNI) allows collaboration between multiple MNOs offering RCS services. This interworking between operators is facilitated by the standard using the IP Packet eXchange (IPX) interface. Additionally, RCS adopts multiple communication protocols for interactions among its service offerings. Each service offering is enabled by an AS deployed in the network. For example, the Telephony AS handles voice call services and Messaging AS handles messaging conversations between subscribers.<sup>2</sup> This also applies to instant messaging and presence services, among others, as depicted in the architecture.

## 2.3 Specifications

The GSMA has chosen an approach that does not define new services, but by efficient packaging and profiling existing services, realises a balanced service offering [52]. Hence, RCS profiles the Open Mobile Alliance (OMA) and 3GPP specifications to provide a suite of multimedia services. These standardisation bodies help facilitate interoperability between different MNOs. Furthermore, GSMA has since defined specifications which are currently in their fifth series. The specifications detail standards and agreements that different MNOs use for deploying RCS under the authority of the GSMA. Hence, by creating these documents it has helped develop common understanding and facilitate interoperability between MNOs [47].

Table 2.1 shows the chronology of RCS specifications including the main features of each document. Each document either enhances or deprecates features from earlier versions. Some of these features are discussed in Section 2.7.

Table 2.1: Chronology of RCS specifications.

Document	Release Date	Main Features
Release 1.0	December 2008	Driven by the concept of an Enhanced Address Book (EAB) on mobile devices. Features include Capability exchange; enhanced messaging; sharing of social presence information; content sharing; video and image share; and file transfer.

---

<sup>2</sup>See Section 2.7

Release 2.0	June 2009	Services were extended toward broadband access for network subscribers. These set of subscribers may decide to use devices such as personal computers to interact with other mobile subscribers. Features include Multi-device environment and the network address book.
Release 3.0	December 2009	Ability to switch to broadband devices if mobile devices are not available. Features include: Further enhancements in social presence information, messaging, content sharing and broadband access.
Release 4.0	February 2011	Included support for Long Term Evolution (LTE). Features include: Messaging evolution; improvements in network address synchronisation, content sharing and broadband network access.
Release 1.2 (RCS-enhanced)	November 2011	A subset of Release 2.0 with a goal of having a lightweight interoperable service extension to voice and message. Features include: An optional activation of social profile Information; No network address book; capability discovery; interoperability; chat; file transfer; content sharing: image and video.
Release 5.0	April 2012	Achieving global interoperability through IPX; Other features include; Capability exchange based on Presence or SIP OPTIONS; Backward compatibility with previous releases.
Release 5.1	November 2013	Focuses on the UNI aspect of the platform. Features include: Deployment options for networks; Enhancements for device configuration group chat and geolocation exchange; personal network blacklist; removal of capability privacy.
Release 5.2	May 2014	Features include: Enhancements on device configurations, one-to-one chat, and file transfer; support for standalone and audio messaging; provisioning of RCS service extensions.

Release 5.3	February 2015	Features include: Enhancements on device configuration enhancements, capability exchange, file transfer and group chat; Ability to separately control transport protocols when using both home and roaming networks; documentation on device registrations and access points (APN); Introduction of the common message store (CMS).
-------------	---------------	---

## 2.4 Clients

RCS supports devices ranging from PCs to smartphones and tablets. However, RCS is a platform and not a service in itself, and thus, there are a variety of RCS clients. The first commercialised RCS client is *Joyn*. It serves as the reference implementation for RCS offerings. Furthermore, MNOs have deployed RCS services over several networks in Europe, North and South America, and Asia [50]. These services may be branded under different names depending on the operator, but the services are branded globally as Joyn.

There have only been two Joyn product releases: Joyn Hot Fixes and Joyn Blackbird Drop [51] which are based on versions 1.2 and 5.1 of other specifications respectively.<sup>3</sup> The clients may be embedded on a mobile device or downloaded from an application store (app store) to the device. Therefore, subscribers may not need to download clients and go through an explicit registration process before the RCS services can be used. The services are automatically provisioned, just as native SMS and voice call applications are readily available once the mobile phone is switched on.

- The downloadable clients utilise two phone books — the one embedded on the host device and the native client address book. These clients can also be downloaded from available application stores onto the device. In this scenario, the RCS contact list may differ from the actual phone book list. This option is made available for the time being in order to allow subscribers to experience the various service offerings.
- Having a pre-installed client on the device depends on the agreements defined between device manufacturers (OEMs) and MNOs. An embedded client can readily interact with native applications such as file explorer and picture gallery that are found on a subscriber's UE. Future smartphones are expected to come with this technology readily installed. This can position RCS as a platform for next generation communication and collaboration.

---

<sup>3</sup>Highlighted in Section 2.3

## 2.5 Mobility

To leverage the convergence of fixed and mobile networks on a single platform provided by IMS, RCS supports a number of access network technologies in a range of device modes. These technologies allow RCS clients to interact with the network regardless of the access technology. They can be categorised into two: cellular and non-cellular networks. RCS supports both LTE and High Speed Packet Access (HSPA) cellular networks. The non-cellular networks include fixed broadband access networks such as Asymmetric DSL (ADSL), cable modem access, and WLAN, among others. In addition, the device modes determine how devices access the network. These modes dictate which technology is used to transport voice traffic over the network. There are four supported modes for RCS-compliant UEs. They include Voice over LTE (VoLTE), Voice over HSPA (VoHSPA), Access-Agnostic (AA) and CS modes.

- **RCS-VoLTE:** A UE is in this mode when it uses VoLTE technology for placing voice calls. This works on devices that support LTE technology. However, IP-based RCS voice and video calls are not allowed for devices in RCS-VoLTE mode. IP-based voice and video calls are commonly referred to as Voice and Video over IP (VVoIP) calls.
- **RCS-VoHSPA:** A UE is in this mode if it supports the HSPA technology for voice traffic. Similar to the RCS-VoLTE mode, it also does not support VVoIP calls.
- **RCS-AA:** A UE is in an AA mode if it can use the various types of network access technologies. This mode allows the transmission of VVoIP calls, but not CS-based calls. That is, this mode does not allow calls which are not placed over IP.
- **RCS-CS:** This mode allows an RCS device to carry voice traffic using legacy networks such as GPRS. VVoIP calls are permitted if they are supported by the MNO.

## 2.6 Supported Protocols

From Table 2.2, it is clear that RCS supports different communication protocols. This section presents an overview of these protocols. RCS clients support both TCP and UDP at the transport layer of the IP stack. Thus, clients use SIP for signalling over either TCP or UDP or both for data (real-time multimedia) transmission. Simply put, RCS clients support both SIP/UDP and SIP/TCP modes. The choice of selection depends on individual MNO implementation strategies.

RCS clients transmit messages during a session, over the network through the Message Session Relay Protocol (MSRP). Thus, with the exception of voice and video, this protocol acts as the medium of information exchange within RCS. The

Internet Mail Access Protocol (IMAP) provides access to message logs and related data sets.<sup>4</sup> RCS clients use the Real-time Transport Protocol (RTP) to transfer voice and video media. Location information is exchanged over the network through the Secure User Plane Location (SUPL) protocol. However, security in real-time communication is achieved through the secured versions of RTP and HTTP. Signalling security is achieved through the Transport Layer Security (TLS) and IP Security (IPSec). In other words, SIP messages can be secured through either TLS or IPSec protocols.

Table 2.2: RCS Protocols. Adapted from: [52].

Protocol name	Description	Transport layer	Secure transport layer/protocol
SIP	A RCS Client-to-IMS Core Signalling protocol	UDP or TCP.	SIP over TLS or IPSec
MSRP	Used for Chat message, media, and file exchange	TCP/IP.	MSRP over TLS
RTP	For exchanging real-time media – voice and video	UDP/IP.	Secure RTP
IMAP	Used to access messaging store	TCP/IP	IMAP over TLS.
HTTP	For RCS Client configuration.	TCP/IP	HTTPS
SUPL	Geolocation positioning.	UDP	TLS

## 2.7 Enabling RCS Services

RCS provides a consolidation of distinct IP communication services into a single platform that leverages the QoS support provided by the MNOs. However, operators deploy different services across the network in order to render the various offerings to RCS subscribers effectively. These services, which can be grouped into application and data services are depicted in Figure 2.2. The IMS session control functions (CSCFs) represent the control services in this case. This section discusses these distinct network services and shows how they facilitate the delivery of RCS offerings.

### 2.7.1 Configuration Service

The first step toward using RCS offerings is for the subscriber to have a configured client on their UE. RCS clients on mobile devices are automatically configured and provisioned to a subscriber through their Mobile Station International Subscriber Directory Number (MSISDN) [55]. The MSISDN is essentially the subscriber's

---

<sup>4</sup>Discussed in Section 2.7.8

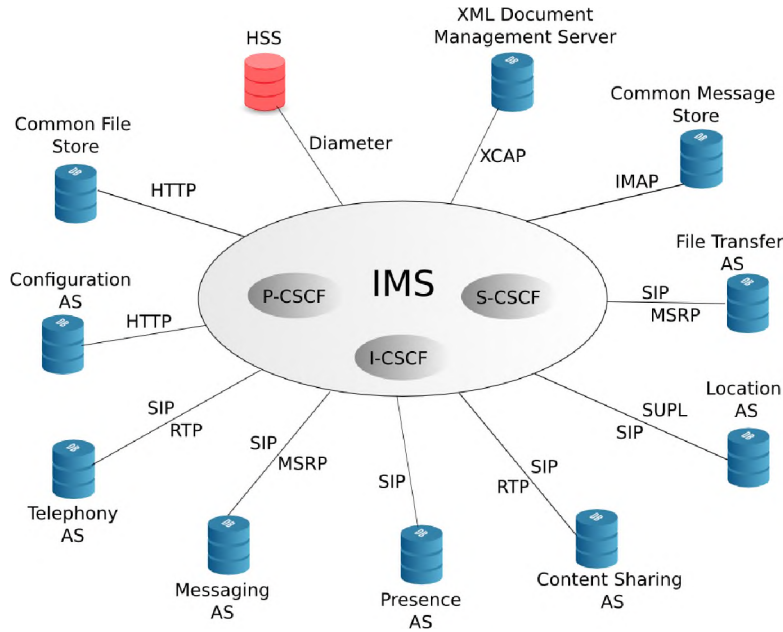


Figure 2.2: Core RCS Service Enablers. Source: [52].

internationally recognised mobile phone number. This procedure is facilitated through an AS within the network and thus, a subscriber’s configuration-related information are managed by this service.

However, it is a transparent process and as such, a subscriber may not be able to interfere with the procedure. A similar technique is adopted for legacy SMS and voice call services, where subscribers can instantly place voice calls and receive messages once the MSISDN is registered with an MNO. Conversely, ICSP offerings such as Skype and Whatsapp tend to configure and provision their services using a mix of different methods such as phone numbers, email addresses, usernames and passwords. This may not always be a transparent process, as the user may have to initiate and provide details for a subscription.

Thus, the common feature for RCS implementations is this unique service provisioning method [52]. Hence, when an RCS-compliant UE is booted for the first time, the initial configuration is automated through network synchronisation.<sup>5</sup> This provides an “out of the box” experience to the subscribers while reducing net operational impact for the operator [36]. A subscriber can use the available service offerings through the client, once the following steps have been completed:

- The device is correctly configured.
- The network creates the IMS Unique Resource Identifier (URI); this is a unique network identity given to the subscriber, which can either be a telephone or SIP URI.
- The network grants the user access to the RCS services via the IMS URI.

<sup>5</sup>Presented in Appendix A.1

These configuration procedures can be triggered either by the client or the network. Client-initiated configuration occurs when the client automatically detects the network. There are two main techniques provided for an RCS automatic device configuration. They include: OMA Device Management (OMA-DM) and HTTP-based configuration.

**OMA-DM** is a procedure used for managing RCS devices registered on the network. It uses management object configuration parameters to set-up an RCS client. An MNO that supports this mechanism creates OMA-DM accounts for the subscribers through the enabling AS. In order to configure the RCS client, a factory bootstrap process is initiated. This involves pre-loading the OMA-DM server address into the RCS device configuration from the factory. On the contrary, the **HTTP-based configuration** technique allows an RCS client to communicate with the network-based configuration server through HTTP. In addition to providing transparency to a subscriber, this method tries to simplify auto-detection mechanism in the network infrastructure. Thus, an RCS device sends an HTTP request to a designated configuration server in order to retrieve the configuration data. Moreover, both OMA-DM and HTTP-based configurations share the same set of parameters.<sup>6</sup> These configurations are stored on the mobile device and cannot be read by the user. Hence, the chosen configuration mechanism determines the communication protocol and parameter encoding between the network and the client.

### 2.7.2 Presence Service

Presence is a capability that indicates the willingness of a user to interact through a particular medium at a given time [20]. It simply relays the interest of a user to interact with a contact. Hence, a calling party can access the presence information of a called party before establishing a dialogue. This influences how and when a prospective caller decides to communicate with a callee. Moreover, it acts as a primary enabler for RCS — since the usage of most services depends on the willingness of the subscriber to communicate.

RCS enables the EAB feature through the presence service capability. The EAB acts as the central point for initiating conversations between subscribers. In other words, the various RCS offerings can be utilised through the EAB. Thus, eliminating the need to search for the featured services on the mobile phone. The EAB allows subscribers to detect communication capabilities of their client and contacts by synchronising with the default address book on the mobile device [57]. This capability discovery mechanism ensures that subscribers are presented with services that will work on their respective devices. Consequently, this guarantees that services shown on the client are supported by the host device.

Further, subscribers can expose personal information such as current activity, mood, and availability, among others to depict their presence status. RCS defines this as Social Profile Information (SPI). Thus, having established a social presence

---

<sup>6</sup>These parameters can be found in Appendix A.2.

relationship with a certain contact, the SPI shall be visible from the EAB. This allows a subscriber to connect with another subscriber by using any of the available messaging and telephony-related services depicted in the SPI.

### 2.7.3 Messaging Service

RCS messaging services are categorised into two: Standalone messaging and Chat. Standalone messaging allows RCS clients to provide features similar to SMS, while Chat renders instant messaging services to a subscriber. Both categories are enabled through a dedicated messaging AS within the network are discussed below:

#### Standalone Messaging

Standalone Messaging denotes an RCS service that supports the delivery of text and multimedia messages through IMS. This service uses a converged message repository based on the OMA Converged IP Messaging (CPM) standard. By using OMA CPM standalone messaging standard, RCS replaces the legacy short and multimedia messaging service (SMS/MMS) standards. OMA CPM removes legacy messaging limitations such as the maximum number of characters allowed per text, usually 160. Further, the standard provides synchronised support for users with multiple devices through its standardised message store.<sup>7</sup>

For interaction with non-RCS compliant devices, OMA CPM facilitates interaction with legacy SMS and MMS services via an Interworking Function (IWF). Thus, enhancing backwards compatibility with earlier messaging standards. To achieve this interworking, RCS defines two modes for standalone messaging: Pager and Large message modes which are used implicitly by the client. When a message is sent in pager mode, the client uses the SIP MESSAGE method, which restricts the message size to a certain maximum. Conversely, large message mode uses a dedicated MSRP channel with SIP to deliver messages without size limitations. The specified maximum for pager mode is 1300 bytes and messages exceeding this size are handled by the client in large message mode.

#### Chat

The chat service enables subscribers to exchange messages in real-time. RCS adopts two OMA standards to handle chat conversations. They are the SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE) and CPM. Chat can be divided into two categories: one-to-one and group chat.

The one-to-one chat service facilitates peer-to-peer dialogue between two subscribers, while a group chat enables interaction between more than two parties. It enhances the messaging experience beyond the basic features found in legacy

---

<sup>7</sup>The CMS, which is discussed in Section 2.7.8



SMS/MMS such as: the delivery and read notifications for sent messages; real-time typing indications; fall-back functionality, which facilitates interworking with legacy messaging services.

Moreover, group chat inherits the basic features of one-to-one chat but provides an enhancement by allowing more than two subscribers to participate in a dialogue. The provisioning of group chat services is optional and therefore, depends on the MNO's implementation of RCS. However, chat services interact with other RCS offerings. For instance, the one-to-one chat service creates a new session within its current session to transfer files between concerned parties.<sup>8</sup>

### 2.7.4 File Transfer Service

A dedicated AS can be provisioned by an MNO to facilitate file transfer activities between subscribers within or without session-based dialogue. The service works often with the RCS messaging services. Therefore, different types of files can be exchanged over a dedicated MSRP channel. In order to send a file through to a recipient, both the relevant file and intended recipient must be selected on the client. If the recipient is found to be unavailable, unreachable or has not registered, the AS provides a *store and forward* functionality. Thus, the file will be stored and forwarded to the recipient when available. In a case where the user has multiple registered devices and automatic acceptance settings are disabled, the transfer request is sent to all the UE's that belong to the recipient. If a file transfer is interrupted under any circumstances, the recipient — through the client — may request the AS to progress from where it stopped. This information is managed within the context of the AS and a Common File Store (CFS) if deployed by the MNO.

### 2.7.5 Content Sharing Service

Content sharing provides subscribers with the ability to exchange pictures and videos in near real-time conditions. Thus, RCS allows a subscriber to share such content during VVoIP conversations. Image sharing is only available during a voice call while video sharing can be done with or without an existing call. Additionally, a subscriber can also share some stored content — retrieved over the network or residing on the UE. Content stored in the network can be accessed through this service and shared during dialogue. The designated AS retrieves the stored content from the appropriate network data store, which may be embedded within the AS itself. The type of content that may be shared depends on the capability of the device. For instance, if the device has a front and rear facing cameras, a subscriber can share content through either of them. However, data are generated while using this service and may not be stored except for store and forward situations.

---

<sup>8</sup>Illustrated in Section 2.8

### 2.7.6 Telephony Service

Telephony services refer to RCS VVoIP services. These services allow an RCS user to place phone calls in audio-only or audio+video modes. They are delivered through a dedicated Telephony AS, which handles a Personal Network Blacklist (PNB) defined by a subscriber. The PNB allows a subscriber to instruct the network to block a list of contacts from reaching them. For example, a PNB containing a list of phone numbers used to regulate prospective voice calls is used by a telephony AS when handling phone calls. The telephony AS can generate other types of data, which may be used by other network services. An example is the CDR, which accounts for the details of conversations held by a subscriber at a given period.

Furthermore, the IP video call shares the same SIP session as the IP voice call. Thus, VVoIP sharing capabilities are mutually exclusive within a client. That is, if both the sender and receiver support IP video call, then it is used to exchange content. Otherwise, RCS invokes the content sharing service. It is noteworthy that IP voice call maintains a separate session when interacting with other services like Standalone Messaging, File Transfer, Content Sharing and Geolocation exchange.

### 2.7.7 Geolocation Exchange Service

This service is used to share an RCS subscriber's geolocation with others in the EAB. This location information might be current location or a proposed meeting spot. Obtaining a subscriber's location data is achieved through the use of the IP-based SUPL protocol [79]. The enabling AS can use the SUPL to keep track of locations in which a subscriber frequently visits. Additionally, map-based tools can leverage this service to determine the current location at a given time. This information is stored and managed within the context of the AS.

However, there are two ways to exchange location information within RCS: Geolocation PUSH and PULL respectively. Geolocation PUSH allows a subscriber to publish their location to their contacts. This information can be embedded in the SPI of the subscriber. Contrarily, Geolocation PULL allows a subscriber to retrieve the location information of another RCS subscriber. RCS enables the PUSH mode by invoking the file transfer service while PULL is enabled through a REST-based API. Hence, a subscriber can send their location and retrieve their contacts through this service.

### 2.7.8 Common Message Store (CMS)

While the services that have been discussed so far are application services, the CMS is one of the data services defined by RCS. As discussed in Section 2.7.3, RCS stores chat and standalone messages in this data store. It acts as a unifying storage for messaging data across all subscriber's devices, which are synchronised over the network. The CMS can also keep the log information of messages sent from and

to a subscriber. This store allows RCS to synchronise messages between multiple clients installed on different UEs, which belong to the same subscriber. Its data is organised in a hierarchical manner and accessed through the IMAP protocol. Hence, a CMS is essentially an IMAP server. RCS specifications [53, 54] explain how the CMS can handle the various types of RCS messages.

### 2.7.9 Common File Store (CFS)

The CFS is also a data service which contains files that may be used in conjunction with the CMS. It can provide both the thumbnail and actual version of a file during a chat session. The CFS relies on the existence of the CMS for effective service delivery. Therefore, to deploy a CFS for RCS, a CMS must have been deployed in the network. In addition, the interaction between the CFS and CMS is enabled through a File Transfer Localisation Function (FTLF) [52]. This localisation function will be invoked by an RCS client when a subscriber wants to download a file stored in the network. Thus, the file transfer AS coordinates the execution of the FTLF. Since the files are transferred across the network through MSRP, a messaging AS may be tasked with managing file transfer services. This, however, is dependent on MNO-specific implementations.

To download or upload a file, the messaging server provides an HTTP URL for the file and another HTTP URL for the FTLF, which are stored in the CFS. The FTLF keeps track of every file transfer request — HTTP URLs — issued by a subscriber. Depending on operator implementation, this can also be managed by the CFS. Although the CFS manages the files being stored, more often than not, the actual data will be kept in a local data store.

### 2.7.10 XML Document Management Server (XDMS)

The XDMS is yet another data service, which can be accessed through the XML Configuration Access Protocol (XCAP) [78]. XCAP is an HTTP-based protocol, which uses the XML Path Language (XPath) to navigate the contents of the XML documents within the XDMS [59]. RCS adopts the OMA standard for this service and thus, is based on the OMA XDM architecture. This can be found at Appendix B.1. Services for configuration, messaging and presence query this data service for their XML documents. Hence, this serves a centralised document management service for RCS. Therefore, RCS services utilise some information used stored in this repository to initiate interaction with other services in the network.

## 2.8 Service Interactions

GSMA specifications have defined ways to enable seamless interaction among RCS service offerings. This is required to deliver an enriched UC experience to subscribers, as illustrated in the matrix in Figure 2.3. For instance, it shows that it is possible to exchange a file (File Share) during a chat (One-to-One Chat/Group Chat) or share recorded videos (Content Sharing) during an IP voice call. Hence, RCS has provided a platform for integrating these distinct services in a seamless manner.

Although, the MNOs can achieve UC with RCS, the actual subscriber data generated and utilised across the network are autonomously managed by different services. Examples include: the messaging service, which stores and queries subscriber data from the CMS; the telephony service, which handles the relevant subscriber data used to deliver VVoIP services; the file transfer service, which uses the CFS to manage stored content; and the HSS, which contains the subscription data — the list goes on.

	Standalone Messaging	One-to-One Chat	Group Chat	File Share	Content Sharing	SPI	IP Voice Call	IP Video Call	Geolocation
Standalone Messaging							✓		
One-to-One Chat			✓	✓	✓				
Group Chat		✓		✓	✓				
File Share		✓	✓		✓				
Content Sharing		✓	✓	✓			✓		
SPI									✓
IP Voice Call	✓			✓	✓			✓	✓
IP Video Call					✓		✓		
Geolocation		✓	✓				✓	✓	

Figure 2.3: A matrix of Joyn service interactions.

Considering that Joyn clients are developed from RCS specifications, other implementations may feature more or less complex service interactions. Nevertheless, services presented in the matrix utilise and generate chunks of subscriber data across the network. For instance, say, two users want to communicate through an RCS IP voice call and both users are registered on the same network. For the network to initiate the call, specific data sets belonging to both users are required. This can include their respective PNB configurations and IMS URIs. The network contacts the telephony AS to check for PNB configurations, which are stored in the XDMS. The HSS is also contacted by the network, in order to validate the IMS URIs supplied by the RCS client. Once the

URIs are validated and PNB conditions are met, the network establishes a dialogue between the users.

Such conversations generate data, which include the CDR, current status and location of the subscribers. However, these data sets are managed by different data services; thus, leading to the fragmentation of subscriber data across the network. These data services adopt different communication protocols for accessing their data sets. For instance, XML documents are retrieved from the XDMS through XCAP; and subscription data from the HSS through Diameter. It is noteworthy that HSS and XDMS, among others, organise their data differently. Hence, different protocols are used to access the heterogeneous data. Additionally, interacting with data silos can increase network traffic, leading to a complex network topology for RCS services. Consequently, it can be a complicated task when trying to consolidate a subscriber's data in the network.

As discussions are ongoing within standardisation communities, regarding the next step to take with RCS, having a common way of managing different types of subscriber data can be beneficial to MNOs. Insights can be gathered from a consolidated view of subscriber data, which are currently managed in silos. Subscriber behaviour and preferences, resulting from service usage, can be mined from a consolidated data repository. This can help operators discover new ways to define and offer attractive service models to subscribers. Defining attractive service models for subscribers can complement the efforts made on the global outreach of RCS.

Moreover, a converged repository can also simplify the network topology and thus, allow RCS services to evolve in the network. Take, for example, the introduction of a service into the RCS platform; instead of providing another local storage facility for the service, subscriber data can be managed by this shared repository. Therefore, MNOs may not have to provide new infrastructure to handle a new set of subscriber data each time a service is introduced. To this end, RCS can leverage an architectural framework, which provides a consolidated data repository for its services, and reduces the signalling impact on a network when trying to obtain an overall view of subscriber data.

## 2.9 Summary

This chapter has discussed the RCS framework, which aims to deliver UC-infused experiences to mobile and fixed network subscribers. Developed by the GSMA, RCS offers a range of IP-based communication services and adopts different communication protocols to deliver these services. In their current form, according to the specifications, the distinct services seamlessly co-interact, use and generate data, which are managed in silos across the network. Hence, obtaining a consolidated view of all these heterogeneous data sets becomes a complex process. The following chapter discusses a standardised approach for creating a converged repository for subscriber data in standard IMS use cases. This discussion serves as the basis for outlining the requirements and specifying the components that will

likely be necessary to implement a converged repository for RCS subscriber data.

## Chapter 3

# Understanding User Data Convergence

The existing services within RCS maintain portions of subscriber data, which are used by other network services to perform their specific roles. The data services expose pertinent data to network services to help in the effective delivery of service offerings. However, as RCS services continue to evolve and new services are introduced into the network, a consolidated approach to managing subscriber data is required. This ensures that there is a consistent way of organising different types of subscriber data within the network. It will also provide MNOs with the necessary insight into subscriber usage patterns and preferences. Moreover, having this approach in place can ensure that services share common data while eliminating fragmentation and duplication of subscriber data across the network.

Since mobile network operators embrace specifications from standardisation bodies, adopting a data consolidation standard is a strong requirement for RCS, so not as to not introduce changes in the way networks behave. UDC is a term coined by 3GPP as a way to define a simple and standard architecture necessary for the consolidation of disparate subscriber data, which are available to numerous data services [8]. This chapter focuses on the 3GPP requirements for introducing a unique and consolidated data repository into an IMS network. The chapter starts out by presenting an overview of the UDC architecture. This is followed by the categories of subscriber data considered within the UDC specifications. The next section explains the data modelling requirements for the unique data repository introduced through UDC. The chapter further discusses the UDC data access requirements. An analysis of UDC and its requirements is given toward the end of the chapter. Finally, the chapter concludes with a brief summary of the main points that have been raised.

### 3.1 Architecture

UDC proposes a layered architecture where user data is separated from application logic [10]. This data is stored in a logically centralised repository that is accessible to network services. Therefore, it provides a logical store for heterogeneous data sources, which organise data differently. This allows the unique repository to share and provision subscriber-related data to heterogeneous services deployed over the network. In addition, UDC supports the existing network infrastructures by ensuring that existing data sources are not eliminated, but rather are transformed into stateless applications. This consequently leads to the creation of data-less network services that depend on a logically unique repository for pertinent data [10]. Its architecture consists of distinct data-less entities known as Front Ends (FEs), a converged logical repository, a single reference point, and external interfaces to other UEs and network services. This is depicted in Figure 3.1.

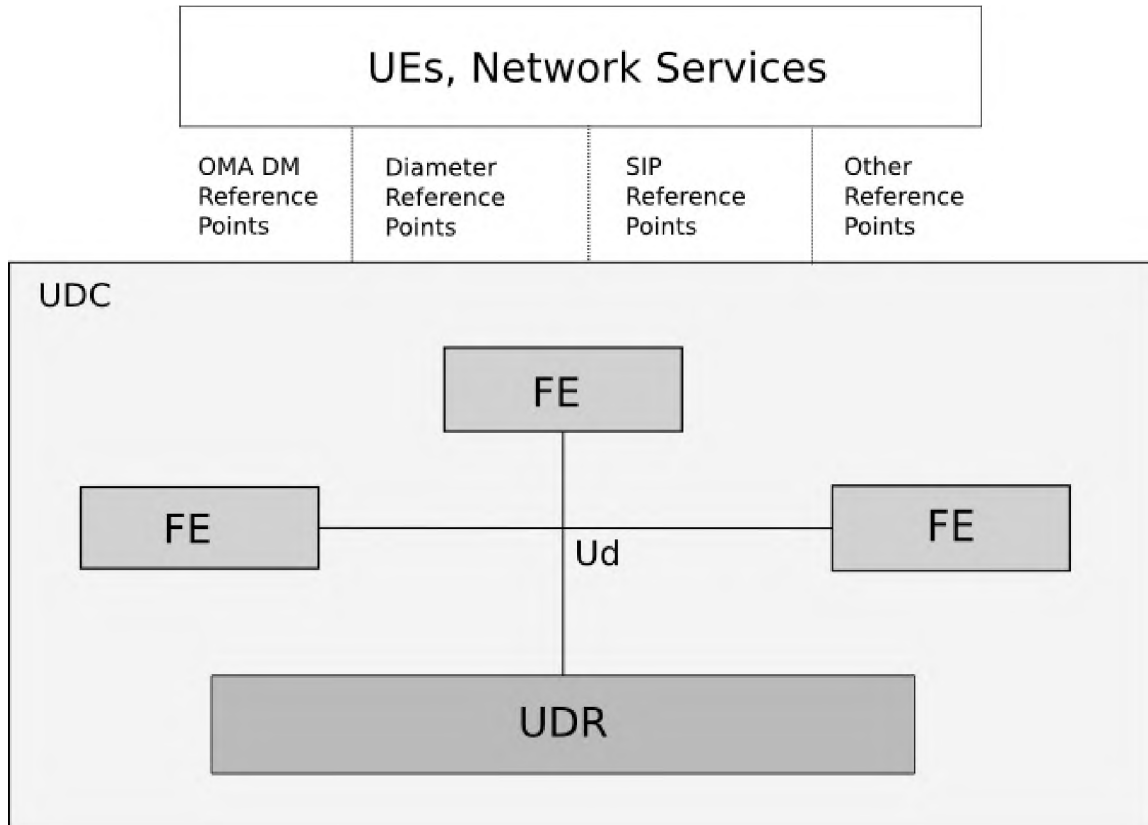


Figure 3.1: UDC Architecture. Adapted from: [10].

#### 3.1.1 Front End

This is a data-less entity that essentially executes application logic and stores its data in the UDR. In other words, network services are transformed to stateless applications which do not permanently store subscriber data. That responsibility is outsourced to the centralised repository. However, FEs can temporarily manage subscriber data but usually discard the data upon completion of the request. For



instance, when a client interacts with an FE that supports SIP, to fulfil the SIP request, the FE may dynamically generate some session-related data, such as request identifier and metadata. The FE can discard this data upon completion of the request. Each FE is an application service, which interacts with multiple clients through different communication protocols. FEs can be grouped into two types:

- **Application FE (AFE)** is an FE that performs the functions of a network service, but in a stateless manner. Examples include: the HSS, Messaging and File Transfer AS. Thus, the type of application handled by the FE determines the behaviour of the AFE. Additionally, the UDC architecture does not impact the existing interfaces between such applications and other network services.
- **Provisioning FE (PFE)** is a specialised FE that is responsible for performing create, read, update, and delete (CRUD) operations on subscriber data that resides in the UDR. With the aid of the PFEs, different user provisioning systems such as self-care systems, which are used by subscribers to interact with the PFE based on different usage scenarios such as self-provisioning. For instance, a user can subscribe to a particular service or activate the service via a provisioning FE deployed by the MNO through a configuration portal. The portal is a self-care system in this case.

### 3.1.2 Client Applications

These are applications, which generate requests toward relevant FEs over the network. They can include RCS clients, SIP ASes, web browsers, and other IMS network services. A typical scenario is when an RCS client contacts the UDR via a relevant FE in order to extract the necessary user data to complete its service execution task. Examples of communication protocols adopted by these applications are Diameter and SIP as shown in Figure 3.1.

### 3.1.3 Ud Reference Point

The Ud Reference Point acts as a unique reference point for all FEs, that interact with the logical repository. Hence, the interface acts as an access mechanism to the repository in the network, which can be used to manipulate heterogeneous data. It provides access to each FE on the basis of a set of access control requirements.<sup>1</sup> In other words, each FE will be permitted to only manipulate data pertinent to its operation. Furthermore, only authorised FEs are allowed to use this interface to perform CRUD operations on subscriber data in the UDR.

---

<sup>1</sup>Discussed in Section 3.4

### 3.1.4 User Data Repository (UDR)

UDR is a data service which provides a single logical storage solution for disparate subscriber data sets. This network entity is situated at the core of the UDC architecture and is, therefore, unique to each FE. It is expected to store different types of data, which were traditionally managed in silos within the network. For example, the silos shown in Figure 2.2 follow this trend. Hence, a UDR should have the capacity to store both transient and permanent subscriber data, which are generated and utilised by different FEs. However, FEs can only to manipulate subscriber data in the UDR through the Ud reference point.

## 3.2 Categories of UDR Subscriber Data

The core idea of the UDC is to create a single logical repository that can encompass different types of data which are stored in silos across the network. For instance, a device's last known IP address, the configuration settings of a registered UE, information pertaining to a real-time voice or video call, among others. As discussed in the 3GPP Technical Report — TR 22.985 [9], the following are the main categories of user data that the UDR can manage.

**Subscription Data** For a user to be provisioned with a service, it is required that the subscriber's UE generates a subscription activation request toward the network. Once the request is granted, the user's data that is related to the subscription is created in the UDR. A user identifier such as the SIP URI and the service profile are both examples of subscription-related data. Data that are stored by FEs for processing their respective business logic are also part of the user subscription data. The duration of this type of data is not specified but limited to whether or not the user is allowed to make use of the service. In essence, subscription data sets can be used for provisioning a service to a user.

**Content Data** This simply refers to the type of data that is created when users make use of an application on their UE. For instance, pictures, videos and audio recordings. Content data may vary significantly in size; video clips usually larger than pictures. The specification recommends that the metadata of such data be converged but the actual content resides outside the UDR.

**Behaviour Data** Such data sets depict the various activities of a particular subscriber over the network. This category of data can be acquired in real-time and therefore will require a rapid and significant amount of processing over the network. An example is CDR which is generated as soon as a call is initiated between subscribers. However, these datasets can be used by an MNO for charging and profiling subscribers based on their behaviours. Thus, the UDR may be designed to cater for these data sets as it assists the MNOs in monitoring subscriber usage patterns of the applications on their UEs.

**Status Data** These are constantly changing data sets that are frequently generated

by FEs when a subscriber makes a status update. These data sets are rarely shared with other services due to their transient nature. Examples of such data include the subscriber's SPI and location information, which changes as the user makes updates or moves from one location to another respectively. Therefore, status data can also help a caller determine which callee's UE is currently available on the network – presence information.

### 3.3 Data Modelling Requirements

TS 32.181 [14] presents a framework for the management of UDR subscriber data. It explains the roles of both an information and a data model. An information model (IM) yields a high-level representation of real-world objects as entities with properties, relationships, permitted operations, and rules and constraints [8]. In contrast, an implementation of the IM is the data model (DM).<sup>2</sup> Figure 3.2 illustrates the relationship between both models. This section highlights the UDC IMs and DMs presented in the report.

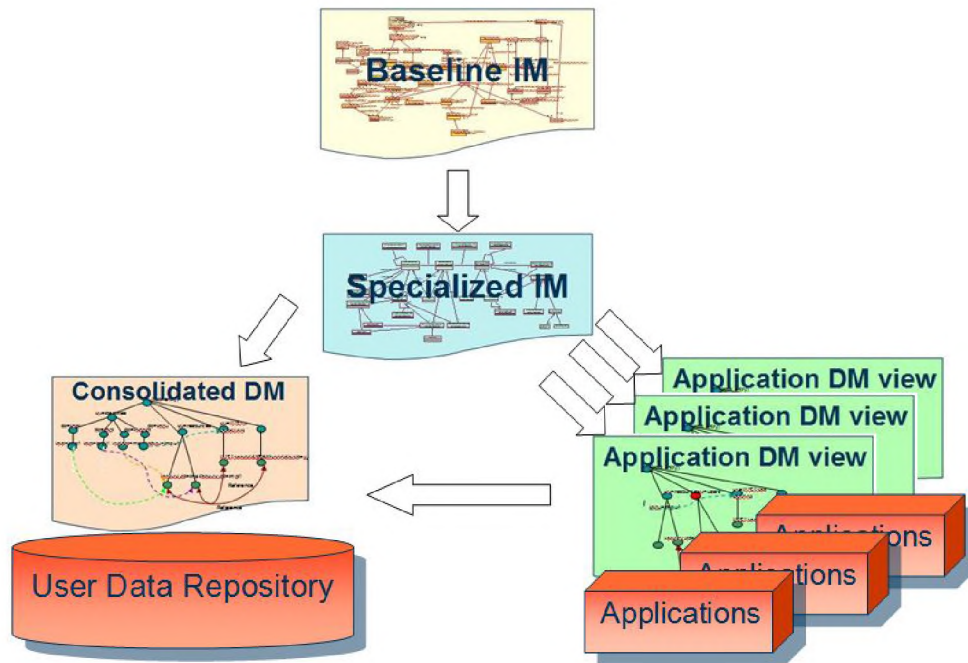


Figure 3.2: Relationship between the UDC IMs and DMs. Source: [9].

**Common Baseline Information Model (CBIM)** To model user data within UDC, a generic IM is created to handle data from distinct FEs. The CBIM provides a flexible data structure and content that is extensible and supports multi-vendor interoperability [15]. The models associated with each application are not specified by the CBIM. Since the CBIM is expected to be “future-proof”, it helps the UDR create an IM that cuts across multiple network services. That is, the model should be able to accommodate both existing and new services as introduced into the network.

<sup>2</sup>Data modelling concepts are further discussed in Chapter 5.

This allows an MNO to flexibly create subscriber data for newly deployed services in the UDR. Therefore, the UDR can have a user with multiple service profiles, a subscriber with multiple user accounts, and a subscriber as a member of a group, among others [8].

**Application Information Model (AIM)** This is an application-specific IM. Thus, it is a distinct IM that is adopted by each FE. An AIM contains two types of data objects for an FE: its service profile and corresponding identifiers.

**Specialised Information Model (SpIM)** The CBIM can be further extended by MNOs to cater for new FEs as required. This extended version is referred to as a SpIM, as it considers the specific applications, functional requirements and the relevant data sets. Thus, a SpIM is MNO-specific and is not standardised by 3GPP. A new SpIM is created when it is combined with an AIM.

**Application Data Model (ADM)** This is a unique data model that belongs to an FE. It is created from the FE's AIM. The UDR exposes relevant view of subscriber data to an FE through the ADM. Moreover, the ADM is a subset of the UDR's Consolidated Data Model (CDM). If the MNO has developed the FE, the AIM is converted into ADM. Otherwise, the CBIM and the AIM are combined to create a new ADM for the Application. This is illustrated in Figure 3.3.

**Consolidated Data Model (CDM)** This is the DM utilised by the UDR, which is created from the aggregation of the SpIM and AIMs. That is, the CDM comprises a set of AIMs and a SpIM. The CDM is the logical data model managed by the UDR. The contents of this model are also not standardised by 3GPP and thus is dependent on MNO implementation preferences.

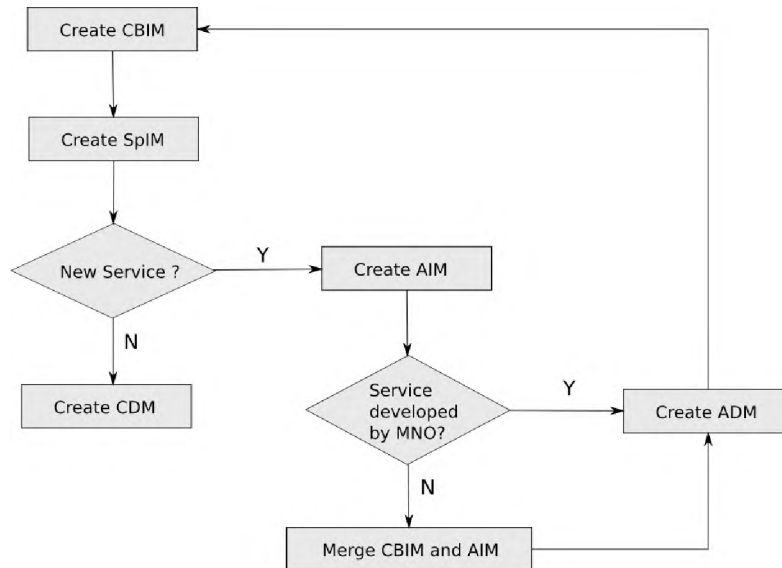


Figure 3.3: Steps involved in creating a CDM. Source: [14].

**Reference Data Model (RDM)** This is a specific DM that enables information exchange over the Ud reference point as discussed in TR 29.935 [7]. It was developed by 3GPP in order to achieve multi-vendor interoperability. It recommends the use of a directory service for managing subscriber data. Hence, hierarchical DMs were

used to represent data objects in the report.

## 3.4 Ud Interface Requirements

This section discusses 3GPP requirements for the Ud reference point presented in TS 29.335 [13]. The report suggests that the UDR provides support for CRUD, and subscription operations. The UDR is also expected to provide access control and ensure integrity in its transactions with the FEs. Additionally, the interface is independent of the DM adopted by the UDR. Hence, there is no tight-coupling with the UDR models. This section is organised as follows: First, the protocols recommended by 3GPP are presented. This is followed by a description of the supported CRUD subscribe/notify messages. After that, there is a brief discussion on access control methods between the FEs and UDR. Finally, an analysis of the interface transaction requirements is given.

### 3.4.1 Access Control

Interactions between the FE and the UDR are managed through the following mechanisms:

**Authentication and Authorisation** The UDC operational workflow is structured such that an FE can effectively interact with the UDR once it has been authenticated and authorised. However, the specifications do not define how this mechanism should be implemented, but flexibility is expected for different implementations. To authenticate and authorise the FEs to interact with the UDR, the following credentials are required — FE Type, FE Identifier, User Identifier, and Request Type.

**Application Data View (ADV)** To further ensure consistency in the CDM, the UDR provides distinct ADVs to each FE. The relationship between the CDM and ADV is illustrated in Figure 3.2. An ADV allows an FE to manipulate allowed portions of the subscriber data within the UDR that they are authorised to. This allows the ADV to create another abstraction over the UDR’s CDM and hence, FEs can not directly manipulate the UDR. For example,  $FE_a$  and  $FE_b$  will have access to common data sets as well as distinct data sets within the CDM. However,  $FE_a$  will not be able to manipulate data sets, which primarily belong to  $FE_b$  unless authorised by the MNO. Notably, the specification [14] does not highlight precisely how this can be achieved using a specific technology.

### 3.4.2 Protocols

As shown in the UDC architecture, network services can only access the UDR through FEs. The specification [13] proposes the use of two protocols for the Ud

interface. They are Lightweight Directory Access Protocol (LDAP) and SOAP. It suggests that the UDR uses LDAP for intensive real-time data processing and SOAP for non-intensive non-real-time processing. For instance, having an FE perform a heavy read on the UDR through LDAP, while non-real-time operations such as batch processing, can be achieved with SOAP. Figure 3.4 shows the network stacks for both protocols.

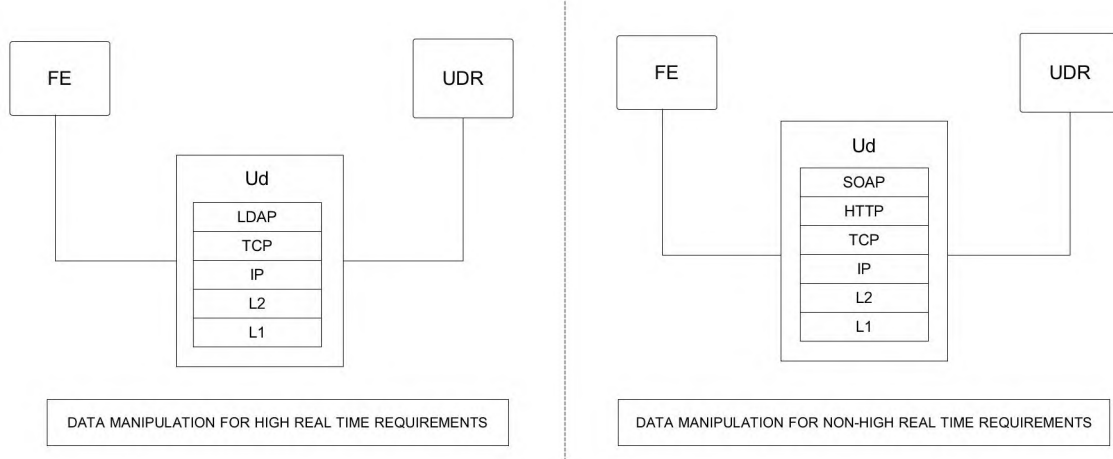


Figure 3.4: LDAP and SOAP network stacks. Source: [13].

### 3.4.3 CRUD Messages

FEs can execute CRUD operations on the UDR either through SOAP or LDAP.

#### 3.4.3.1 Create

To create data, the process is similar to the query operation. An FE sends a create request to the UDR and access control is invoked. Then the new data is stored in the UDR. If there are subscriptions, the UDR notifications to the concerned FEs. The UDR concludes the procedure by sending a status message to the FE.

#### 3.4.3.2 Query

An FE initiates a read request in order to process application logic or provide user data to its client. The UDR checks each request and validates it. At this stage, access control is exercised. If the FE lacks required permissions for the request, a denied notification is sent from the UDR toward the FE. Otherwise, if the subscriber data exists, it is retrieved and converted to a format that is compatible with the FE — this depends on the protocol used. Finally, the data is sent to the FE.

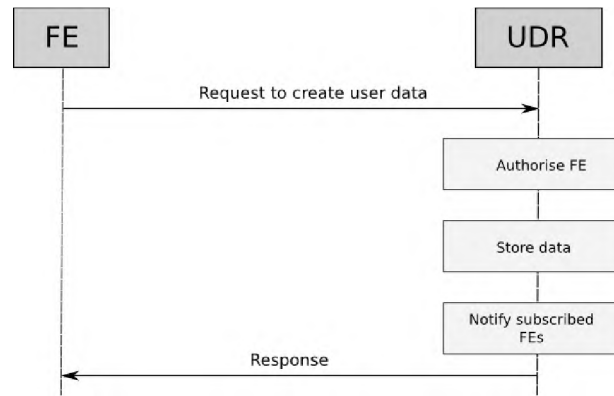


Figure 3.5: Basic call flow for creating user data in the UDR.

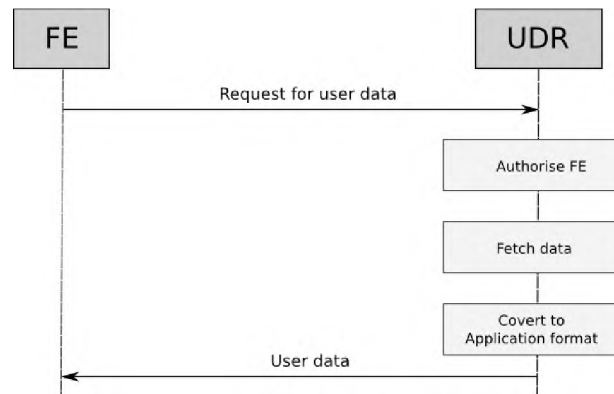


Figure 3.6: Basic call flow for querying user data from the UDR.

### 3.4.3.3 Update

An FE updates data by supplying the identity of the subscriber who owns the data. The UDR invokes access control and uses this identifier to fetch the obsolete data. If the user data is found, it is replaced with the new data and subscribed FEs are notified of the change. Otherwise, a status message is sent to the requesting FE.

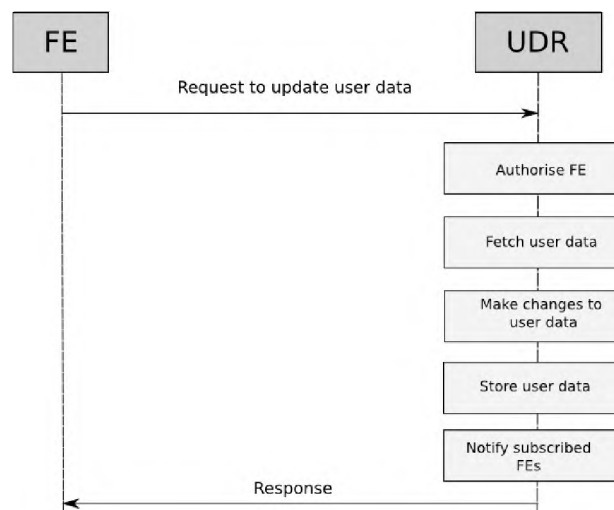


Figure 3.7: Basic call flow for updating user data in the UDR.

### 3.4.3.4 Delete

The procedure here is similar to the previous update operation. Once the UDR finds and deletes the subscriber data, it notifies the subscribed FEs. Thereafter, the process is finalised as illustrated below.

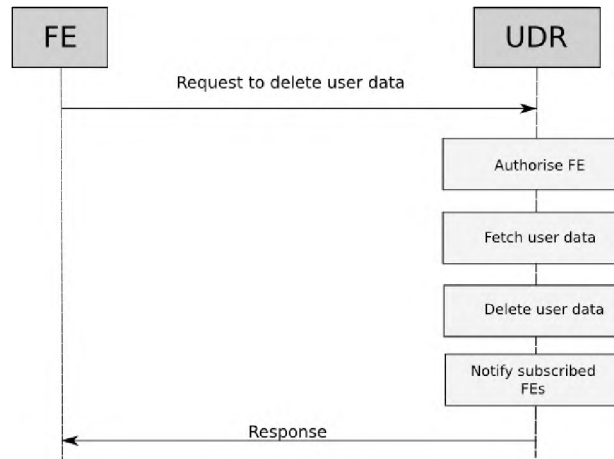


Figure 3.8: Basic call flow for deleting user data in the UDR.

## 3.4.4 Subscribe/Notify Messages

An FE monitors specific data sets by creating a subscription in the UDR. 3GPP recommends the use of SOAP to realise this behaviour. The message sent to the UDR to achieve this is the *subscribe* request while the notification generated when data changes is the *notify* message. Furthermore, subscribe messages are sent using the HTTP POST method. The HTTP POST request will contain a SOAP message envelope. Similarly, Notify is an HTTP message that encapsulates a SOAP envelope.

### 3.4.4.1 Subscribe

For an FE to subscribe to notifications, the following needs to occur: First, send a subscription request message to the UDR. Then the UDR receives this request and stores the subscription information. An acknowledgement message is then sent back to the FE. This is illustrated in Figure 3.9. The subscription request message contains the following information:

- The FE identifier
- The user identity
- Subscription Type; this is to indicate if the request is to subscribe or revoke the subscription.
- Notification Type; this states whether the notification should be sent to all FEs in the same category or to only the requesting FE.



- User data specific information; this includes the user identity, the notification conditions, and the expiry time.

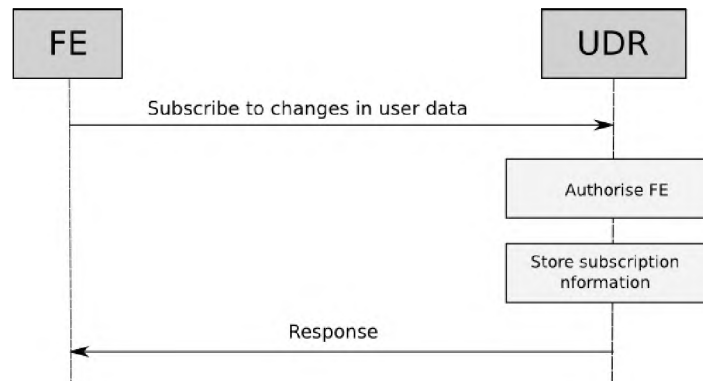


Figure 3.9: Procedure for subscribing to change events on user data.

#### 3.4.4.2 Notify

For an FE to receive a notify message from the UDR, the following occurs: First, an FE or the UDR has successfully modified a specific subscriber data. Second, the UDR checks for FEs (if any) which have been subscribed to that particular data. If the notification condition(s) are met, the UDR selects the appropriate FE(s). Third, the UDR constructs and forwards a notification message to the selected FE(s). This is shown in Figure 3.10:

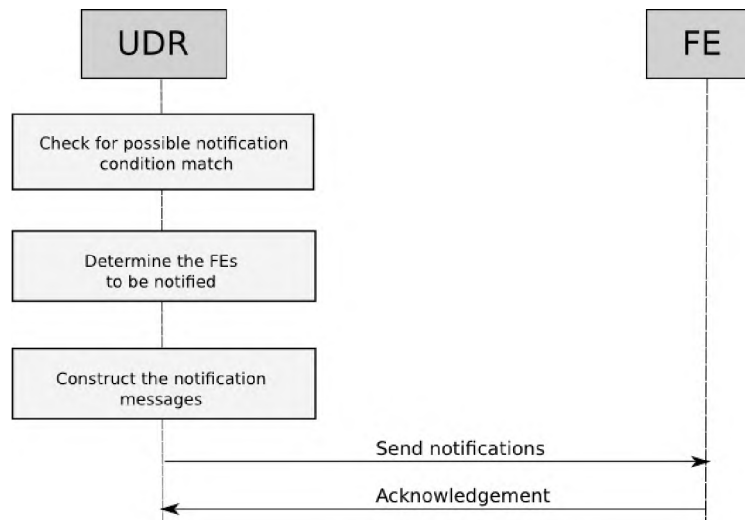


Figure 3.10: Procedure for notifying FEs when events occur on user data.

#### 3.4.5 Transactions

A transaction “is a sequence of operations, which the UDR performs as a single logical unit of work” [10]. An example of a transaction is the registration of a

subscriber in the network which can be stored in a relational data store.<sup>3</sup> Depending on the implementation, the provisioning network service retrieves, stores and makes the subscriber's data accessible to other network services. Therefore, this series of operations can be collectively described as a subscriber registration transaction. In order to ensure data consistency and integrity, transactions are characterised by four properties [77]:

- **Atomicity:** This means that a transaction can only be successful when all the required operations are completed. For example, if the transaction executes all the tasks required to register a subscriber, then the transaction is successful. Otherwise, the transaction fails. There are two terms used to describe what the UDR does when a transaction succeeds or fails: Commit and Rollback. The UDR commits a successful transaction and rolls back an unsuccessful transaction. Simply put, all operations must be successful and are committed or none will.
- **Consistency:** Simply means that changes made to data sets should not have an unpredictable outcome or leave the UDR in an illegal state. Thus, a consistent transaction is one which guarantees the same behaviour when executed each time. Using the registration example, a consistent transaction that a set of rules defined by the storage system is followed when handling a transaction in order to successfully or unsuccessfully register a subscriber. Therefore, a subscriber can be successfully registered once the set conditions are met. This ensures that transactions performed on the UDR should always keep data in a valid or uncorrupted state.
- **Isolation:** When UDR is handling more than one transaction at a given time, then it is said to have concurrent transactions. This property ensures that the UDR manages a transaction as an independent unit of work which is not impacted by other concurrent transactions. For example, two network services trying to manipulate a portion of a subscriber's data, the UDR gives the distinct services an illusion that only their respective transactions are being handled at a given time. Consequently, the effect of those transactions is only committed when the UDR decides to store the results of the transaction.
- **Durability:** In the event of system failure, it should be possible to restore the UDR to the last known consistent state. In other words, once data is committed to the store, it should remain so. This ensures that UDR commits the results of transactions before sending an acknowledgement to the initiator of the transaction — A network service, for instance. In consequence, data loss is less likely when failures coming from system errors, power outages and crashes occur.

---

<sup>3</sup>Relational data stores are further discussed in Chapter 5

## 3.5 Critique of the UDC Specifications

This section discusses some concerns that the 3GPP specifications have not addressed. These shortcomings were highly influential in informing the design motivated by this thesis of a suitable construction of a logically unique repository for heterogeneous subscriber data. First is the integration of the Ud reference point protocols. Second, is the defined data storage technology.

### 3.5.1 Integration of LDAP and SOAP

Section 3.4.2, highlighted the use of LDAP and SOAP for processing real-time and non real-time requests. LDAP clients interact with a directory service [99] for pertinent data. These directory services can be used to store information such as employee contact details for an organisation, or manage subscriber identities within a network. Further, SOAP allows web services to exchange information through HTTP. The integration of both protocols as depicted in Figure 3.11 suggests the need for a protocol converter for non-high real-time requirements. The converter will semantically map CRUD messages between both protocols.

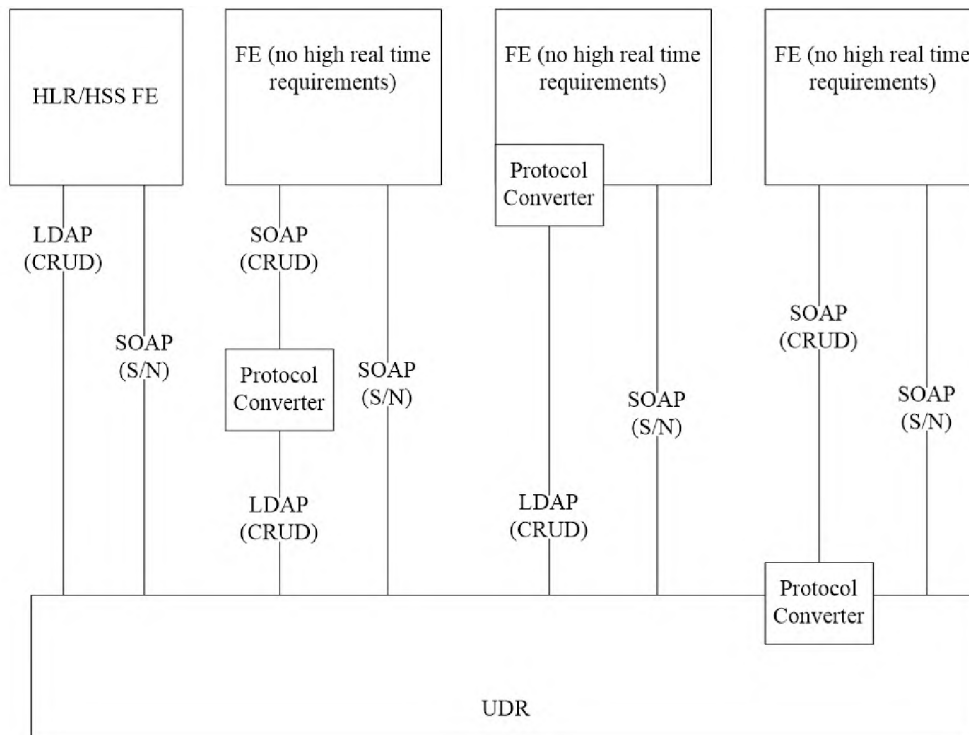


Figure 3.11: LDAP and SOAP Integration. Source: [13].

However, the conversion between SOAP and LDAP can introduce some complexity in request processing through the Ud interface. Take, for example, the work done in [35], which tried to address the protocol conversion issues for application layer protocols. In their case, they used the SIP and SOAP protocol. Notably, SIP and SOAP were designed for different use-cases — network signalling and CRUD

operations respectively. Conversely, LDAP and SOAP integration is quite different, since both support CRUD operations. Nevertheless, the study reported that a protocol converter has to analyse the contents of a request in order to create adequate responses. This analysis can be achieved through a defined set of rules, which can complicate a simple CRUD request. The request can be executed in either of the real-time conditions, depending on the FE's usage. Additionally, even if RCS services generate data in real-time, batch processing of subscriber provisioning requests will be executed against the UDR at specific periods. Therefore, it would be advantageous to the UDR, if the UDR adopted a single communication protocol that could handle both categories of real-time requirements. Leveraging a single protocol, the use of a converter would be eliminated.

Further, the range of CRUD operations which can be performed in the UDR is determined by the adopted Ud interface protocol. A consolidated view of data exposed by the UDR should provide flexible techniques for querying subscriber data. An LDAP directory presents a set of query capabilities to its clients. These query capabilities [99] are defined, such that there is an expected format to which the directory service responds. In other words, innovative ways for searching data may not be allowed. In addition, LDAP is object-oriented, and thus, uses complex search criterion to retrieve objects, their attributes, and the tree of objects within the Directory Information Base (DIB) [106]. The DIB is a collection of the data sets in a directory service. The detailed explanation of the LDAP query construction can be found in Section 4.5.1 of [99]. Alternatively, SOAP adopts an approach where data objects are represented in an XML format. While this provides more flexibility, there is no specific way to present data that has been gathered from heterogeneous sources. Thus, both protocols may not be sufficient for executing advanced query capabilities such as data aggregation. Aggregated data can be effectively mined by MNOs to gain insights into subscriber data. Hence, an alternative protocol is required if the UDR is required to provide such capability.

The Ud interface specification [13] also suggested the use of SOAP for subscribe/notify with CRUD operations. Since SOAP messages are wrapped into an XML envelope, the messages are bulkier when compared to their RESTful counterparts, which use plain HTTP messages with flexible data format payloads [68]. Moreover, using both LDAP and SOAP on the Ud interface implies that the LDAP server will also be a web service, which responds to HTTP requests. Hence, a simplified alternative is to adopt a web service framework that performs both CRUD and subscribe/notify operations with fewer limitations. Such a unifying framework can drive the evolution of UDC as described in TR 23.845 [6], which considers the different scenarios for FE-to-FE interactions through the Ud interface, among others.

### 3.5.2 Data Storage Technology

While the role of the UDR as a logically unique repository for heterogeneous subscriber data has been widely discussed, the actual data store has not been the

subject of much discussion within the UDC specifications. Discussions in the specifications have implicitly suggested the use of a directory service, since using LDAP requires a directory service to handle CRUD requests. In other words, using LDAP implies that the actual data store for the UDR is the directory service. This contradicts the UDC requirement, which expects the reference point to be independent of the storage technology DM; as discussed in Section 3.3.

Moreover, managing data in this manner ensures that the access protocol is coupled with the storage technology. Consequently, changing the directory service can impact the Ud interface. For example, if the UDR evolves and needs to cater for more use-cases and the directory can not adequately meet the new requirements; the MNO can decide to change the storage technology for the UDR and migrate its content to another technology which is not a directory service; inconsistencies can occur while migrating such data sets. Therefore, the storage technology such as the directory service should not be tightly-coupled with Ud interface protocol

Furthermore, LDAP directory services are optimised for frequent reads and fewer writes [105]. RCS services have complex interactions and thus, they read and write data as frequently as the interactions occur. Hence, handling high real-time write operations for heterogeneous RCS subscriber may render a directory service inadequate. This makes it a necessity to find alternative data storage technologies which can handle these use-cases. That is, all use-cases for RCS services. Hence, there is a need to understand the common design practices adopted when using these technologies. This insight can be leveraged when deciding the actual store that a consolidated data repository can utilise.

## 3.6 Summary

This chapter has presented the architectural framework provided by the 3GPP UDC standard. This framework separates subscriber data from data services and outsources the management responsibility to a unique repository defined as the UDR. UDC provides a single interface for accessing heterogeneous subscriber data, known as the Ud reference point. The specifications suggest a tight-coupling of the protocol and the UDR storage technology. That is, using LDAP with a directory service for managing heterogeneous subscriber data. SOAP and LDAP were recommended for this reference point. Furthermore, an overall view of subscriber data can be obtained through the Ud interface. The integration of LDAP and SOAP raises some concerns, which were highlighted by the author in a critique [100]. However, in seeking alternative data access mechanisms (technologies) for the unique repository, a protocol known as the Open Data Protocol (OData) was considered. An overview of OData is presented in the next chapter.

## Chapter 4

# An Overview of the Open Data Protocol

OData is an application layer protocol that is used by RESTful data services. The protocol was introduced by Microsoft Corporation [27]. It is now maintained by the Organization for the Advancement of Structured Information Standards (OASIS) Technical Committee [69]. OData aims to simplify the querying and sharing of data across disparate data services. The OData ecosystem is gradually expanding and some of the systems currently using the protocol can be found at [76]. Furthermore, the most recent specifications discuss the requirements for the fourth version of the protocol, indicating that much work has gone into developing OData into a robust protocol over time, making it mature and reliable. The previous chapter closed on the author's analysis that the LDAP/SOAP protocol proposed by the 3GPP is not well suited to the UDC requirements. This necessitated the search for a more accommodating solution. This search led to the discovery of OData.

This chapter provides an overview of OData by highlighting some of its important features and components. The chapter starts out by discussing the main components of OData architecture, in order to explain precisely how OData can work in practice. The chapter further discusses the OData modelling approach, which is then followed by a section that discusses OData exchanges data over the network. The following section highlights the various service provisioning mechanisms used by OData services. The next section exposes some of the relevant data manipulation operations, which can be performed through the protocol. Thereafter, is a section that presents the aspects of the protocol, which can be extended to provide desired functionalities. This chapter concludes with a summary of the concepts discussed.



## 4.1 OData Architecture

OData, as a communications protocol, adopts a client-server architecture where the client is a service consumer and the server is an OData service. The architecture is divided into three main components: The Data Consumer, Interface and Data Producer. This is illustrated in Figure 4.1 where the Client is the Consumer, the Interface is the protocol, and the Data Source is the Producer.

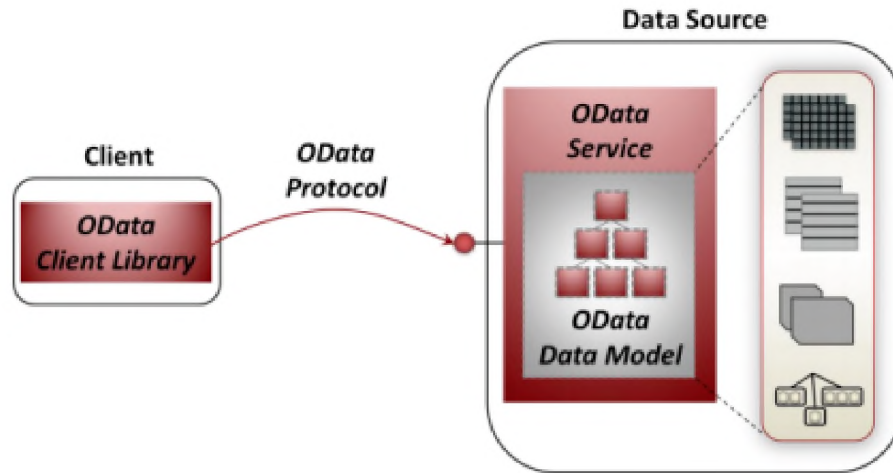


Figure 4.1: Fundamental Components of an OData Architecture. Source: [27].

**Data Consumer** This is a client application that performs CRUD operations on a data producer. It implements a specific OData client library or uses a simple Web browser to interact with the producer. Currently, OData consumers are supported by the following platforms: Java, C++, Objective-C, JavaScript, Microsoft .Net framework and Python [76]. In addition, an OData consumer deployed on different environments such as mobile, desktop and cloud can leverage the common interface exposed by an OData producer for business intelligence and data analytics.

**Interface** This is the medium that an OData consumer uses to interact with an OData producer. For both the consumer and producer to interact, their versions must be compatible.<sup>1</sup> Since the protocol is based on HTTP and the REST architectural design, each unit of data is considered a resource, which is identified by a unique URI. Thus, the following HTTP verbs can be used to manipulate data:

- **POST**; is used to create a new resource.
- **GET**; is used to fetch one or more resources.
- **PUT**; is used to replace an existing resource in its entirety.
- **PATCH**; is used to replace specific properties of an existing resource.
- **DELETE**; is used to remove a resource.

---

<sup>1</sup>Discussed in Section 4.4

**Producer** This is a data service that exposes data to Consumers in a unified and standardised manner. The data producer handles the underlying data store implementation. In other words, the actual data store technology is managed by the data producer. Adopting OData as a medium of communication between different data services can help achieve a network of systems exposing heterogeneous data in a uniform manner. This can facilitate interoperability between data silos existing within a network. Figure 4.1 shows that an OData producer can be packaged as a single data source, which can use different data storage technologies. This follows the SOA principle where services are deployed in a loosely-coupled way. This is what makes an OData data store agnostic as distinct data stores can be used as the underlying storage platform. Hence, the exchange of data between a producer and consumer is not impacted by the heterogeneous data models adopted by the underlying data stores. Consequently, an OData producer can be easily extended and scaled as required.

## 4.2 Data Modelling

An OData producer uses an abstract model known as the Entity Data Model (EDM) [73] for the logical organisation of data. In essence, the OData DM is an abstract model that the producer exposes to its consumers. This OData model can also be referred to as a schema.<sup>2</sup> The OData EDM provides a flexible way for modelling heterogeneous data objects. This allows a producer to model data from heterogeneous sources since each distinct data object can be mapped to a corresponding EDM element. Furthermore, an EDM is defined using the Conceptual Schema Definition Language (CSDL), which is discussed in Section 4.2.2. Hence, there are other components of the EDM that support the defined entity types, which are presented in Section 4.2.1.

### 4.2.1 Entity Data Model

This model describes the structure of data regardless of how it stored at the lower level. Since OData is an HTTP-based protocol, producer resources can be described using the EDM elements. The relationship between these elements is illustrated in Figure 4.2. The main EDM elements are highlighted as follows:

**Primitive Type** These are the basic data types upon which other types are created. A primitive type denotes an individual attribute of a corresponding data object. Examples of this type include Integer, String and Boolean.<sup>3</sup>

**Entity Type** This represents a distinct data object within the EDM. It contains all of the attributes defined for the modelled object. For instance, user subscriptions and CDRs are distinct data objects and they will, therefore, be represented as individual

---

<sup>2</sup>Schemas are further discussed in Section 5.3

<sup>3</sup>Appendix A.3 provides a list of all EDM primitive types



EDM entity types. In addition, the EDM shows existing relationships with other entity types.

**Complex Type** This consists of multiple primitive types. It is used to model data types, which cannot be represented as primitive types but are re-used in entity types. For example, a subscriber’s profile information may have an address attribute. This attribute consists of several attributes such as country, state, and street address. Thus, the subscriber’s address can be adequately represented using a complex type.

**Enumeration Type** This is synonymous with enumerated data types used in some programming languages. It essentially provides a fixed list of attributes with the same primitive types. This is different from a complex type, which allows different types of primitive types.

**Collections** This represents a distinctive grouping of the data types, which have been highlighted — primitive, entity, complex, and enumeration types. A collection of entity types that share similar properties is called an Entity Set. Thus a homogeneous set of EDM data types. The EDM further allows the producer to expose a group of the other types by using a simple *Collections* notation.

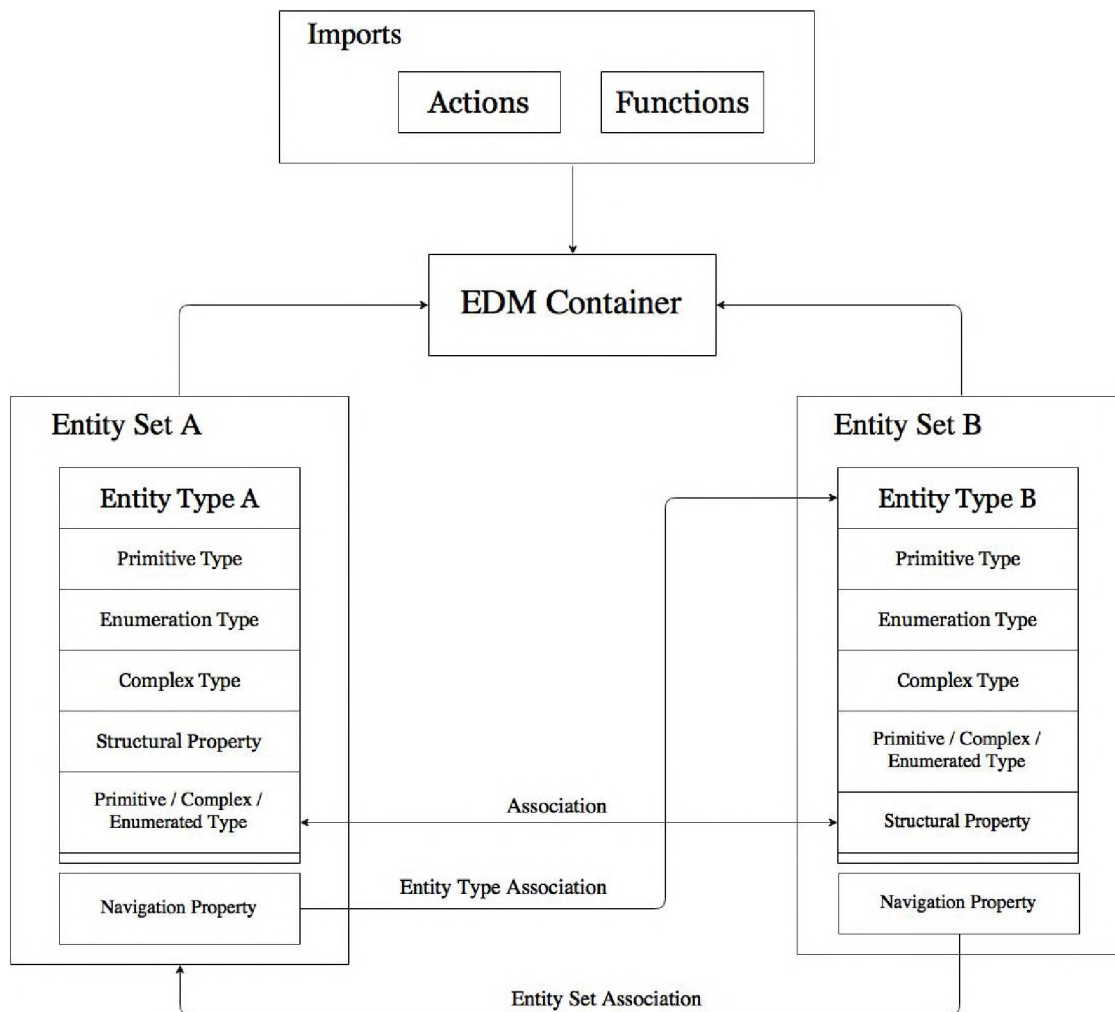


Figure 4.2: An overview of the EDM. Source: [73].

**Structural Property** This is used by the EDM to denote the relationship between attributes of an entity type, which can be declared or dynamic. A declared property is created with an entity type definition. A dynamic property is a nullable property added to instances of an entity type. Thus, a declared property is clearly represented in the EDM and the dynamic property is not represented.

**Navigational Property** Relationships between an entity type and entity sets are represented by this property. They denote specific associative properties of a specific entity type.

**Actions** Actions are operations exposed by an OData producer that may result in side effects in the actual data when invoked by the producer. An action may be bound to a particular resource within the EDM. The first parameter of a bound operation is known as the binding parameter. The value of a binding parameter is that of the resource identified by the URL prior to appending the function name. An action bound to a particular resource as described in Section 11.5.4 of [73] can be invoked by sending an HTTP POST request to an action URL.

**Functions** are operations that do not cause side effects when invoked by the producer. A function can also be bound to a resource within the EDM. As described in Section 11.5.3 of [73], a function can be invoked by using an HTTP GET request to the function's URL.

## 4.2.2 Common Schema Definition Language

The CSDL represents OData's EDM in an hierarchical structure stored in an XML document under specific namespaces. A namespace is a value, which an OData producer uses to prefix the identifiers of its EDM elements. According to OASIS CSDL specification [75], the components of the EDM can be described within the scope of any of two namespaces. The first namespace being EDM Extensions (EDMX) while the other is simply EDM.

Namespace EDMX qualifies the CSDL wrapper and its associated elements and attributes. The CSDL document contains the `edmx:Edmx` root element. The namespace prefix `edmx` can be used to represent the EDM for service packaging. Furthermore, the Namespace EDM qualifies the components of the EDM exposed by an OData producer.

A producer comprises a single EDM that can be distributed over numerous schema definitions. This is shown in Figure 4.3. The schemas, in turn, may also be dispersed over several physical locations while maintaining a single access point to the elements of the EDM. OData producers are characterised by a single CSDL document, which can be accessed by sending an HTTP GET request to the producer in the following format:

```
<service-root>/$metadata
```

The service-root is simply the URL to the deployed OData producer. For example:

```
http://192.168.34.5/<service-name>/
```

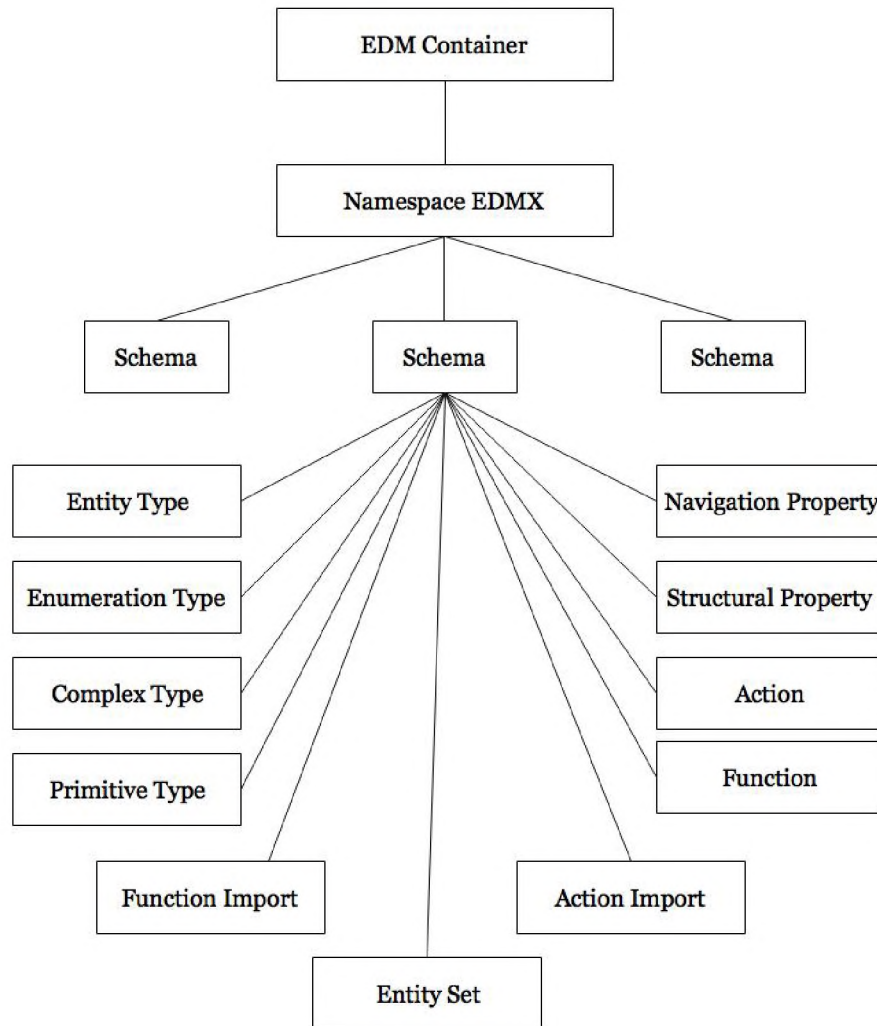


Figure 4.3: The elements of a CSDL Document. Source: [75].

## 4.3 Data Exchange

This section discusses the mechanisms adopted by OData producers to facilitate effective interactions with its consumers over a network. This includes the message types supported, various data formats used over-the-wire, and procedures for determining the size of the transmitted information.

### 4.3.1 Messages

OData messages are grouped into request and responses respectively. Each request comprises header fields and payloads specifying the metadata and body of the

message respectively. Table 4.1 presents the common request message headers found within the OData protocol as specified in RFC 7230 [45] and RFC 7231 [88], which are HTTP technical documents. Further, OData response messages are presented with any of the valid HTTP status codes.

Table 4.1: Common OData message headers.

Header	Function
Content-Type	Specifies the format for an individual request or response. OData specifies value: application/json.
Content-Encoding	Indicates the additional content coding that have been applied to an OData entity.
Content-Language	Indicates the natural language format of the sent OData message.
Content-Length	Specifies the message's length.
OData-Version	Specifies the version of the protocol in use.
Accept	Determines the type of control information used between the Consumer and Producer.
Prefer	It is used on data modification or action requests. The header is included in the request to indicate the Consumer's preferred behaviour for the producer, essentially to hint the Producer.

### 4.3.2 Data Format

OData version 4 supports JavaScript Object Notation (JSON) as the primary format for exchanging data. JSON allows the presentation of data in collections of AVPs. OData's JSON format extends the standardized JSON format [72]. This extension allows the removal of predictable parts of the wire format from the actual payload. The aim is to optimise the data transfer process. Thus, an OData entity — a consumer or producer — recreates the data with the aid of defined expressions.

The other defined format which OData supports is the Atom Syndication Format which is used by the Atom Publishing Protocol (AtomPub) [70]. This is an XML-based format which describes its feeds as a set of entries. The decision to make JSON the primary data exchange is evident in the prioritisation of both formats in the fourth release of the OData specifications. Furthermore, JSON is described an OASIS standard for OData while Atom remained a committee specification [69].

However, there are two ways to change from JSON to Atom if required:

- Setting the \$format parameter in an OData request to Atom
- Setting format value in the Accept header of the OData request.

### 4.3.3 Payload

OData provides a mechanism for controlling the size of information embedded in messages sent between a consumer and a producer. This is done by providing a control value in the Accept header of an OData request. Thus a consumer can provide any of the following values:

- `odata.metadata=minimal`; is used when the consumer prefers a very small wire size. That is, the client is capable of processing data using metadata expressions.
- `odata.metadata=full`; is used when the consumer is incapable of re-creating control information. It is adequate for situations when the computation is more important than wire size.
- `odata.metadata=none`; is used when a consumer does not control the information.

## 4.4 Service Provisioning

This section describes how an OData producer presents consistent behaviour toward its consumer with regard to interactions discussed in the previous section. This includes the definition and validation of the protocol version used by consumers and producers alike, and the service metadata documents exposed which a producer exposes to the consumers.

### 4.4.1 Capability Negotiation

There are different versions of the OData protocol, and thus messages state the version number with the aid of the OData-Version header highlighted in Table 4.1. This means that there are scenarios where the OData producer and consumer are using different versions of the protocol. Such situations trigger an OData capability negotiation mechanism. The consumer specifies the maximum version number it can handle. If the consumer does not specify this value, the producer assumes that the consumer is compliant with a protocol version that is not greater than the one it supports. If the producer cannot validate the specified version number, an error response is sent to the consumer. To guarantee the possibility of successful negotiation, the consumer specifies the lowest OData version that the producer can use to process the request. This allows the consumers and producers to evolve as required. However, enhancements made to the EDM which affects interoperability with existing consumers require a new producer version. Thus, a different service-root URL from the old version of the producer is created to achieve interoperability.

### 4.4.2 Service and Metadata Document

An OData producer presents two well-defined resources to its consumers as illustrated in Figure 4.4. These resources describe the DM of the producer – essentially the content of the producer. They are known as the service and metadata documents. A service document outlines the entity sets, functions and singletons that can be retrieved from the producer. It is used by consumers to navigate the service model in a hypermedia-driven style. The service document is presented once the service-root is visited by a consumer. Moreover, a metadata document defines the service contract between a producer and its consumers. In other words, the consumer of an OData service is presented with pertinent information on how requests can be executed, the expected result structure and in general, how the service can be navigated. Furthermore, using the \$metadata URL tells the OData producer to fetch and render the metadata document to a consumer. However, both documents are fixed service resources as they are statically defined. Dynamic resources are often retrieved and manipulated via provided URLs, which are generated from the information stated in the metadata document.

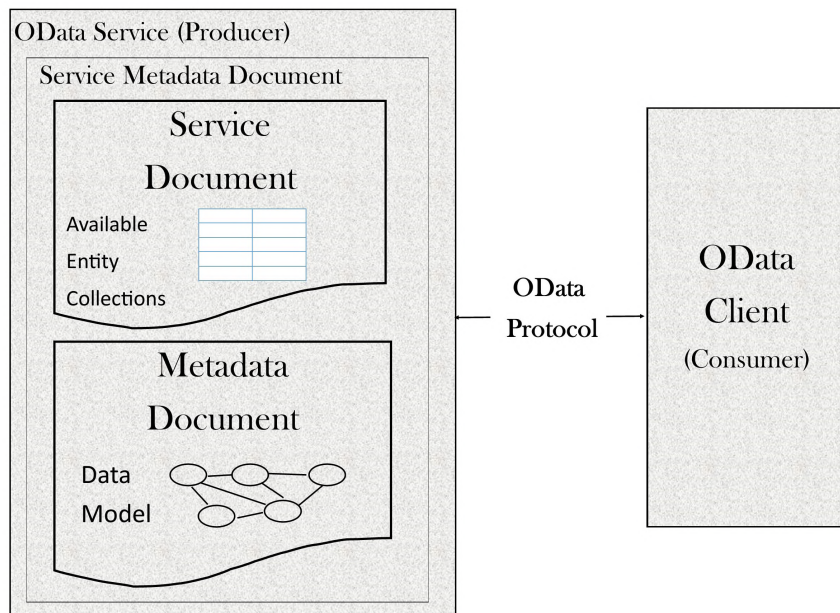


Figure 4.4: Description documents exposed by an OData Producer. Source: [100].

## 4.5 Data Manipulation Capabilities

As a candidate for the Ud reference point, this section discusses the capabilities of the OData protocol. It explains how basic CRUD operations are carried out and further highlights some advanced functionalities that it provides. Although there are other EDM elements, this discussion illustrates the CRUD operations with the aid of an OData entity. An entity is a resource which is defined by an entity type. Thus, an entity belongs to a specific entity set.

Furthermore, there is a special type of entity known as the media entity. It is an entity that represents an out-of-band stream. In other words, it can be used to represent multimedia such as videos and pictures. The media entity may also provide a means to write the media stream to a location. Hence, the media entity is a derivation of the standard entity as it keeps both a media stream and the typical entity properties. The following sections discuss various operations which can be performed on an OData entity.

### 4.5.1 Create

In order to create an OData entity, a consumer sends an HTTP POST request toward the producer. The body of the POST request message carries the actual entity payload data to be stored. For example, adding a new user to a collection of existing users, the POST request carries the new user's information that passes as a valid entity. Upon the completion of this operation, an HTTP 201 Created or 204 No Content response is sent to the consumer, if successful. If the entity to be created is a media entity, a source URL that will be used to read the media stream must be stated. The POST request for a media entity set contains a media type value which is specified in the header of the request. However, upon successful completion of the request, the response message contains both the location header<sup>4</sup> and the 201 created or 204 No content success codes.

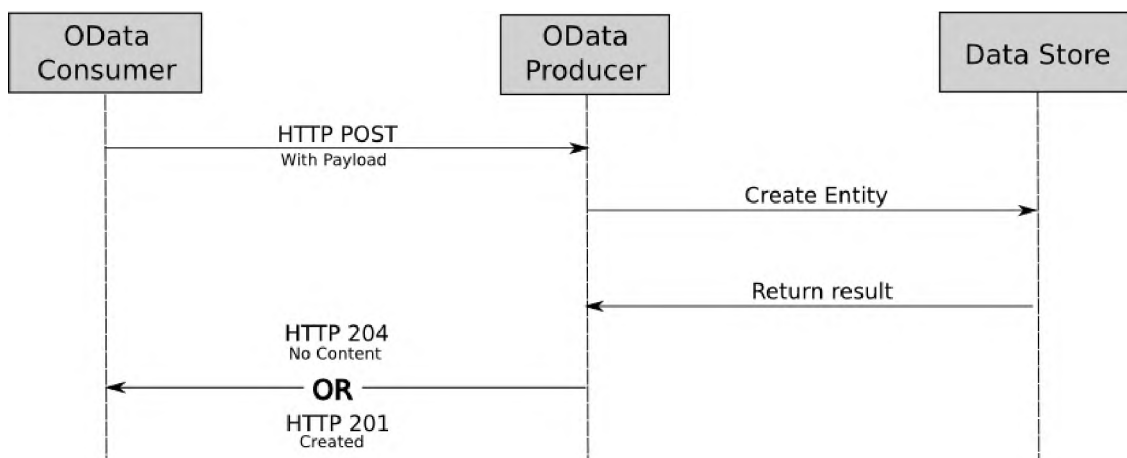


Figure 4.5: Example call flow for creating an OData Entity.

To create a new entity that is linked to an existing entity or multiple entities within a single HTTP request, the entity id(s) of the related entities are specified in the body of the request. This is done via the navigation properties of the entity to be created. Furthermore, to create an entity with relationships to other entities that have not yet been created, a list of definitions within the actual entity definition is composed in the request. This is referred to as a deep insert. However, due to the fact that media entities possess a binary nature, they cannot be represented in this form. Hence, this is not applicable to media entities.

<sup>4</sup> The header specifies the edit URL for the stream.



### 4.5.2 Update

The HTTP PATCH request as specified allows an OData consumer to send update requests to the OData producer. The PATCH request simply collects updated values of properties listed in the request payload, and processes the update, leaving all unspecified properties unchanged. Another alternative is the HTTP PUT request, but this has a significant impact on how data is to be updated. It does not provide the consumer with the option of changing relevant properties, it changes the properties specified and sets the value of any unspecified property to their default values. Further, if an update request is sent to a valid URL that identifies a single entity that is not extant, the OData service handles this as a create request. This phenomenon is referred to as an upsert — derived from a combination of commands update and insert. Upserts are also not allowed for media entities or entities whose key values are generated by the OData producer. A media entity, on the other hand, relies on the PUT request to modify its edit URL which changes its stream.

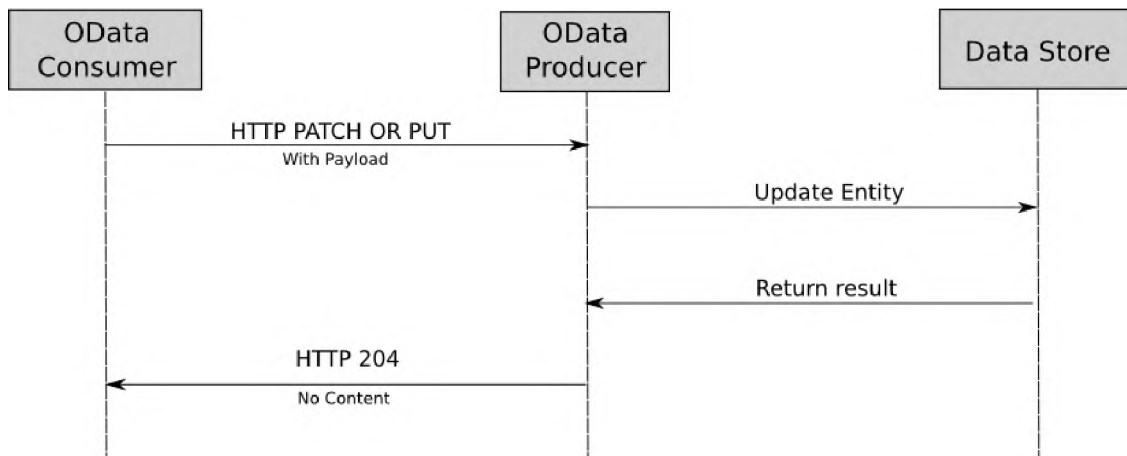


Figure 4.6: Example call flow for updating an OData Entity.

In order to modify or update relationships between entities, their navigation properties are edited. By sending a POST request to an entity's navigation properties collection, a new relationship is established with an existing entity. However, an HTTP DELETE request facilitates the removal of a relationship with another entity. Furthermore, a PUT request to an entity's navigation property containing a single value will, in turn, change the entity linked to it.

### 4.5.3 Query

The HTTP GET request method is used to query an OData producer for a required resource specified with a URL. Requested resources with expired URLs after subsequent advertisements trigger a 410 Gone response message toward the client. Figure 4.7 shows a basic OData query operation. Additional query operations that can be executed against a resource are specified through the system query options. OData provides support for various query options which can



be embedded in CRUD requests. Thus, a producer exposes some of these natively defined query options to its consumer. These options are presented in Table 4.2.

Table 4.2: Common OData query options.

Command	Function
\$search	It confines the result to a set of entities matching the specified search expression.
\$filter	It is used to restrict the data set from a queried dataset.
\$count	It is used to calculate the number of items present in a collection returned by an OData Producer.
\$orderby	Specifies the order in which items are returned from the OData Producer.
\$skip	It defines a non-negative integer X which excludes the first X items from an entity set. Hence, the first entity in the retrieved set begins from position X+1.
\$top	It allows the producer to return an available set of items less than or equal to the value of a specified number.
\$expand	Indicates related entities that must be displayed alongside a principal entity. It is expressed as a navigation property set.
\$select	It is a list of properties, qualified action and function names separated by commas used to fetch data from an OData Producer.
\$format	Specifies the media type of the an OData response message.

Requesting an entity property is done with the HTTP GET request sent to the property URL. This URL is essentially the property name appended to the entity read URL with a forward slash. A property with a null value triggers a 204 No Content response while an unavailable property triggers a 404 Not Found response. Further, to retrieve the raw value of a property, \$value option is used [74]. The producer determines the content type of the response message with the aid of the Accept header field and the \$format query option. The default content type format for single primitive property types is *text/plain*. Binary and Geo types are the only exceptions.<sup>5</sup> A 204 No Content response is received by the consumer if the raw value is null. Also, a 404 Not Found is given for an unavailable property.

Entities that are associated with a particular entity can be retrieved by issuing a GET request to the entity's request URL. The consumer prepends the URL to the navigation property name while being separated with a forward slash. If the navigation property does not exist, a 404 Not Found response is sent by the producer. If there are no related entities to the principal entity, a 204 No Content response is sent to the consumer. Furthermore, a consumer can fetch actual links to other entities rather than the related entities through a GET request with \$ref appended

---

<sup>5</sup>Appendix A.3 presents an overview of all EDM primitive types.

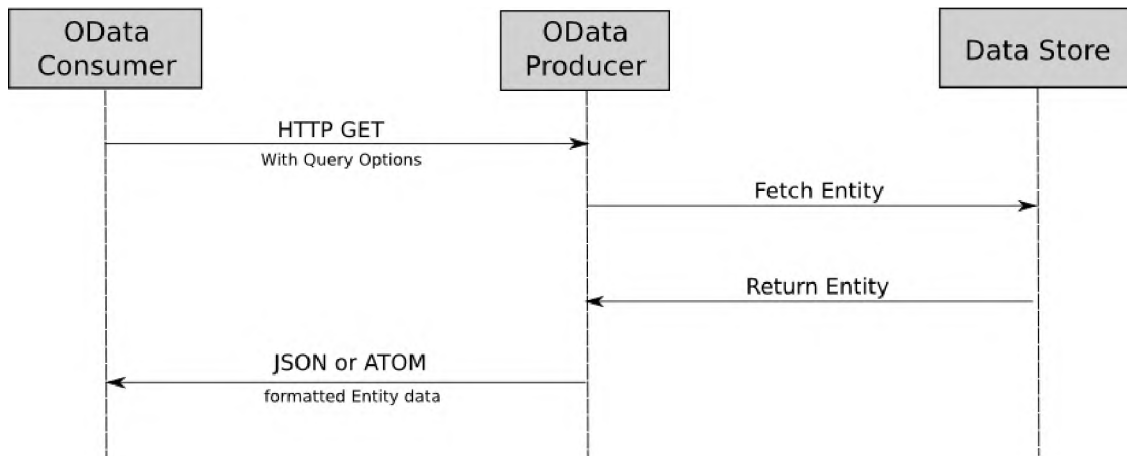


Figure 4.7: Example call flow for retrieving an OData entity.

to the request URL. If the specified URL does not point to an entity or collection of entities, a 404 Not Found message is sent by the producer. However, in situations where the consumer receives the entity id instead of the actual entity, it can resolve the id by sending a GET request to the \$entity resource within the producer. The \$entity resource can be found in the URL relative to the service root. In other words, \$entity is appended to the service root separated by a forward slash. A 404 Not Found is received if the identified entity is not of the specified type or a derivation of the specified type.

#### 4.5.4 Aggregation

Aggregating data with OData simply means that the querying capability of the protocol can be further extended to avoid the possible disarray of a producer as the number of modelled entities grows dramatically. The aggregation feature [71] provides consumers with the ability to do the following without compromising the basic principles of OData:

- A consumer to query aggregated data on top of a conceptual EDM.
- A producer can annotate the type of data which can be accumulated and the aggregation technique that is supported. This allows a consumer to avoid sending invalid aggregation requests.

An aggregation action is triggered with a system query \$apply. The action then applies a set of transformations on the data sets being consolidated. A transformation is a function that maps a data set to itself. For a series of set transformations to be consecutively applied, they must be separated with forward slashes. For example, the output of a set transformation can be used as an input to the next transformation. It is enforced with the use of bindable and composable functions in path segments specified by the OData producer. Aggregated instances of the entities rely on the individual data structure from which they have been

generated. Hence, the structure of the result is consistent with the EDM of the OData producer. They are logical instances of the specified type of the collection described in the resource path of the request. A collection of entities specified within a request also constitute entities which are its aggregated instances. Thus, these entities are either transient or persistent.

In addition, the navigation paths defined in the EDM assists consumers in understanding and navigating entity relationships. However, there may arise a situation where requests need to extend over entity sets with no predefined relationships. A request such as this needs to be executed over a special resource \$crossjoin instead of a specific entity set. The cross join of a series of entity sets is the Cartesian product of the listed entity sets.

### 4.5.5 Delete

Deleting data requires the use of the HTTP DELETE request against an entity's edit URL. A delete request does not have a payload. Also, a singleton entity cannot be deleted. When a delete request is issued against an entity, all existing relationships are implicitly removed together with the entity. An OData service will delete or modify a related entity as specified in a Referential Constraint defined in the service metadata document. A media entity and the actual media attached to the entity are also deleted using the entity's edit URL or that of its media resource.

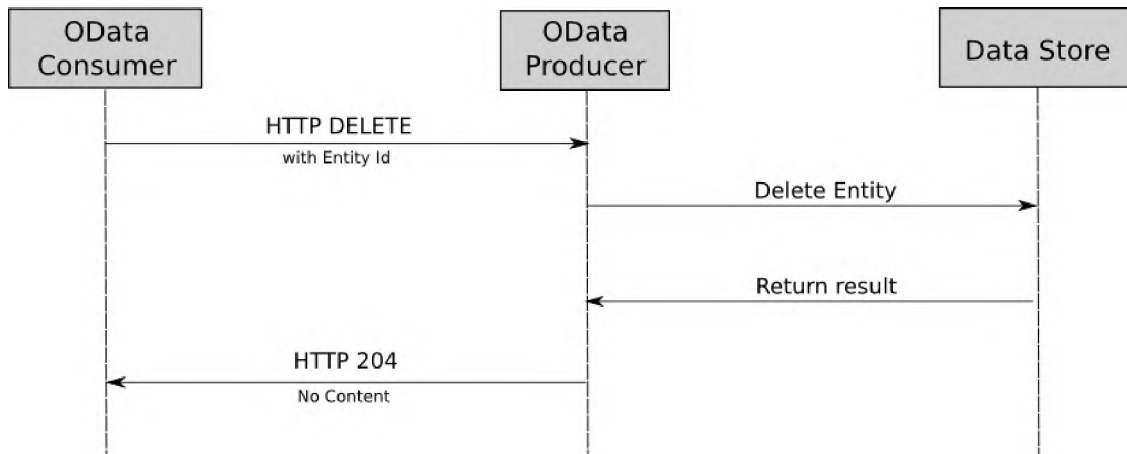


Figure 4.8: Example call flow for deleting an OData entity.

### 4.5.6 Callback Mechanism

An OData producer is capable of providing notifications to consumers on updates made on stored data. As with the subscribe/notify model discussed in Chapter 3, a producer can send updates to subscribed consumers when the triggering conditions for notifications are met. OData producers can publish updates asynchronously, when consumers specify callback parameters through the *Prefer* header. These parameters include: odata.callback, respond-async and odata.track changes.

- Using *odata.callback*: A consumer requesting updates invokes an HTTP GET request. The request is accompanied by *odata.callback* parameter as shown below:

```
Prefer: odata.callback;
url="<service-root>/<notification-id>"
```

- Using *odata.track-changes*: This parameter allows consumers to track changes through delta links. Delta links are “opaque, service-generated links that are utilised by the consumer to fetch the subsequent changes that had been applied to a result” [73]. When a producer returns an entity set, delta links can be used to track changes which have occurred in the set.
- Using *respond-async*: This parameter can be used in conjunction with the *Wait* preference header parameter. Thus, a producer can notify a consumer based on the wait-time specified, as shown below:

```
Prefer: respond-async, wait=10
```

Furthermore, query options are embedded within the request in order to retrieve updates only on the entity types that have changed. If the producer grants the request, it stores the callback URL and sends a 202 Accepted response to the consumer. However, the producer includes a Preference-Applied header into the response to signify that the previous entity set has changed. Once notified, the consumer uses the status monitor resource from the Location header of the previously returned 202 response to retrieve the results. Figure 4.9 depicts a callback scenario triggered by an update made by Consumer<sub>B</sub>. The producer notifies Consumer<sub>A</sub>, which initially made the callback request.

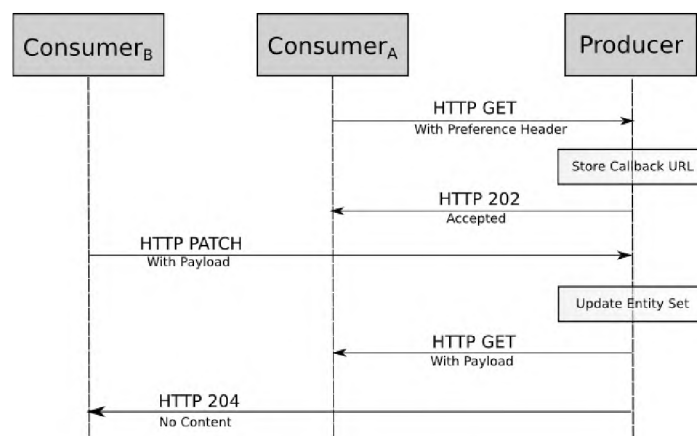


Figure 4.9: Using OData Callback.

### 4.5.7 Action and Function Imports

An import allows a consumer to invoke the functions and actions defined in the EDM. It is important to note that functions and actions provide logical views for defined subsets of the EDM. However, since functions and actions can be bound to EDM resources, an import will interact with the related EDM elements. An import also allows unbound actions and functions to be called directly from the `<service-root>`.

To invoke a function, a consumer sends a GET request to the function URL. However, to invoke a function through an import, the consumer sends a GET request to a URL that identifies the particular function. An example is shown below, where subscribers is the entity set name and ID=1 points to a specific entity.

```
GET <service-root>/Subscribers(ID=1)
```

To invoke an action through an import, a consumer sends a POST request to an action URL. Since actions, can change the state of data within the producer, the consumer can embed parameter values within the request. For example, a consumer tries to add SIP URIs for two RCS clients belonging to a subscriber; it creates a JSON document which is embedded with the POST request, as presented below:

```
POST <service-root>/Subscribers(ID=2765434434)/AddSIPUri
{
  "devices": [
    { "client": 2, "SIP_URI": "sip:ronnie@domain.com" },
    { "client": 3, "SIP_URI": "sip:ronnieXYZ@domain.com" }
  ]
}
```

## 4.6 Extending Producer Capabilities

OData can be improved through the release of a new version or customised by the producer to meet service requirements. That is, the protocol can be extended to incorporate new specifications or emerging use-cases. This can be achieved through the use of appropriate conventions, version definitions and explicit extension points. Different aspects of a producer that can be extended include query, action, function, header field, payload, vocabulary and data format.

**Query** OData URL requests are limited by the query options available to the consumer. These determine how the request itself is processed by the OData producer. The prefixes “\$” and “@” are reserved by the OData producer. So, an OData service can distinguish between a query option that is part of the specification [74] by the prefix attached to the option. Customised query option

can be implemented on the OData service if it does not use the reserved system prefixes.

**Action and Function** The range of operations that can be performed on a service or resource are extended by Actions and Functions. As discussed in Section 4.2.1, actions can be used to manipulate stored data while functions simply are used for retrieval purposes. Thus, both can be extended as required.

**Header Field** OData provides facilities that support the creation of custom header fields. It defines a set of well-formed rules that can be applied against specific HTTP requests and response headers. However, custom headers must not begin with the “OData” reserved word.

**Payload** OData provides support for payload extensions to a precise format. The OData-Version header is used to determine how a payload is interpreted regardless of the Content-Type. Hence, consumers and producers must provide effective mechanisms to handle or safely ignore contents that are not defined in the payload version specified by the OData-Version header.

**Vocabulary** Shared vocabularies provide greater extensibility to OData services. Since a vocabulary comprises a set of annotations, the vocabulary can be extended by using the annotations to improve the capabilities or increase the features of a schema element.

**Format** OData producers can support both Atom and JSON data formats. Although OData version 4 primarily supports JSON and Atom [69], other formats can be introduced to meet specific use-cases. This can be in the form of extensions to within both HTTP request and response payloads.

## 4.7 Summary

This chapter has presented a brief overview of the OData protocol. The protocol adopts a SOA model of communication over a RESTful interface. OData provides a native support for two data exchange formats, which are JSON and Atom, but can be extended to support others. It allows a data service to expose its content through an Entity Data Model, which can be accessed through the service and metadata description documents. Thus, it allows the unique repository to provide a logical view of heterogeneous data. In its attempt to provide OData as a potential candidate for the Ud reference point, this chapter has highlighted OData’s data manipulation and extension capabilities. OData supports both CRUD and subscribe/notify operations over HTTP, which are the main data manipulation requirements for the Ud reference point. The subscribe/notify feature can be realised through the callback mechanisms discussed in this chapter. However, OData provides a data-store-agnostic technique for managing data, and thus, does not define the actual model used by an underlying data store used by the logically unique repository. Therefore, the aim of the next chapter is to investigate different design practices that may be adopted by the unique repository.

# Chapter 5

## Data Store Design Practices

More often than not, data stores are found at the core of an organisation's infrastructure. They assist in the proper management of data which are created and utilised daily. Choosing the most suitable among the extensive range of data stores for managing an organisation's data can be a challenging task. This is because different trade-offs have to be made as most storage technologies are created for specific use-cases. A typical data store technology organises data in a certain way, exposes these data through specified access mechanisms and enforces some constraints on how its data can be manipulated. Thus, creating a systematic way for executing CRUD operations and in essence, manage data. Moreover, the effective management of data can provide insights, which can be used in making intelligent business decisions. Hence, it is important to understand the concepts used by data storage systems to decide how these technologies can be leveraged.

However, since the converged repository provides a logical model of consolidated subscriber data, this chapter explores strategies and data stores for managing subscriber data. The discussion builds on Chapters 3 and 4, which highlighted the design requirements for introducing a converged repository for RCS services and presented a candidate protocol for accessing this repository respectively. Thus, the chapter starts out by providing an overview of common data modelling techniques adopted by storage technologies. The next section discusses the concept of data abstraction as a way to provide access restrictions to stored data. This is followed by a brief explanation of schema management. The chapter further discusses the concept of data store transactions, fault tolerance techniques and their trade-offs. Thereafter, the chapter discusses the different approaches that can be used for storing data. The chapter concludes with a summary of concepts discussed.

## 5.1 Common Data Models

A DM as explained in Section 3.3, is derived from a specific IM. Distinct data store technologies adopt different approaches to how stored data is organised and viewed. The author regards the following as the common DM categorisations: Hierarchical, Relational, Key/Value, Document, Column-Oriented, and Graph models. They can be used to model information which is specific to required use cases. In the case of this thesis, it seemed appropriate to investigate these models in order to determine which would be the most appropriate one to handle the different categories of RCS subscriber data. This section briefly summarises this investigation.

### 5.1.1 Hierarchical

A hierarchical data model organises stored data in an inverted tree-like structure, with the root being the topmost part of the model. It adopts a parent-child relationship between its data sets which are represented as segments [33, Ch. 2, p. 35]. In other words, a parent can have multiple children while a child can only have one parent. Each segment (with the exception of the root) has a parent which determines its position within the hierarchy. Examples of hierarchical data store implementations include 389 Directory Server<sup>1</sup> and OpenLDAP.<sup>2</sup>

Furthermore, this model is effective for one-to-one and one-to-many data relationships [33, Ch. 2, p. 35]. In situations where many-to-many relationships occur, this model will be inadequate. For example, a network subscriber uses a set of MNO services which are also used by many subscribers. Thus, a one-to-many relationship from the subscriber perspective and a one-to-many relationship from MNO services perspective — leading to a many-to-many relationship. Consequently, a hierarchical data store requires more resources to create these relations and may introduce complexities in its management.

Figure 5.1 shows a basic relationship between the model segments.  $A$  is the root segment while  $A_1$  is a parent segment to  $A_{11}$  and  $A_{12}$ . With the aid of the diagram, if a linear search was conducted on the data store for  $A_{22}$ , it has to start from  $A$  to  $A_2$  then  $A_{22}$  which will be quite slower than a search for  $A_2$ .

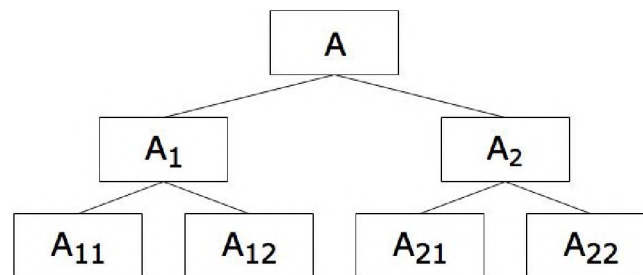


Figure 5.1: Simple Hierarchical DM.

---

<sup>1</sup><http://directory.fedoraproject.org/>

<sup>2</sup><http://www.openldap.org/>



### 5.1.2 Key/Value

This DM provides a simple organisation of data by binding keys to values. Hence, the data store can fetch specific data sets using their respective keys. This approach to data modelling is similar to techniques such as Maps and Dictionaries, which are data structures used in programming languages [102]. Key/value data stores allow for different data structures such as collections of data. Basic units of data can range from structured data such as integers and strings to unstructured data such as images and videos. Thus, a key/value data store may be used to store both structured and unstructured types of data. Figure 5.2 depicts a simple key/value DM. The simplistic nature of this DM restricts the set of possible operations since a data store provides access to data through specified keys. Simply put, complex logical relationships among data may not be easily achieved as data sets are isolated and accessed only through their keys. Examples of key/value data stores include [4]: BerkelyDB, Voldermort, Aerospike, LevelDB, and Redis.

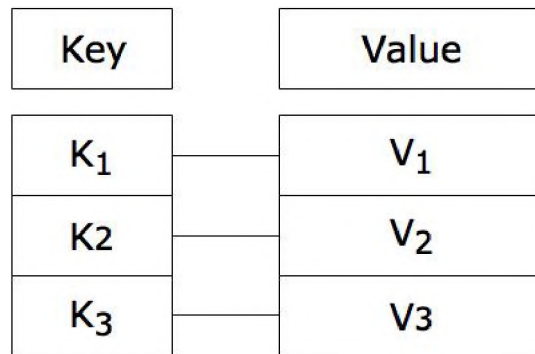


Figure 5.2: Simple key/value DM.

### 5.1.3 Relational

This model has a background in mathematical set theory and relational algebra [29]. It is based on the following concepts [17, Ch. 8, p. 404]: Set, Tuple, Arity and Relations.

- A Set consist of a number of elements at a given time. For example  $A = \{a, b, c, d\}$ . Set A comprises elements a,b,c, and d.
- A tuple can be defined a set of values for a defined data object. For example, a vehicle object has attributes such as the make, type, and year of production. Therefore a tuple  $X = \{Tesla, car, 2016\}$ .
- The arity of a tuple is the number of the element it comprises. Therefore, tuple X above has an arity of three (3).
- A relation is a group of tuples with the same arity.

Data stores that adopt this model represent data in rows and columns within a table [97]. The columns define the attributes of a data object while the rows, which can also be referred to as tuples, comprise the actual values given for each column. Thus, a table presents a template for the structural arrangement of the values specified for a data object. Examples of relational data stores include MySQL,<sup>3</sup> and Postgresql.<sup>4</sup>

Figure 5.3 shows a basic relational modelling approach where different tables are logically connected. This allows them to share common data sets using multiple keys — the primary and foreign keys. Data duplication can be eliminated since a data store can extract related datasets using the primary and foreign key notations. From the diagram, Table *A* contains attributes from Tables *B* and *C* while Table *D* contains attributes for Tables *A* and *C*.

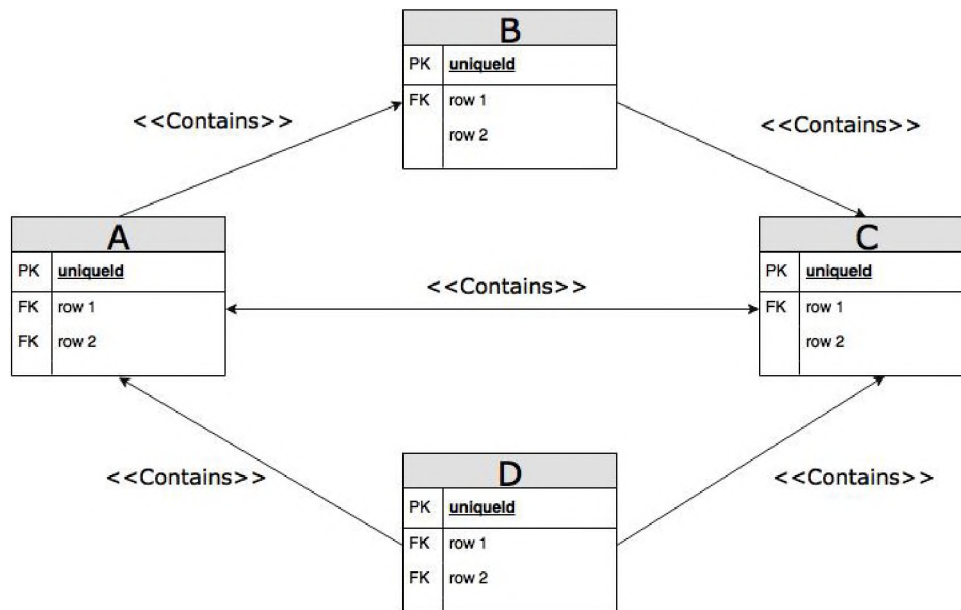


Figure 5.3: Simple Relational DM.

#### 5.1.4 Document

The document DM is a derivative of the key/value and hierarchical data models. This DM represents a unit of data as a document, which can either be structured or unstructured. A key is used to identify a specific document, which can also consist of multiple nested documents [26]. The nested documents are organised in a hierarchical manner and consequently provides the ability to embed more documents within a parent document. Within a document model, a primary key is used to access a parent document while multiple secondary keys are used to access embedded documents. This is depicted in Figure 5.4. Further, the documents are similar to tuples in a relational model as they are organised into equivalent representations of tables known as collections. Thus, a document can be added or

<sup>3</sup><https://www.mysql.com/>

<sup>4</sup><https://www.postgresql.org/>

detached easily within a collection. Examples of document data stores include [4]: MongoDB, CouchDB, SimpleDB and RavenDB.

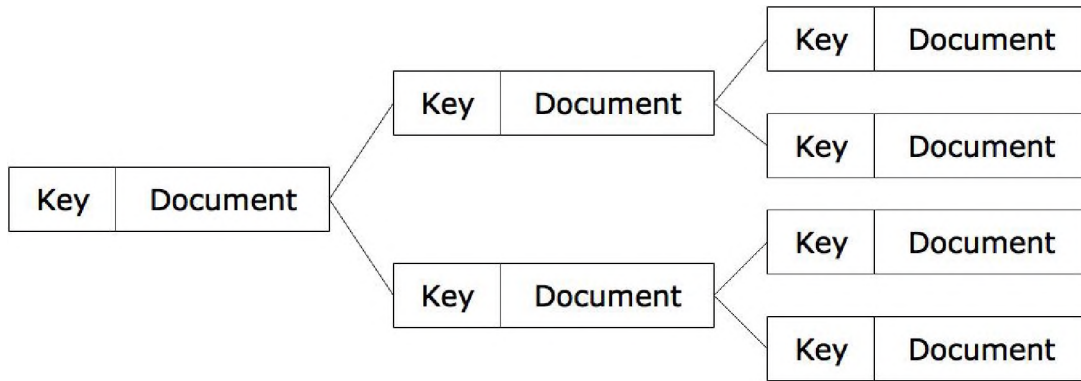


Figure 5.4: Simple Document DM.

### 5.1.5 Column-Oriented

A column-oriented model adopts a relational DM by also representing data in rows and columns [56]. The difference between both models is the flexibility of the tuples. The relational model can be termed a row-oriented model as tuples are generated against fixed set of columns within the data store. Conversely, column-oriented DM ensures that columns are only generated as required by the tuple. Therefore, a relational model uses a fixed arity value while a column-oriented model possesses variable arity value. Additionally, the columns in this DM contains a set of key/value paired data as depicted in Figure 5.5. Examples of column data stores include: HBase<sup>5</sup> and Apache Cassandra.<sup>6</sup>

Rows	Columns			
Row <sub>1</sub>	Column Name <sub>1</sub>	Column Name <sub>2</sub>		
	Value	Value		
Row <sub>2</sub>	Column Name <sub>1</sub>	Column Name <sub>3</sub>	Column Name <sub>2</sub>	
	Value	Value	Value	
Row <sub>3</sub>	Column Name <sub>2</sub>	Column Name <sub>4</sub>	Column Name <sub>3</sub>	Column Name <sub>1</sub>
	Value	Value	Value	Value
Row <sub>4</sub>	Column Name <sub>1</sub>			
	Value			

Figure 5.5: Simple Column-Oriented DM.

<sup>5</sup><https://hbase.apache.org/>

<sup>6</sup><http://cassandra.apache.org/>

### 5.1.6 Graph

In this DM, the basic unit of data is called a node and the relationships between nodes are represented with arrows known as arcs (otherwise known as edges) [17, Ch. 9, p. 451]. The core of this model is in established relationships between nodes. Relationships comprise a type, start node, end node and property. Properties give more description to the relationship. Graph models can be directed or undirected. An example of directed graph model is a hierarchical model, which can be traversed in a certain direction from a start point (the root segment) to the end point (specified destination). Undirected graph models may not provide a specific way in which the model can be traversed. Examples of graph data stores include: Neo4j<sup>7</sup> and FlockDB.<sup>8</sup> Figure 5.6 shows a simple directed graph DM. It shows that the node for User A has a *known* relationship with User B. Then the Users A and B are subscribed to Service<sub>1</sub>. Further, the diagram suggests that Service<sub>1</sub> uses a specific data store within a network that manages two data categories — Data<sub>1</sub> and Data<sub>2</sub>. This simple illustration explains how the graph DM can help find manage and define relationships between stored data.

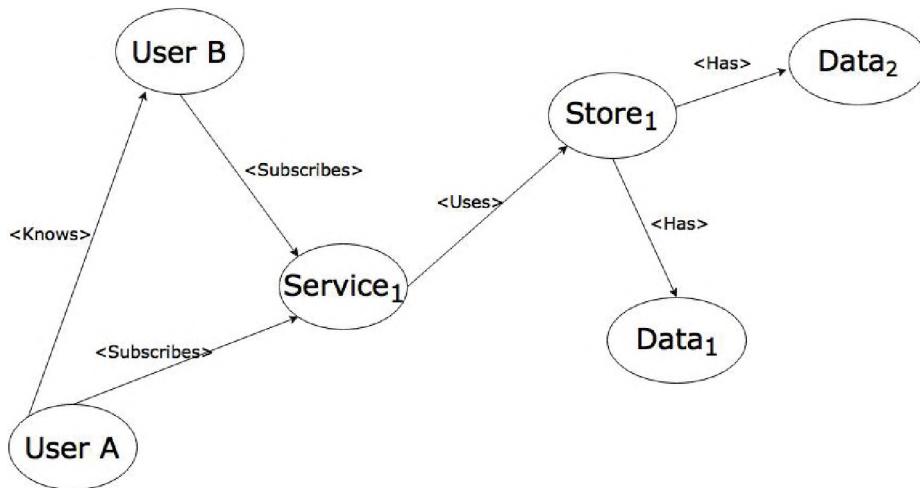


Figure 5.6: Simple Graph DM.

## 5.2 Data Abstraction

The term data abstraction is synonymous with relationships between primitive data types and objects in object-oriented programming languages such as Java. That is, an object can comprise different primitive types, which may not be exposed to other objects. In data management scenario, the type of data that can be retrieved is dependent on the level of abstraction provided by the data objects. Hence, when designing a data store, different data structures have varying representations at different levels — each hiding some information present at the lower level. These data structures and their relationships constitute data models at each level. There

<sup>7</sup><https://neo4j.com/>

<sup>8</sup><https://github.com/twitter/flockdb>

are four categories of such models: External, Conceptual, Internal and Physical [33, Ch. 2, p. 47].

- **External Model** Constitutes the different DM views presented to the clients applications that interact with a data store. This provides the highest level of abstraction within a data store. In an SOA environment, this is the model that is exposed to data service consumers.<sup>9</sup>
- **Conceptual Model** Provides a flexible data structure that represents a logical organisation of the contents of a data store or an SOA's data service.<sup>10</sup> This DM provides a higher level of abstraction over the internal model (described below). The conceptual model is logically independent of the internal model implementation as changes made with the internal model does not impact the conceptual view of data.
- **Internal Model** Describes how data is organised within a data storage technology. This view of data describes how the data store organises its data. An example is the LDAP directory service which internally organises its data in a hierarchical manner. Changes made on this model does not impact the physical model. Hence, it is independent of the physical model as shown in Figure 5.7.
- **Physical Model** Shows the actual organisation of data at the file system level of the host Operating System (OS), which uses storage media such as hard disks. It provides the lowest level of abstraction.

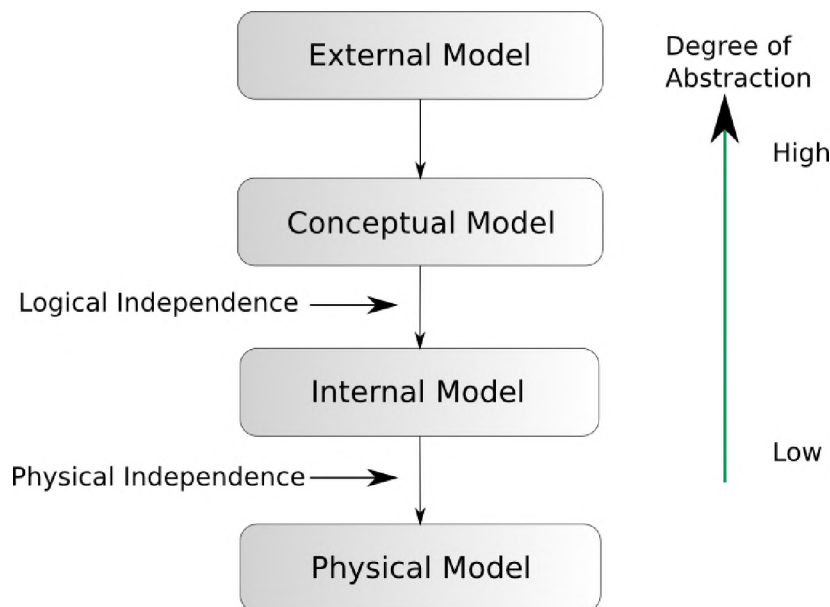


Figure 5.7: Data abstraction levels. Adapted from: [33].

<sup>9</sup>An example of the external model is the UDC's ADM discussed in Section 3.4.

<sup>10</sup>The conceptual model is synonymous to the CDM discussed in Section 3.3.

## 5.3 Schema Management

Schemas are derived from the data abstraction models discussed in Section 5.2. A schema presents a logical representation of stored data [96]. Data store technologies adopt two main principles for managing their schemas. They employ either schema-full or schema-less models [67], which are discussed in the following subsections.

### 5.3.1 Schema-full Model

A schema-full model enforces constraints on how data is stored, accessed and manipulated. In other words, there is limited flexibility allowed in data management when a strict policy is adopted. Relational data stores often adopt this policy as a way to ensure consistency on how data is managed. Data models on such systems may be re-designed to accommodate new behaviour. For instance, introducing additional relationships between tables in a relational data store can lead to unanticipated complexities. This may consequently result in the re-design of the initial DM. Examples of such storage systems include: MySQL and Mnesia.<sup>11</sup>

### 5.3.2 Schema-less Model

A schema-less model has native policies but allows a data service to access and manipulate data with fewer structural constraints and greater flexibility when compared to systems with strict schema policies [96]. Data services that use such storage systems determine how data is stored and accessed. Thus a mixed schema data store allows a data service to mount a conceptual model over its internal model. Consequently, the data service is burdened with the responsibility of determining the way data is accessed and utilised. In addition, this allows data services to create and manage their own DM without being impacted by the modelling constraints (if any) of an underlying data store. Examples of such storage systems include MongoDB, Cassandra, and HBase. These systems are regarded as non-relational data stores as they do not enforce a rigid relational schema [102].

## 5.4 Transaction Management

Section 3.4.5 discussed the ACID properties that guarantee successful transactions. More often than not, relational data stores are transaction-oriented as they try to ensure consistency and enforce data integrity when handling data manipulation requests. However, there is a paradigm adopted by non-relational data stores known as BASE [86]. BASE is an acronym for a system that is basically available, soft

---

<sup>11</sup><http://erlang.org/doc/man/mnesia.html>



state and eventually consistent. This concept follows a distributed data storage architecture, which is discussed in Section 5.5. The terms are explained as follows:

- **Basically Available:** This simply describes how data is perceived among server nodes after it is updated. The probability that data can be accessed and served within acceptable response times can be expressed as a ratio.
- **Soft state and Eventually consistent:** This implies that nodes may contain different versions of the same data in a short period. There is a partial consideration for data consistency as changes made on data are propagated across server nodes. A BASE-compliant data store is always in a soft state when compared to an ACID-compliant system, which attempts to guarantee consistency at a given period [86]. Thus, the state of a data store may continuously change at a given time, even if there are no data manipulation activities being handled. Consequently, this might result in delays over the network, making it almost impossible for two server nodes to be in the same state at a given period. However, the data store is expected to produce consistent data when it is queried.

## 5.5 Fault Tolerance Techniques

Systems that adopt ACID or BASE concepts, utilise distributed architectures to handle data store performance and availability, with respect to, increasing data volumes [104]. Hence, a storage technology can also be described by how it manages its data across multiple server nodes. The techniques used to manage these distributed data sets are known as Sharding and Replication respectively [104]. It is important to note that some data stores can leverage both concepts when managing data.

Sharding allows a data storage technology to store unique copies across different server nodes [19]. Each node is referred to as a Shard. A Shard is created based on the use-cases for the data store — the functional requirements. Thus, a shard can be created to handle data subsets for different domains. For example, having a shard to handle subscription profiles, and another to keep CDRs. Hence, each server node contains a unique set of data, which is consistent for the data services that access it. As a result, a data service can interact directly with a specific node for pertinent data. This ensures that similar data sets are grouped and stored on the same node. Hence, a data store can effectively manage transactions on data sets across each server node as data continues to grow. However, if a node fails, the data store may lose data if there are no backup mechanisms in place or if its current state is not yet backed-up.

Replication, however, allows a data store to produce multiple copies of its data sets across server nodes [16]. Thus, if a node goes down, the data can be retrieved from any other available node. The strength of this approach depends on the concurrency mechanisms created for the data store. This is because changes made to the state of

the data store have to be replicated across all server nodes. The relationship between nodes can either be peer-to-peer or master-slave. In the master-slave relationship, the master is the authoritative node that propagates changes to slave nodes. In the peer-to-peer relationship, writes are allowed on each node and a node synchronises its copy with other replicas. Therefore, multiple server nodes are expected to contain the same set of data at a given time. However, isolating a unit of data at a given period becomes unrealistic for replicated nodes as they are all in synchronisation.

## 5.6 The CAP Theorem

The CAP theorem by Eric Brewer [23] considers three related system properties namely: Consistency, Availability and Partition Tolerance (CAP). It asserts that any distributed data store can only possess two of the three expected properties at a given time.

- **Consistency:** This is not synonymous with the ACID form of consistency. This is used to describe the state of distributed storage systems. Hence, it implies that all server nodes must contain the same data sets at all times.
- **Availability:** It has a similar meaning to the one described as a BASE property.
- **Partition Tolerance:** In the event where communication among servers is unreliable (for instance, the network stops, delays or drops messages sent between the servers), the system is expected to continue with its tasks. This is the inherent nature of the Internet and its related services. Hence, the system has to tolerate this kind of situation.

However, according to Brewer, it is only possible to have a data store that can reliably provide two of the three properties. Thus, trade-offs are made between these properties when deciding on the appropriate data store to adopt. More often than not, consistency and availability are favoured since partition tolerance is essentially a fault-tolerance mechanism, whereas consistency and availability ensure the reliability of stored data. This is because data services expect these technologies to be reliable and capable of handling their specific use cases [16].

## 5.7 Persistence Model

Persistence simply means that “data survives after the process with which it was created has ended” [83]. This section discusses the different methods adopted by data services, which provide a conceptual model of data when managing the persistence of heterogeneous data sets. They are thus classified into single and multi-store persistence models respectively.



### 5.7.1 Single-Store Persistence

A single storage approach to persistence guarantees a uniform DM. A distributed store can shard or replicate data across multiple server nodes, but they all adopt a single DM. Thus, simplifying the data managing process. This is illustrated in Figure 5.8. However, when considering data heterogeneity, the data stores can be made to fit in disparate data sets into a uniform DM. As discussed thus far, different data storage technologies are built for specific use cases and different DMs. For instance, relational and non-relational data stores are designed to solve specific problems like scaling, transaction and schema management, among others. Hence, trying to fit heterogeneous data into a single data storage technology may eventually lead to a non-performant system [95]. Consequently, an alternative to handle data heterogeneity is persisting data across multiple stores.

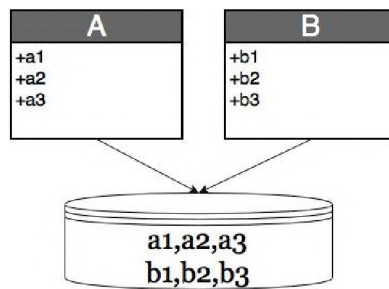


Figure 5.8: Single store persistence example.

### 5.7.2 Multi-Store Persistence

As illustrated in Section 5.1, there are different types of data storage technologies suitable for specific models of data. Persisting data across multiple stores allows a combination of heterogeneous data sets within a single data service. The term used to describe this is Polyglot Persistence [95]. It follows the paradigm that data stores are designed to solve different problems. The idea stems from the concept of polyglot programming where multiple programming languages can be integrated into a data service to tackle specific problems [104]. This approach allows an application to leverage the strengths of different languages. An example is the Scala programming language, which operates on the Java Virtual Machine (JVM). Hence, both Scala and Java can interwork on a single platform.

Polyglot persistence provides an architecture that allows data services to store data from heterogeneous sources based on data access requirements. For instance, persisting session-related data in a relational store when a simple key/value store would be sufficient. Or managing relational data with a document-oriented store. Polyglot persistence provides an architectural answer to such scenarios. Thus, data sets with similar use cases can be persisted in distinct data stores. Additionally, the concepts discussed in Section 5.4 can be leveraged when choosing appropriate data stores for a data service.

With polyglot persistence, a data service can persist a data object and its attributes,

across multiple data stores. For example, instead of keeping a subscriber's profile information within a single data store; they can be separated into multiple data stores to manage different use cases such as frequent reads and occasional writes. This is illustrated in Figure 5.9, where a data service that manages data objects A and B, persists their attributes across multiple stores.

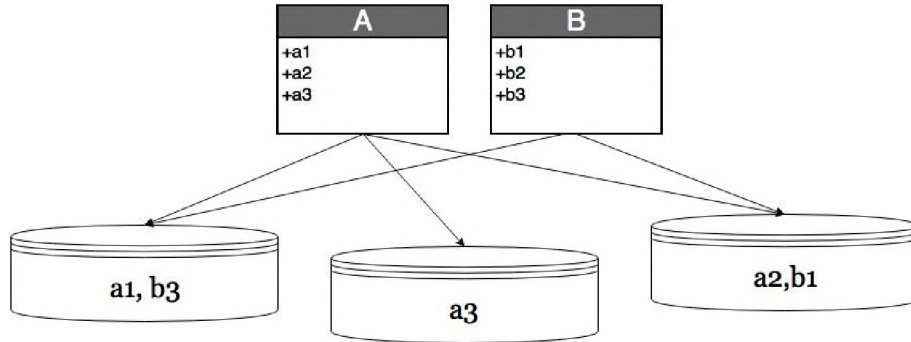


Figure 5.9: Polyglot persistence example.

## 5.8 Summary

This chapter has described common approaches which can be used to design a data management system. The different ways in which data is often represented in data stores have been shown. Additionally, the different data abstraction, schema and transaction management techniques have been discussed. More importantly, the CAP theorem, which can be used to make trade-offs between different storage technologies have also been presented. The persistence models, which explain the use of single and multiple storage technologies were discussed toward the end of the chapter. With this insight, the chapter has set the stage for the design of the unified data repository for RCS, which is presented in the next chapter.

## Chapter 6

# Introducing the Converged Subscriber Data Repository

Up until this point, this thesis has discussed at length the requirements for creating a converged data repository for RCS services. This involves both the data access and modelling requirements. Since this repository needs to conform to the UDC standard, the thesis has also explored alternative mechanisms to address the issues raised with 3GPP specifications. These mechanisms were explained in Chapters 4 and 5. However, this chapter presents an architectural design for the unique repository, which the author has coined the Converged Subscriber Data Repository (CSDR). Thus, the chapter starts out by presenting an analysis of the design decisions that were made for the CSDR. Then follows, an overview of the CSDR and FE components while outlining their various functions. The section further defines the core components of the converged repository. This is followed by a description of how the CSDR handles CRUD and subscribe/notify messages. The chapter concludes with a summary of the ideas discussed.

## 6.1 Design Considerations

This section discusses the factors, which informed the design of the CSDR.

### 6.1.1 Data Protection

Protecting the contents of the CSDR from services that are not primarily supported by RCS is important. Such services include unauthorised FEs and third party applications. Although OASIS has not defined a standardised security mechanism for protecting the contents of the OData EDM, TS 22.985 [9] discussed the impact of security for a converged repository. The specification encourages the use of, but not limited to, the Advanced Encryption Standard (AES) to protect sensitive data such as authentication keys or passwords. Adopting the UDC standard, the CSDR preserves existing authentication and authorisation procedures within an IMS network [100]. Thus, these data sets can be encrypted over OData, since AES is protocol-independent. Moreover, since OData is a REST protocol, HTTP security mechanisms discussed in RFC 2616 [21] and RFC 5789 [40] can be adopted. Hence, advanced security techniques can be built over HTTP, which is used by OData.

### 6.1.2 Formalised Access Mechanism

As discussed in Section 6.3, the CSDR supports both CRUD and subscribe/notify operations over OData. UDC specifications suggested the use of LDAP and SOAP to achieve the same objective. Hence, providing access to heterogeneous data sets with a single unifying protocol simplifies the processing activities of the CSDR. Consequently, the RCS core and network service enablers can interact with the CSDR using OData as the common interface. However, there are other RESTful alternatives to OData, such as the OPEN-PaaS-Database API (ODBAPI) [98] and Backend-as-a-Service REST API [46]. OData is different from both APIs on two levels. First, OData is currently standardised and maintained by OASIS while the other two APIs are not currently standardised. The second difference is evident in the advanced querying and aggregation techniques provided by OData<sup>1</sup> which are not present in the other APIs. Thus, the CSDR leverages OData to expose data in a consistent, flexible and reliable manner to the relevant FEs.

Furthermore, OData provides a much more flexible way of querying its data as discussed in Section 4.5.3. OData leverages the flexibility of the EDM to navigate entity relationships and query entities with their attributes. This is also achieved without the knowledge of the underlying data store implementation and actual models used by the store. Thus, the CSDR offers querying capabilities such as data aggregation to FEs, through the OData query options. This offers an improvement over the use of LDAP, which navigates the contents of a directory service through the filtering entries and attributes in the Directory Information Tree (DIT) [106].

---

<sup>1</sup>These techniques are discussed in sections 4.5.3 and 4.5.4 respectively.

### 6.1.3 Interoperability with External Systems

The CSDR should be able to exchange its data with external systems within the network. Such systems may include data mining systems or others, which may query the CSDR for pertinent data. Considering the heterogeneity of the protocols used across a mobile network, external services may have to interact with the CSDR using LDAP, SOAP or both. However, using OData allows these external systems to interact with the CSDR using a simple REST interface, which is gradually being adopted by MNOs. If the external systems expose their data through OData, the CSDR can also interrogate their content when trying to aggregate data. In addition, these systems can study the contents of the CSDR through the metadata document. Existing systems that currently incorporate OData, which will be able to access the CSDR include [76]: Microsoft Sharepoint, SAP NetWeaver Gateway, IBM Websphere eXtreme Scale, and JBoss Data Virtualization, among others. Thus, subscriber data within the CSDR is made available for access by operator-owned network services, and authorised third-party systems.

### 6.1.4 Coping with Schema Evolution

One of the functions of a schema is to enforce certain rules regarding stored entities in order to ensure data integrity. This determines how the CSDR reads and writes subscriber data. However, this can lead to complex data structures as data access requirements to the repository continue to change [94]. A possible way to deal with this complication is to re-engineer the DM of the CSDR at each point. Alternatively, adopting OData allows the schema defined by the CSDR to accommodate changes in models as RCS services evolve. In other words, the EDM can allow MNOs to create wrappers around the underlying data store(s) and in consequence, the CSDR can evolve as required.

### 6.1.5 Persistence and Transactions

The data sets utilised and generated by RCS services are of different structures represented as distinct DMs. Simply put, the data sets are heterogeneous in nature. The CSDR manages this heterogeneity by adopting a polyglot persistence architecture. Hence, the CSDR manages distinct models by utilising suitable storage technologies. However, combining different underlying DMs requires the CSDR to provide complex data processing techniques for the distinct data sources, since it provides an abstraction over the stores. OData reduces this complexity by providing a consistent way to manage and expose data at the conceptual level. Therefore, schema-full and schema-free data stores can be integrated with the CSDR in a “plug and play” fashion.

Furthermore, not all types of data require transactional integrity. For example, data generated within a session is largely transient in nature. Hence, adopting

polyglot persistence allows transaction and non-transaction-oriented stores to manage pertinent data. Consequently, ACID-BASE the dichotomy is resolved as the CSDR leaves transaction management to each underlying data store. It forwards the outcome of these transactions to the FEs as OData result codes, thus, eliminating the need to enforce transactional properties on CRUD operations.

## 6.2 System Overview

The design for the CSDR architecture is inspired by both UDC and OData architectures described in Sections 3.1 and 4.1 respectively. The CSDR is both modular and multi-layered by design. The modular aspect of the system ensures that each task is handled by a designated unit — resulting in clearly defined roles. The layered architecture provides levels of abstraction for data access and manipulation. These layers are created for the following access requirements: multi-interface, unified, and low-level data access. Figure 6.1 illustrates this layered design and its core components.

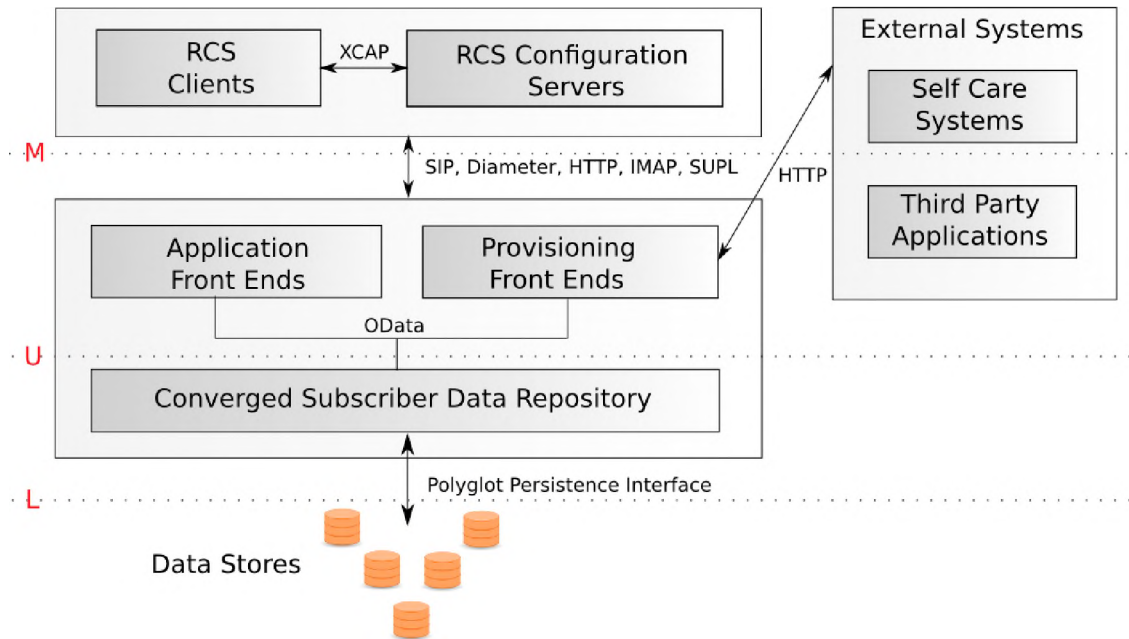


Figure 6.1: A high level overview of the CSDR architecture.

The *multi-interface* layer — labelled ‘M’ — acts as the topmost data access layer where different communication protocols and APIs are used between the clients and network services. The network services that respond to requests at this layer are the application and provisioning FEs. The clients that can operate at this level range from smartphones, third-party applications, and operational support network services, to applications that allow subscribers to manage portions of their data through an external user interface (that is, self-care systems). Hence, different application layer protocols can be implemented and adopted between clients and services that operate at this level. Consequently, most services within IMS can query the CSDR for subscriber data at this level.

The *unified* layer — labelled ‘U’ — is the mid-level data access layer. It comprises the logically unique data repository, which supports a single protocol. The protocol adopted is the OData Protocol. This access interface is synonymous with the Ud reference point specified in the UDC architecture. By using OData, the CSDR can provide a conceptual view of heterogeneous subscriber data to FEs. OData enables FE-to-FE and FE-to-CSDR interactions. Thus, it is the core component for converged data access within this architecture.

The *low-level* layer — labelled ‘L’ — is the data access layer consisting of the underlying data stores within the architecture. At this level, the CSDR can store subscriber data into actual data stores. This layer ensures that there is loose-coupling between the CSDR and actual data stores through the introduction of the Polyglot Persistence Interface (PPI). The role of the PPI is further discussed in Section 6.2.3.4. Therefore, underlying data stores can be migrated, swapped, upgraded and removed as required.

### 6.2.1 Application FE

An AFE, as previously discussed in Section 3.1.1 is a stateless service which responds to client requests. It is stateless because it does not possess any subscriber data. Clients perceive an AFE as either an application or data service. AFE-type interactions within this architecture are twofold: firstly, interactions occur between FEs and their clients; secondly, they also occur between FEs and the CSDR.

#### AFE Interactions

An AFE receives requests from clients and serves back response messages. It achieves this client-server interaction through its defined interface(s) as depicted in Figure 6.1. In other words, each AFE can either support a single or multiple data access interfaces. Thus, if the AFE supports multiple interfaces, client requests are handled by designated interface modules. Chapter 8 gives different illustrations for AFE interfaces. However, the generic interaction between an AFE and the CSDR, which is triggered by a client is described as follows:

- The client issues a request toward the AFE. This request can be a message composed using any of the following protocols — SIP, SUPL, Diameter, IMAP, and HTTP. The AFE handles messages for protocols that it has been provisioned to support.
- The AFE authorises the client initiating the request.
- If the client requires subscriber data, the AFE contacts the CSDR to fetch the relevant data. In other words, the AFE interacts with the CSDR when there is a need to execute a CRUD-type request.

- Otherwise, it constructs and sends a response to the client through the same interface.

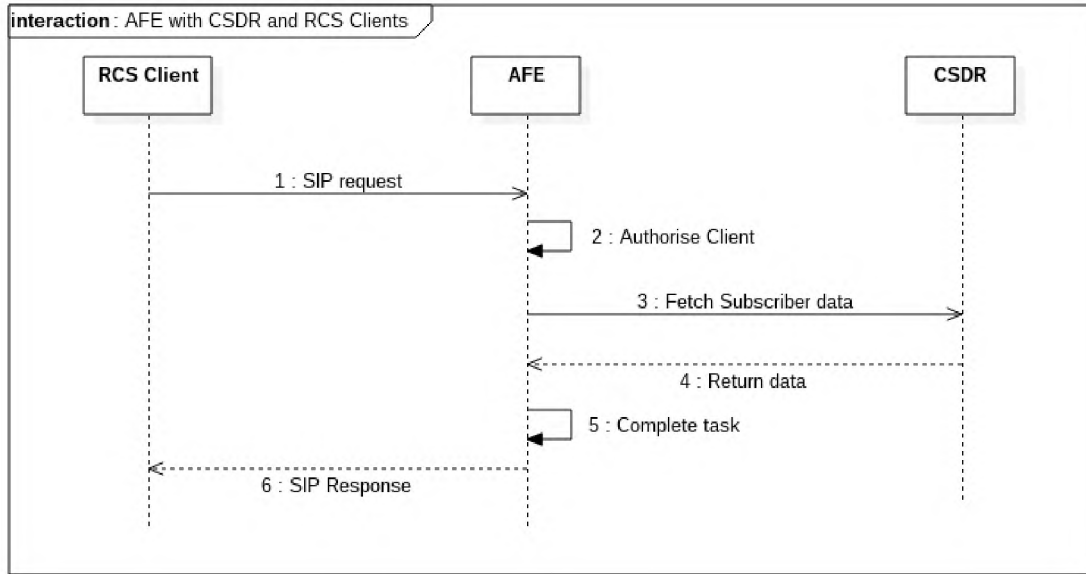


Figure 6.2: Sequence diagram depicting a generic interaction between an RCS client, an AFE and the CSDR.

Figure 6.2 illustrates a basic interaction between an RCS Client, an AFE and the CSDR. The task performed could be a subscriber registration or de-registration, among others. Such operations may require interaction with subscriber data before they can be completed. This is when the AFE decides to interact with the CSDR to fetch the relevant subscriber data to complete the given task. However, it is important to note that an AFE can generate multiple CRUD requests in order to complete a specific task.

## 6.2.2 Provisioning FE

As discussed in Section 3.1.1, a PFE is a specific type of AFE. It is designed specifically for provisioning tasks such as management of subscriber profiles. Thus, a PFE receives CRUD requests from external systems such as the RCS configuration servers, self-provisioning interfaces for subscribers, and data reporting tools, among others. A PFE supports the use of HTTP or other application layer protocols that use HTTP, such as XCAP. Furthermore, it provides the ability to create all the data sets required by the AFEs interacting with the CSDR. In other words, the PFE can be used to generate both shared and AFE-specific data sets. Shared data sets imply that multiple AFEs use the data from the same pool, while specific sets are utilised by distinct AFEs. For example, a PFE can create common and distinct data sets for the RCS messaging and location services. In contrast, an AFE is primarily concerned with its own data set. Hence, a PFE has greater access to subscriber data when it is compared to an AFE.



## PFE Interactions

A typical scenario is when the PFE is used to create subscriber data that allows multiple RCS services to be provisioned. In other words, when a user is being provisioned, the PFE creates all pertinent data for RCS service activation, usage, and account provisioning. This is possible because the PFE is aware of these data sources by leveraging the consolidated view from the CSDR. The generic interaction between a PFE and the CSDR is depicted in Figure 6.3.

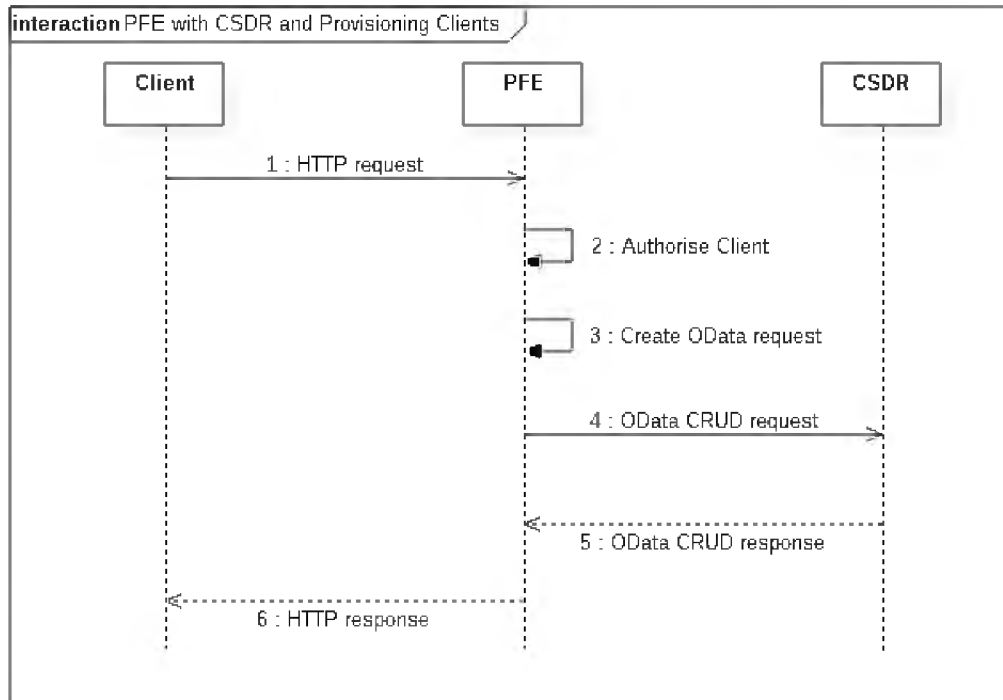


Figure 6.3: Sequence diagram depicting interactions between a Client, the PFE and the CSDR.

The interaction is described as follows:

- The client constructs a CRUD message containing the type of request. The request type ranges from the creation of a subscription profiles and user account to specific data sets which are known to the PFE.
- The client issues the request toward the PFE through a protocol. Thus, the protocol must be supported by the PFE.
- The PFE authorises the client initiating the request.
- The PFE constructs an OData equivalent of the CRUD request and forwards it to the CSDR.
- The CSDR performs the CRUD request and sends the result of the operation to the PFE.

- The PFE converts this OData response to the corresponding HTTP response and forwards it to the client.

The client could either be a SIP application service or a web browser, which the MNO has securely provisioned to interact with the PFE. Additionally, CRUD requests that the CSDR supports are discussed in Section 6.3. Moreover, the PFE sends a response to the client using HTTP response codes. In consequence, the PFE supports a single protocol in contrast to an AFE.

### 6.2.3 Converged Subscriber Data Repository

The CSDR is the facility that handles all data storage, management and querying of subscriber data sets used by RCS services and their supporting network services. It provides the overall view of RCS subscriber data to the network services, administrators and management alike. There are four main parts that handle the different behaviours of the CSDR. These components are shown in Figure 6.4. They are divided according to their respective roles within the CSDR, which are: access control, data processing, schema management and persistence.

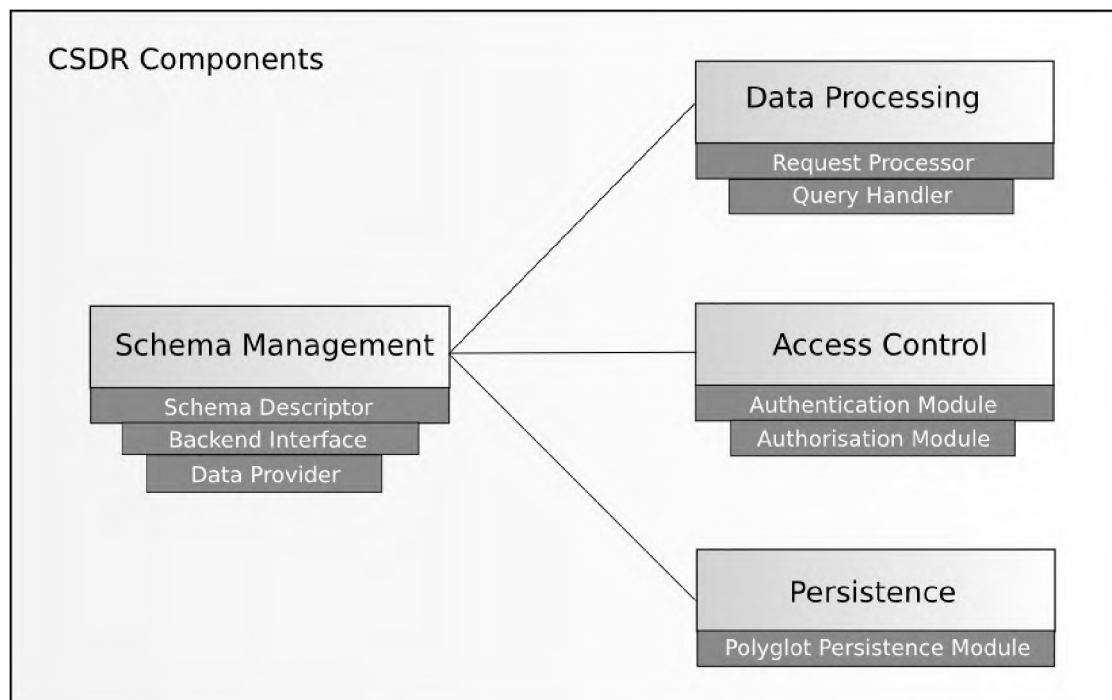


Figure 6.4: Components of the CSDR.

#### 6.2.3.1 Access Control Component

Both AFE and PFE can successfully execute CRUD operations if the CSDR has their unique identifiers. These identifiers are generated through the backend interface provided by the CSDR. The interface provides the list of registered FEs containing

their network addresses and corresponding identifiers. This backend interface is further discussed in Section 6.2.3.3. However, this data access control component performs two main functions: authentication and authorisation of FEs.

**Authentication:** This process is described as follows: First, the CSDR checks for the FE's network address and uses it to find the corresponding unique identifier. If an identifier is found, the CSDR grants it access to the FE. Otherwise, the FE is denied access.<sup>2</sup>

**Authorisation:** For the FEs to now manipulate data through CRUD operations, they must be authorised by the CSDR. This technique ensures that an FE which has not been granted privileges on a data set can not manipulate that specific group of data. This also helps the CSDR separate data access into distinct views for each registered FE. In other words, the CSDR will present each FE with a unique view of pertinent subscriber data, thus leaving out other data sets available in the CSDR. Hence, an FE has a restricted view of subscriber data. This has been factored in to the realisation of the ADV concept described in Section 3.4

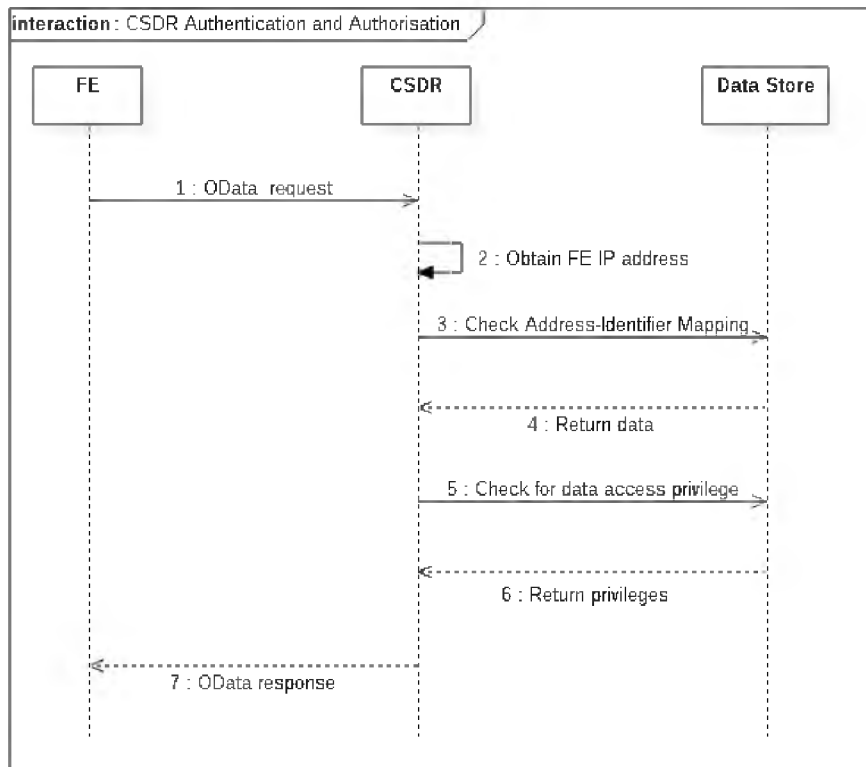


Figure 6.5: CSDR Authentication and Authorisation.

### 6.2.3.2 Data Processing Component

This component handles the execution of business logic within the CSDR. It performs two functions, namely: request and query processing. Both functions are

---

<sup>2</sup>This is illustrated in Figure 6.5.

handled by two distinct modules: The Request Processor and Query Handler.

**Request Processor:** This module handles the execution of basic OData requests. It decides which component within the CSDR to invoke in order to complete a given task. The basic function of a request processor is to handle the CRUD operations that are executed against each component of the CSDR's DM. As a result, different CSDR elements are handled by different processors. There are numerous OData request processor types and they are presented in Appendix A.4.

**Query Handler:** The CSDR provides a flexible querying mechanism through OData, which is similar to the way SQL is used with relational data stores. Queries are sent toward the CSDR as HTTP requests but with some additional commands that alters the result of the request. These commands were discussed in Section 4.5.3. For example, an OData \$select command fetches the corresponding attributes from the CSDR. The \$select command allows a developer to specify certain attributes to be returned. This is similar to the SQL select operation. Additionally, this module allows a PFE to obtain an aggregated view of subscriber data in the CSDR. Given this capability, external systems can discover new service usage patterns between subscriber data sets.

### 6.2.3.3 Schema Management Component

The actual data managed by the CSDR is persisted into underlying data store technologies. However, to provide consistent data access to different FEs, the CSDR exposes its data through the OData EDM. The EDM provides a flexible but consistent way to model data which is exposed to the FEs using its standardised interface, the OData protocol. Hence, an EDM design represents a logical realisation of the CDM, which was discussed in Section 3.3.<sup>3</sup> This component contains three sub-components: A schema descriptor, backend user interface and data provider.

**Schema Descriptor:** Describes the various components of the CSDR's EDM. It uses the service and metadata documents described in Section 4.4.2 for this purpose. The documents are the CSDR's description documents which are published to FEs. Hence, the schema descriptor showcases to the registered FEs when invoked, the different EDM elements, their attributes and relationships. Moreover, this component performs the job of a description document in an SOA environment.

**Backend Interface:** Acts as the management interface for the CSDR. It allows an administrator to manage the contents of the CSDR. This is an important CSDR module, as it allows quick administrative access to the CSDR when needed. This can be used to perform CRUD operations on subscriber data sets, FE data sets, and some CSDR configurations.

**Data Provider:** Is the module within the CSDR, which provides a structured view

---

<sup>3</sup>The steps through which a CDM can be created is shown in Figure 3.3.

of the contents of the CSDR. Upon the reception of CRUD requests from the request processors, it interacts with the persistence layer through the PPI to complete the set task. It forwards the data which it has received from the persistence component to the appropriate processor. Hence, it coordinates the data sets within the CSDR that are exposed through the EDM. Therefore, it interacts with both the persistence component and the schema descriptor. With the aid of the schema descriptor, it can choose the entity types that are queried from underlying data stores, based on the contents of the request. This is because the request processors only operate on data, they are not aware of the structure or form of the given data. Hence, the EDM is transparent to the request processors but is known to a data provider.

#### 6.2.3.4 Persistence Component

This component supports the data abstraction that the CSDR has introduced. It provides an interface between the data provider and the persistence layer. This interface is described as the PPI. The PPI helps achieve persistence by integrating the lower-level DM with the EDM. For example, when an OData CRUD request is received from the Request processor, this component converts from EDM to PPI DM. This is achieved by mapping EDM Entity Types to a corresponding model in the PPI's DM — this is how the CSDR's low-level data access is achieved. Thereafter, an appropriate data store is selected with the aid of the **Polyglot Persistence Module** (PPM). In addition, the PPM handles the transactions between the CSDR and its underlying data store. Thus, providing a configurable interface that allows a single-store or multi-store persistence approach. Once this component is configured, the CSDR can persist the subscriber data across multiple data stores.

## 6.3 CSDR Messages

### 6.3.1 Create

When an HTTP POST request to create subscriber data is received by the CSDR, the request is validated as described in the authentication and authorisation section. This is the first activity before further processing is executed. The behaviour of the CSDR when handling a create request is illustrated in Figure 6.6. The received data will be represented as an entity within the body of the HTTP message. The CSDR will extract this entity and check if its definition is valid. Then the request is forwarded to the appropriate request processor for further analysis, once it has been selected by the schema management component. This component further converts the POST request to the corresponding PPI request format. Thereafter, the CSDR will select a store to persist the subscriber data. However, in situations where the CSDR does not persist the data, it sends out the appropriate HTTP response.

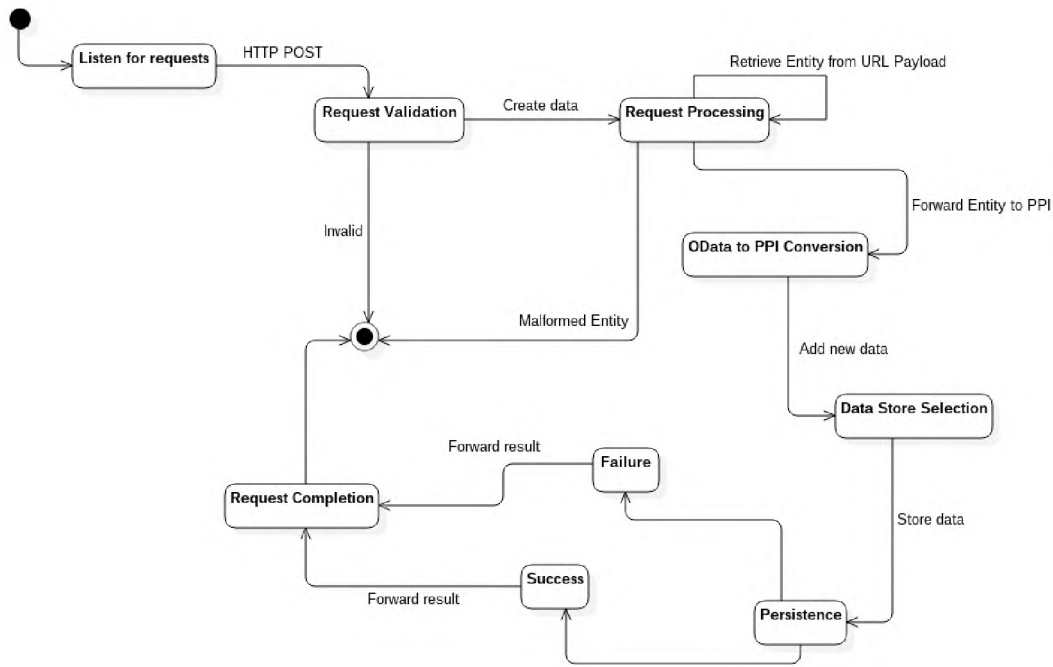


Figure 6.6: Defined CSDR behaviour when creating data.

### 6.3.2 Read

When an HTTP GET request to fetch subscriber data is received by the CSDR, the request will also be validated. The CSDR will extract the subscriber data identifier, which must be of the same data type in the EDM. Once a valid identifier has been obtained, the CSDR converts the GET request to its corresponding PPI message. Thereafter, it chooses the appropriate store to query the data based on the information provided by the PPM. Upon the retrieval of the pertinent data set, the CSDR can filter the result through the usage of query options. These options are specified with the GET request and therefore, are part of the HTTP URL. The CSDR applies these options and then generates the result of the operation. This is then sent out as a response to the initial request. The process is described in Figure 6.7.

### 6.3.3 Update

Upon the receipt of either an HTTP PATCH or PUT, the CSDR will validate the request. Thereafter, it extracts the identifier of the data to be obsoleted and the payload of the new data. Then it creates the PPI message equivalent of either HTTP PUT or PATCH. The CSDR will select the concerned data store and attempt to modify the existing data using the specified update method. When the CSDR finalises this update, it will send out an appropriate HTTP response code. This is depicted in Figure 6.8.

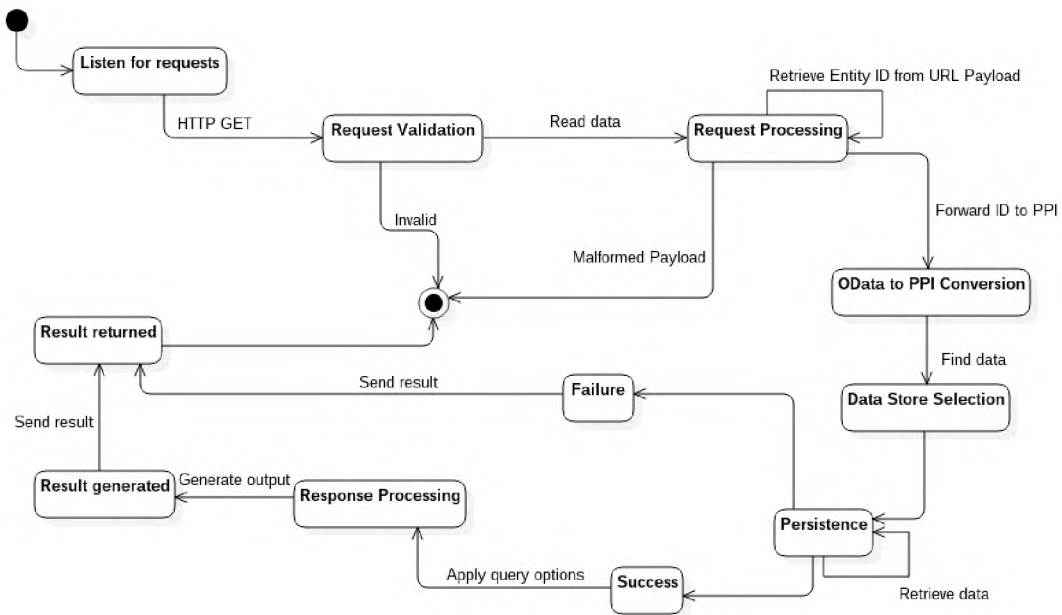


Figure 6.7: Defined CSDR behaviour when fetching data.

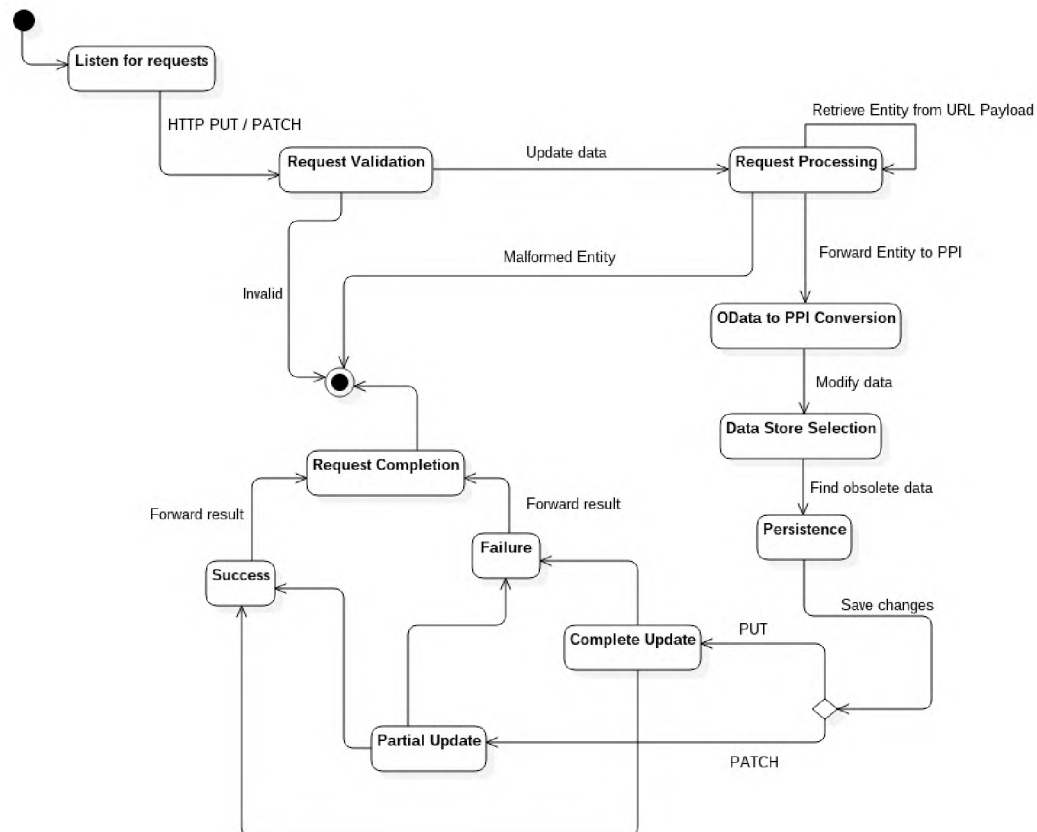


Figure 6.8: Defined CSDR behaviour when updating data.

### 6.3.4 Delete

The CSDR validates the HTTP DELETE message upon reception. It extracts the identifier for the specific subscriber data from the URL. Further, the CSDR will create a PPI message that allows it to search and delete the specified subscriber data. Thereafter, it sends out the result of this operation as illustrated in Figure 6.9.

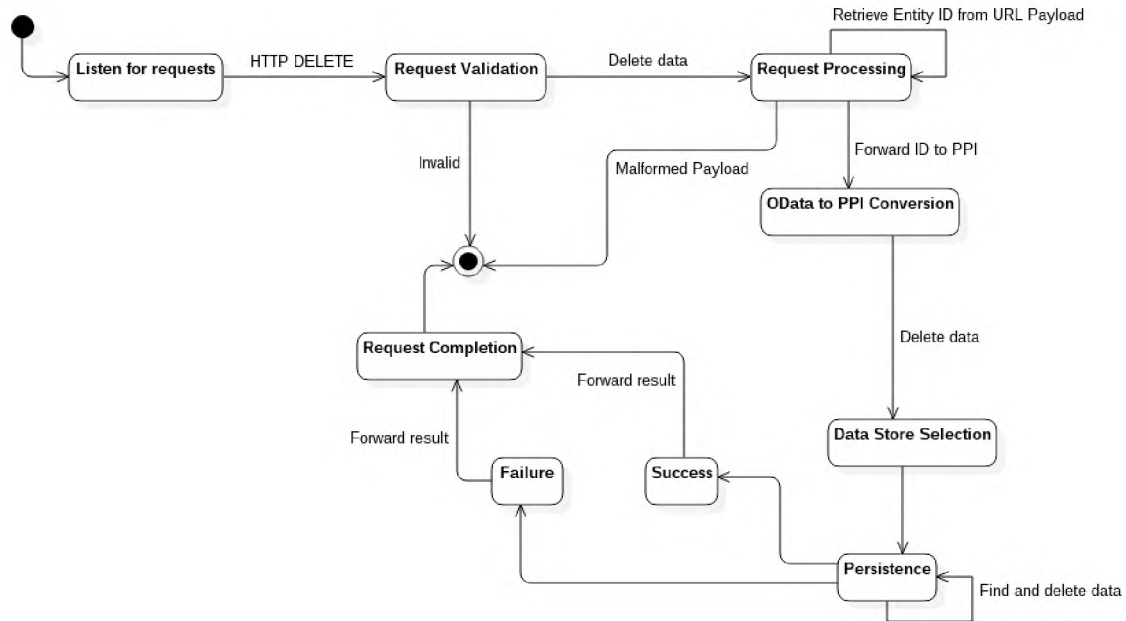


Figure 6.9: Defined CSDR behaviour when deleting data.

### 6.3.5 Subscribe

When the CSDR receives a GET subscription request, it extracts the identifier of the subscriber, data set and callback URL of the requesting FE. The callback URL is stored as the known address of the FE. It creates a PPI equivalent of a POST request in order to store the subscription. This request is then executed after the CSDR has selected the appropriate data store. This is followed by an acknowledgement of the request being granted or denied. The subscription process is presented in Figure 6.10.

### 6.3.6 Notify

To publish updates made on data to relevant FEs, the CSDR constructs an HTTP GET message. This message is sent toward the address specified in the OData callback URL. Figure 6.11 depicts the notification process of the CSDR. If the CSDR fails to deliver the notification, the cause of the failure is stored in a log file. The information in the file can further be used to determine the type of error that has occurred.



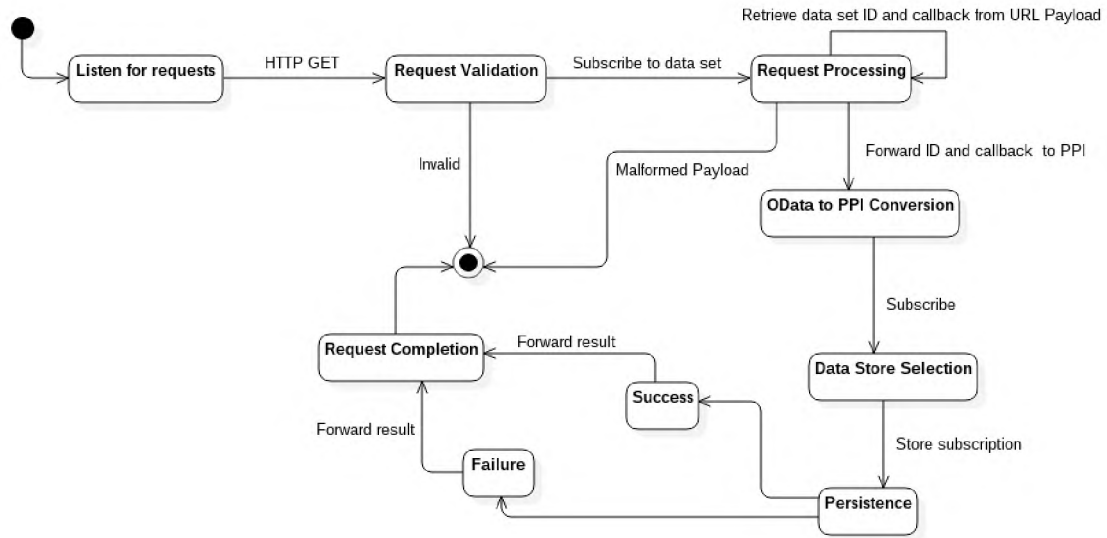


Figure 6.10: Defined CSDR behaviour when handling subscription.

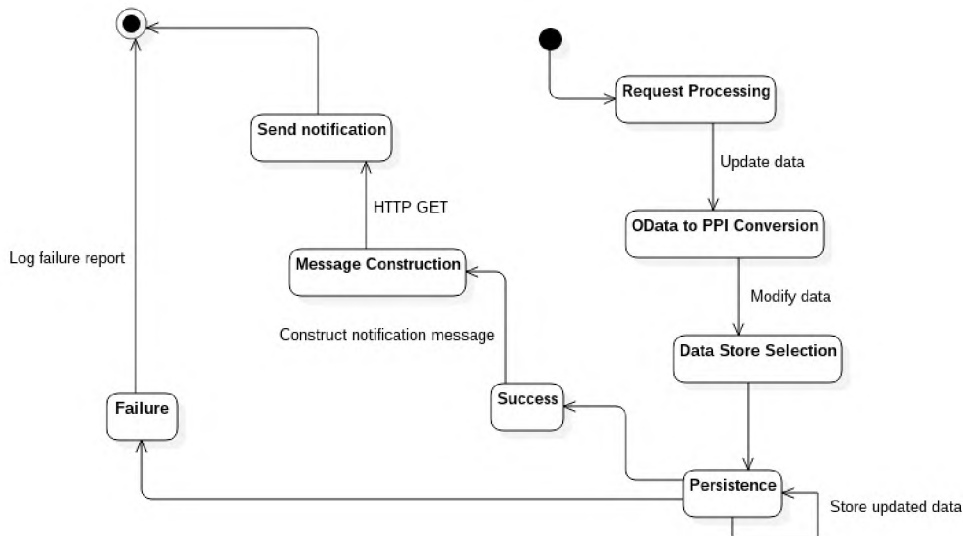


Figure 6.11: Defined CSDR behaviour when handling notification.

## 6.4 Summary

This chapter has presented a modular and multi-layered architectural design of the CSDR. The architecture provides access to data at three abstraction levels, namely: multi-interface, unified and low-level data access layers. Furthermore, the way in which the CSDR is accessed provides flexible and consistent schema management and query handling through OData. Other features such as access restrictions and polyglot persistence are also part of the CSDR design. The chapter has also highlighted the factors that have informed the CSDR design. The defined behaviour of the CSDR in response to CRUD and subscribe/notify requests was discussed toward the end of the chapter. The following chapter

presents a discussion on the implementation of a CSDR prototype that follows the specified design.

## Chapter 7

# Constructing a Prototype

The design of the CSDR has been presented in Chapter 6. This chapter discusses a proof-of-concept implementation of the components described in the previous chapter. The chapter starts out by presenting an overview of the tools and frameworks used in the implementation. This is followed by the discussion of polyglot persistence implementation. Next, is the implementation of the schema management and data processing components. The chapter further discusses the implementation of the access control component within the prototype. An analysis of the prototype is given toward the end of the chapter. This chapter concludes with a summary of the concepts that have been discussed.

## 7.1 Choice of Implementation

This section highlights the tools used in the development of the prototype. The CSDR is implemented as a data service that manages heterogeneous data that is used and generated by the distinct RCS services. Figure 7.1 presents an overview of the implementation. However, with the exception of the underlying data stores, the prototype was developed in a Linux Ubuntu (14.04 LTS) OS environment, using the Java programming language. Hence, the adopted tools are Java-based technologies.

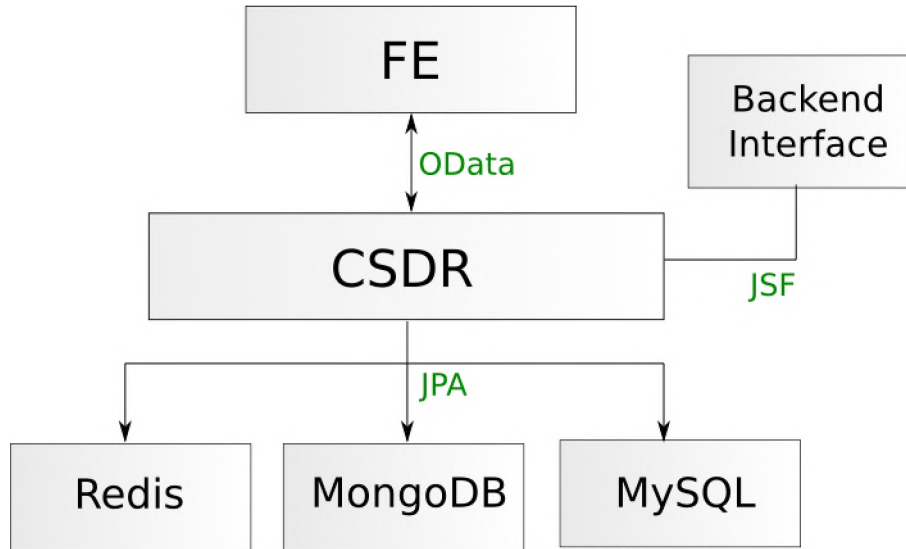


Figure 7.1: Overview of the prototype.

### 7.1.1 Maven

Maven is a tool for managing the development of a Java project from a common XML configuration file. This file is known as the Product Object Model (POM) document [3]. The core idea is to be able to manage the dependencies and build process of a Java project, in this case, a data service in a centralised manner. This simplifies the management of the CSDR codebase and eliminates the need to package dependencies with the prototype as a simple configuration file keeps track of them. The POM file used by the CSDR is presented in Appendix C.5.2.

### 7.1.2 Wildfly AS

This is an open source AS developed by Redhat.<sup>1</sup> It facilitates the deployment of Java-based application services. Wildfly version 10 [61], was adopted for managing the deployment of the CSDR prototype. The CSDR was deployed using the Wildfly deployment scanner, which is enabled by default. However, Wildfly allows the deployment of services in one of two modes: as a standalone server or a

---

<sup>1</sup><http://www.wildfly.org/>

managed domain. The domain version is built for cluster-awareness, whereas the standalone version is not. The CSDR was deployed in a standalone mode. Thus, the CSDR web application archive file was placed in the JBOSS\_HOME/standalone/deployments folder.

### 7.1.3 Apache Olingo

This Java Library can be used to create OData consumers and producers [18]. It provides support for the second and fourth versions of the OData protocol. Olingo is an open-source project that is released under the Apache license v2.0. Since OData is platform independent, Olingo is one of its numerous language-based library implementations. This makes it possible for REST clients to interact seamlessly with an OData producer.

### 7.1.4 Java Persistence API (JPA)

This is a Java specification that allows Java-based services to interact with data stores through an object-oriented DM. Real-world objects are represented as entities comprising a set of attributes, similar to the way OData represents objects as entity types.

There are two categories of JPA frameworks [24, 101]:

- **Object Relational Mapper (ORM)** works precisely for relational data stores.
- **Object NoSQL — Not Only SQL — Mapper (ONM)** works for a family of non-relational data stores. These group of data storage technologies can be also referred to as NoSQL stores.

A JPA framework provides loose-coupling between a data service and underlying storage technologies, thus avoiding vendor lock-in. The Kundera<sup>2</sup> framework was integrated with the CSDR to realise JPA functionality. The list of data stores that Kundera supports can be found at [2].

### 7.1.5 JavaServer Faces (JSF)

This is a framework which can be used to build web interfaces for Java-based applications [1]. It is part of the Java Enterprise Edition (EE) specifications and thus, can work directly with JPA models. JSF<sup>3</sup> separates the presentation of data from a business logic. The Primefaces JSF framework [85] was adopted by the CSDR.

---

<sup>2</sup>Kundera is an ONM framework that supports polyglot persistence.

<sup>3</sup>JSF pages are written in Extensible Hypertext Markup Languages (XHTML).

### 7.1.6 Data Stores

The prototype adopts three underlying data storage technologies to handle the different categories of data set used for the CDM.<sup>4</sup> These stores include: MongoDB (v3.2.9), Redis (v3.2.1) and MySQL (v5.6).

- **MongoDB** manages its data sets in a Binary JSON (BSON) format [28]. MongoDB documents are identified with keys which are of string primitive types. It provides a shell (named `mongo`), which allows an administrator to view and manage stored data. In addition to the command interface, it provides different REST APIs to clients, which can be integrated with external applications and services [66]. It adopts the replication technique for scaling, which enables the distribution of common data sets across different server nodes. However, it follows the BASE paradigm and hence, does not guarantee ACID properties when data is being manipulated.
- **Redis** is a key/value data store which also belongs to the non-relational family due to its simplistic nature. Similar to MongoDB, clients can access the Redis server through different APIs which are available based for different chosen programming languages [87]. Redis provides data structures such as lists, sets, bitmaps, and hashes for storing its values, which are accessed using distinct keys.
- **MySQL** is a relational storage technology that was adopted to cater for the CSDR's transaction-oriented operations. In other words, it guarantees ACID properties when data is being manipulated. It allows a client to view and manipulate stored data through the SQL query language. This makes it different from MongoDB and Redis, which expose multiple APIs to clients.

## 7.2 Implementing Polyglot Persistence

The CSDR uses Kundera to access all three data stores through a common JPA interface. This interface performs the role of the PPI described in the design chapter. Hence, eliminating the need for the CSDR to access Redis, MongoDB and MySQL through different APIs. This section discusses the various steps taken to set-up this interface for heterogeneous subscriber data sets within the CSDR.

### 7.2.1 Creation of the Domain Model

The prototype considers the four categories of heterogeneous RCS subscriber data. They include: subscription profiles, XCAP configuration documents, call and message logs, and session-related data.

---

<sup>4</sup>The contents of the CSDR that are discussed in Section 7.2.1

### 7.2.1.1 Subscription Profile

Subscription profiles are used over an IMS network to authenticate and authorise RCS subscribers before using the service offerings. IMS also needs an orchestration logic for executing those services. This logic is defined within the subscription profile using the initial filter criteria (iFC) element described in [11]. The CSDR stores this profile in MongoDB as a subscription document. Furthermore, the subscription profile changes occasionally. For example, the primary identity of the subscriber<sup>5</sup> may not change at all, and service orchestration rules do not change frequently. Thus, storing this data in MongoDB allows frequent reads and lower rates of write operations, which are eventually consistent. Other elements of the subscription profile considered are: an RCS AS, dedicated S-CSCF, visited networks, trigger points,<sup>6</sup> authentication and authorisation data sets. They are presented in Figure 7.2. However, Visited Network information is stored in MySQL. More importantly, these profile data sets being stored in the CSDR were autonomously managed by an HSS service within IMS.

### 7.2.1.2 XCAP Document

RCS adopts XDMS as a repository for managing XML documents, which are exposed through the XCAP protocol. XDMS supports different documents such as presence list and PNBs, which are structured through different Application Usage (AppUsage) definitions. An AppUsage defines the XML schema for the data used by the corresponding service along with other key pieces of information [92]. Therefore, creating a sample data set for XCAP documents with the CDM adds heterogeneity to the type of data managed by the prototype. The following metadata were considered for XCAP documents that are stored in the CSDR:

- Application Unique ID (AUID); an identifier that describes the document's AppUsage.
- XCAP User Identifier (XUI); a string, valid as a path element in an HTTP URI, that is associated with each subscriber served by the XDMS.
- Entity tag; allows a number of conditional operations to be performed against an XCAP document.
- Document name; identifies a specific XCAP resource.
- Scope; which makes a document available to all subscribers or a specific subscriber.

However, RFC 4825 [92] presents an extensive list of metadata associated with XCAP documents. In addition, the CSDR stores these XML documents and its

---

<sup>5</sup>This can include the IMS Private Identity (IMPI), and IMS Public Identity (IMPU).

<sup>6</sup>Set of conditions used by the iFC to define how RCS service enablers can be invoked.

metadata in MongoDB. Each XML document is converted into a primitive string data type, which can be validated by the defined AppUsages when required. This data is represented as *XcapDocument* in Figure 7.2.

### 7.2.1.3 Call and Message Logs

The CSDR also manages logs generated from RCS voice and messaging services. These logs can be used to charge the subscriber for service usage. Further, having an overview of such information is also important to the CSDR. This data is represented through the *CallDetailRecord* component in Figure 7.2. Moreover, logs are transactional in nature, as consistent data is required for accountability at any given period. In other words, ACID compliance is required for the management of these data set. Therefore, this makes MySQL a suitable candidate for managing such data. Consequently, changes made to this log can be committed by the CSDR when successful and rolled back otherwise.

### 7.2.1.4 Session-related Data

During dialogues, RCS clients generate transient data in the form of session-related information, which are discarded when the session is terminated. This information can be used for error tracking and reporting. For example, when a phone call terminates without either party pressing the end button. Data gathered while the session was active can provide insights into why the network cancelled the call. This is useful when other means of troubleshooting have been explored. Considering the ACID properties required of the CSDR, session-related data sets are modified or updated as the subscriber engages in more dialogues. Thus, using Redis helps the CSDR manage this portion of subscriber data with little emphasis on transactional integrity. This data is represented in the *ClientSessionData* component in Figure 7.2.

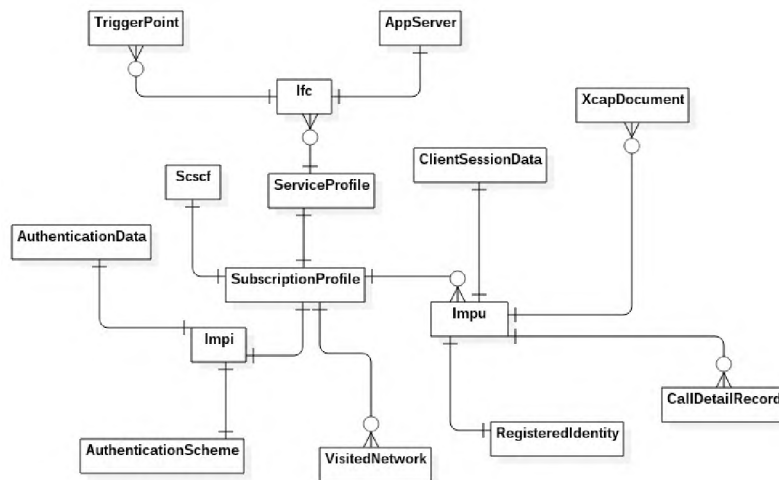


Figure 7.2: The Domain Model as an Entity Relationship Diagram.



### 7.2.2 Mapping the Domain Model across Stores

JPA represents the different data objects mentioned in Section 7.2.1, as distinct entities. Each entity together with its relationships collectively defines the domain model of the CSDR. Since ONMs and ORMs convert object-oriented data to a specific DM, multiple data stores can be accessed through the same domain model. Hence, eliminating the need to re-write code to execute CRUD operations on each underlying data store. Therefore, to represent each data store entity with JPA, the mapping technique described in Table 7.1 was used. This shows how Kundera supports polyglot persistence by mapping JPA entities to corresponding models in the MySQL, MongoDB, and Redis data stores. Consequently, this provides a uniform way to access multiple storage technologies.

Table 7.1: Multi-Store JPA Data Model Mapping.

Data Store	Entity Definition	Entity Property	Instance
MySQL	Table	Column	Row
MongoDB	Collection	Field	Document
Redis	Application Type	Key	Value

Once the data objects have been successfully mapped to corresponding entities, the CSDR can reliably execute CRUD requests over the JPA interface. Thus, all subscriber-related data are retrieved as objects with individual properties. This is done irrespective of how the objects are represented in the underlying data stores. Section 7.2.4 further discusses how the properties of these entities are mapped within the CSDR.

### 7.2.3 Configuring the JPA Interface

To execute CRUD requests against the defined domain models, Kundera needs to be configured to support the underlying data stores. This configuration occurs in two steps: Firstly, a *persistence.xml* file is created containing the connection information for the data stores. Secondly, Kundera dependency information is added to the Maven configuration file.

To configure the *persistence.xml* file, distinct persistence units are defined for each data store. A persistence unit presents a description of the connected data stores, a pertinent domain model, and specific properties as represented in Listing 7.1. It groups pertinent JPA entities for each underlying data store. This simplifies the multi-store persistence as each data source only manages relevant entities. Kundera provides this grouping information to its Entity Manager (EM) at runtime. The EM is aware of the different persistence units that Kundera provides to the CSDR and therefore, manages the direct execution of data manipulation operations over the stores. Hence, the EM performs the role of the PPM as it serves as the core module for integrating polyglot persistence within the CSDR.

Listing 7.1: A version of the polyglot persistence.xml file.

```
<persistence>

  <persistence-unit name="csdr-mysql">
    <provider>com.impetus.kundera.KunderaPersistence</
      provider>
    <properties>  <!-- Connection Properties -->
    </properties>
  </persistence-unit>

  <persistence-unit name="csdr-mongo">
    <provider>com.impetus.kundera.KunderaPersistence</
      provider>
    <properties> </properties>
  </persistence-unit>

  <persistence-unit name="csdr-redis">
    <provider>com.impetus.kundera.KunderaPersistence</
      provider>
    <properties> </properties>
  </persistence-unit>

</persistence>
```

Furthermore, adding dependency information to Kundera is also a trivial task. However, Kundera provides different client libraries for accessing the data stores. Hence, for the CSDR to interact with MongoDB, Redis and MySQL, the following is added to the POM file:

Listing 7.2: Maven dependencies for the data stores.

```
<dependency>
  <groupId>com.impetus.kundera.client</groupId>
  <artifactId>kundera-mongo</artifactId>
  <version>${kundera.version}</version>
</dependency>

<dependency>
  <groupId>com.impetus.kundera.client</groupId>
  <artifactId>kundera-redis</artifactId>
  <version>${kundera.version}</version>
</dependency>

<dependency>
  <groupId>com.impetus.kundera.client</groupId>
  <artifactId>kundera-rdbms</artifactId>
  <version>${kundera.version}</version>
</dependency>
```

- **groupId** specifies that the library is a Kundera client.
- **artifactId** denotes the client-type for MongoDB, Redis and MySQL respectively.
- **version** states the version of the client, which is 3.4.

## 7.2.4 Exposing the Data through JPA

Once the domain model was created and the configuration settings were completed, the following step was providing access to these data sets through JPA. This is where the PPI becomes relevant. Each of the JPA entities previously highlighted are assigned specific handler classes. These classes handle the CRUD operations for the respective entity through the EM. The following table below presents the list of handler classes provided within the CSDR.

Table 7.2: JPA Handler Classes.

Handler Classes	Description
AppServerProvider	Works with the AppServer entity
CallDetailRecordProvider	Works with the CallDetailRecord entity
ClientSessionDataProvider	Works with ClientSessionData entity
IfcProvider	Works with the Ifc entity
ImpuProvider	Works with the Impu entity
ImpiProvider	Works with the Impi entity
RegisteredIdentityProvider	Works with the RegisteredIdentity entity
ScscfProvider	Works with the Scscf entity
ServiceProfileProvider	Works with the ServiceProfile entity
SubscriptionProfileProvider	Works with the SubscriptionProfile entity
XcapDocumentProvider	Works with the XcapDocument entity
VisitedNetworkProvider	Works with the VisitedNetwork entity

Additionally, the classes convert JPA models to corresponding EDM entities which are used by the CSDR. The table below shows how the JPA entities were mapped to EDM entities:

Table 7.3: JPA and EDM Mapping

OData EDM element	JPA Representation (with Annotation)
Entity Type	@Entity
Complex Type	@Embeddable
Enum Type	@Enumerated
Entity Container	@PersistenceContext
Property	@Column
Property Reference	@Id

Navigation Property	toOne and toMany Relationships; @OnetoOne, @OnetoMany, @ManytoOne, @ManytoMany.
---------------------	---

## 7.3 Providing Schema Management

Once the domain model had been created and the JPA interface configured, the CSDR required a CDM that could present the conceptual view of the heterogeneous data set. The CDM here is essentially an OData EDM for the CSDR, which was created with the Olingo library. This section discusses the implementation of the modules that enable the management of the CSDR's CDM. Firstly, the section explains how the Schema Descriptor was implemented. Secondly, it describes how the CDM is managed by the data producer, which works closely with the EM. Finally, it describes the implementation of the backend interface.

### 7.3.1 Schema Descriptor

The Schema Descriptor describes the content of the CDM through service and metadata documents. The CDM was created using the model depicted in Figure 7.2. The service documents show the list of entity sets available within the CSDR as depicted below. Snapshots of both documents are shown in Figure 7.3 and Figure 7.4. However, the CDM was defined through the implementation of the *CsdrEDMProvider*, which extends the *CsdlAbstractEdmProvider* class. This class implemented the methods presented in Table 7.4.

```

    <EntityType Name="Ifc">
      <Key>
        <PropertyRef Name="ID"/>
      </Key>
      <Property Name="ID" Type="Edm.Int32"/>
      <Property Name="Sid" Type="Edm.String"/>
      <Property Name="Name" Type="Edm.String"/>
      <Property Name="Priority" Type="Edm.String"/>
      <Property Name="ProfilePartIndicator" Type="com.inted.csdr.ProfilePartIndicator"/>
      <NavigationProperty Name="ApplicationServer" Type="com.inted.csdr.ApplicationServer" Nullable="false"/>
      <NavigationProperty Name="TriggerPoint" Type="com.inted.csdr.TriggerPoint" Nullable="false"/>
    </EntityType>
    <EntityType Name="ImsPrivateIdentity">
      <Key>
        <PropertyRef Name="ID"/>
      </Key>
      <Property Name="ID" Type="Edm.Int32"/>
      <Property Name="Sid" Type="Edm.String"/>
      <Property Name="Uri" Type="Edm.String"/>
      <Property Name="AuthenticationData" Type="com.inted.csdr.AuthenticationData"/>
      <Property Name="AuthenticationScheme" Type="com.inted.csdr.AuthenticationScheme"/>
    </EntityType>
    <EntityType Name="ImsPublicIdentity">
      <Key>
        <PropertyRef Name="ID"/>
      </Key>
      <Property Name="ID" Type="Edm.Int32"/>
      <Property Name="Sid" Type="Edm.String"/>
      <Property Name="SipUri" Type="Edm.String"/>
      <Property Name="IdentityType" Type="com.inted.csdr.PublicIdentityType"/>
      <Property Name="ImsUserState" Type="com.inted.csdr.ImsUserState"/>
      <Property Name="BarringIndication" Type="Edm.Boolean"/>
      <Property Name="CanRegister" Type="Edm.Boolean"/>
      <Property Name="DisplayName" Type="Edm.String"/>
      <Property Name="PsiActivation" Type="com.inted.csdr.PsiActivation"/>
      <NavigationProperty Name="SessionData" Type="Collection{com.inted.csdr.ClientSessionData}"/>
    </EntityType>

```

Figure 7.3: Snapshot of the CSDR Metadata Document.

```

▼<app:collection href="ClientSessionDataSet" metadata:name="ClientSessionDataSet">
  <atom:title>ClientSessionDataSet</atom:title>
</app:collection>
▼<app:collection href="FeApplicationDataSet" metadata:name="FeApplicationDataSet">
  <atom:title>FeApplicationDataSet</atom:title>
</app:collection>
▼<app:collection href="IfcSet" metadata:name="IfcSet">
  <atom:title>IfcSet</atom:title>
</app:collection>
▼<app:collection href="ImsPrivateIdentities" metadata:name="ImsPrivateIdentities">
  <atom:title>ImsPrivateIdentities</atom:title>
</app:collection>
▼<app:collection href="ImsPublicIdentities" metadata:name="ImsPublicIdentities">
  <atom:title>ImsPublicIdentities</atom:title>
</app:collection>
▼<app:collection href="MergedDataSet" metadata:name="MergedDataSet">
  <atom:title>MergedDataSet</atom:title>
</app:collection>
▼<app:collection href="ScscfSet" metadata:name="ScscfSet">
  <atom:title>ScscfSet</atom:title>
</app:collection>
▼<app:collection href="PrefScscfSets" metadata:name="PrefScscfSets">
  <atom:title>PrefScscfSets</atom:title>
</app:collection>
▼<app:collection href="RegisteredIdentities" metadata:name="RegisteredIdentities">
  <atom:title>RegisteredIdentities</atom:title>
</app:collection>
▼<app:collection href="ServiceProfiles" metadata:name="ServiceProfiles">
  <atom:title>ServiceProfiles</atom:title>
</app:collection>
▼<app:collection href="SubscriptionProfiles" metadata:name="SubscriptionProfiles">
  <atom:title>SubscriptionProfiles</atom:title>
</app:collection>
▼<app:collection href="TriggerPoints" metadata:name="TriggerPoints">
  <atom:title>TriggerPoints</atom:title>
</app:collection>
▼<app:collection href="VisitedNetworks" metadata:name="VisitedNetworks">
  <atom:title>VisitedNetworks</atom:title>
</app:collection>
▼<app:collection href="XcapDocuments" metadata:name="XcapDocuments">
  <atom:title>XcapDocuments</atom:title>
</app:collection>
</app:workspace>
</app:service>

```

Figure 7.4: Snapshot of the CSDR Service Document.

Table 7.4: Definition of the CDM elements.

Method	Description
getEnumType()	Defines the the enumerated types.
getEntityType()	Defines the entity types.
getComplexType()	Defines the complex types.
getEntitySet()	Defines the entity sets. Additionally, the relationships between the entity types are specified in this method.
getSchemas()	Defines the schemas, but only one in this case.
getEntityContainer()	Defines the container, which hosts the entity sets.
getEntityContainterInfo()	Gives some metadata about the entity container, which is displayed in the service document.

### 7.3.2 Data Producer

This module manages both the JPA and the CDM models. It is aware of the defined entity types within the CDM, and the respective handler classes exposed through JPA. It receives CRUD request from a processor and determines the appropriate technique to handle the request. This bridging functionality allows it to navigate different CDM properties and leverage the JPA to fetch relevant data sets. A *DataProducer* class was created to handle the activities summarised in the following steps:

### 7.3.3 Backend Interface

This is a provisioning interface that can be used to manipulate the contents of the CSDR. This was implemented with the aid of the JSF Primefaces framework to expose data management capabilities to a user. A set of pages can be accessed through a web browser, thus serving as the administrative interface to the CSDR. Primefaces was configured within the CSDR through the web.xml servlet configuration file. The configuration parameters are shown in Listing 7.3 and a snapshot of the interface is presented in Figure 7.5.

Listing 7.3: Primefaces configuration

```
<context-param>
  <param-name>javax.faces.PROJECT_STAGE</param-name>
  <param-value>Development</param-value>
</context-param>

<context-param>
  <param-name>primefaces.THEME</param-name>
  <param-value>omega</param-value>
</context-param>
```

- **context-param** specifies a context parameter that can be accessed within the CSDR.
- **param-name** specifies the name of the parameter. Two were defined in this case: Project stage and Primefaces theme respectively.
- **param-value** specifies a value for the parameter. The first value indicates the project is in a development stage while the second indicates that the CSDR adopts an omega theme for its user interface.

The screenshot displays the CSDR Backend Interface. On the left is a sidebar menu with the following items: Subscription Profile, Service Profile, Trigger Point, Scsf, Application Server, Visited Network, Client Session Data, Saved XCAP Documents, Registered Subscriber Identities, and Front Ends. The main content area is titled 'Add Subscription Profile' and contains several form fields: Name, DSAI Value, Service Profile, and Scsf. Below these is a 'Private Identity' section with fields for URI and Secret Key (Password). Further down is a table for 'Attached IMS Public Identities' with columns: Identity Type, SIP URI, Can Register, and Is Barr. The table currently shows 'No records found.' Below the table is an 'Attach IMS Public Identity' section with a SIP URI field.

Figure 7.5: Snapshot of the CSDR Backend Interface.

## 7.4 Implementing the Data Processing Component

This section discusses the implementation of the request processors and query handlers within the CSDR. The CSDR uses both categories of processors to respond to CRUD requests that are received from the registered FEs. Thus, the section starts with a description of the implementation of the request processors, which is followed by that of the query handlers.

### 7.4.1 Request Processor

Requests received by the CSDR are coordinated by an *ODataHttpHandler*, which invokes the appropriate processor. The request processors<sup>7</sup> that were implemented are shown in Figure 7.6. The processors extract the contents of an OData request through the Olingo *UriInfo* class. UriInfo breaks down the contents of the requests into segments, which are further used to identify the requested type. For a request to be processed, the payload-type<sup>8</sup> must be supported by the CSDR. Once the payload-type is validated, the requests are processed in the following steps: First, the CRUD request is sent to the Data Producer. The processor waits for the producer's response. If the request is a read request, the CSDR forwards the response to the defined set of query handlers. Once the query handlers have been applied, the processor converts the resulting object into an *InputStream* object through an *ODataSerializer*. This is then sent to the requesting consumer (or FE), over the network in a JSON format.

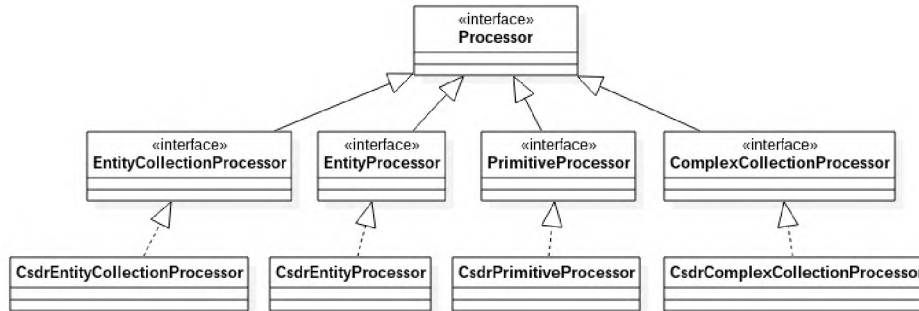


Figure 7.6: Class Diagram of the CSDR Request Processors.

### 7.4.2 Query Handler

The CSDR implements the query handlers<sup>9</sup> depicted in Figure 7.7. These handlers are applied on the entity collections forwarded by the request processors. The handlers use the *UriInfo* class to check if their corresponding options are specified

<sup>7</sup>Appendix A.4 gives a description of these processors.

<sup>8</sup>Discussed in Section 4.3.3

<sup>9</sup>The handlers cater for the query options described in Table 4.2.

in the OData request. If true, the query option is executed against the entity collection and returned back to the request processor.

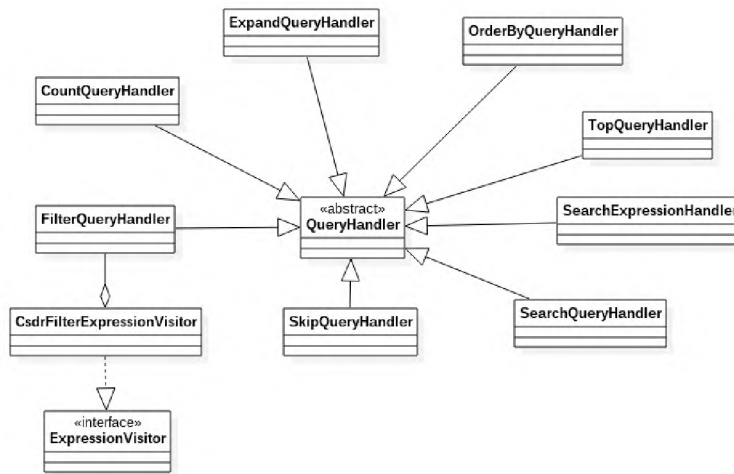


Figure 7.7: Class Diagram of the CSDR Query Handlers.

## 7.5 Implementing Access Control

The prototype achieves authorisation through the use of servlet filters. The servlet filter sits between traffic originating from an FE or backend interface and the CSDR. This was implemented with an *AuthFilter* servlet, which implements the *javax.servlet.Filter* interface. *AuthFilter* was also configured in the *web.xml* file by adding the `<filter>` and `<filter-mapping>` parameters. The procedure for the authentication filter is represented in Listing 7.4. The servlet extracts the IP address of the FE which has sent the request and checks if a binding exists within the CSDR. If that happens the CSDR has the ability to deny requests if the binding is not found. The CSDR uses the MySQL data store to keep track of all FE-to-IP bindings.

Listing 7.4: Pseudo-code for the *AuthFilter* Procedure

```

doFilter(request)
    Extract IP address from request
    Find IP Address mapping

    if(matchFound)
        Forward request to the ODataHttpHandler
    else
        Block request
  
```



## 7.6 Discussion

Choosing three storage technologies and four categories of data emphasises the capability of the CSDR in managing heterogeneous, transient, permanent, transactional and non-transactional data sets. Polyglot persistence was achieved through a JPA framework. This allows the CSDR to simplify heterogeneous data access at the PPI level. This section discusses the implications of using this approach to managing multi-store persistence.

To begin, implementing multiple APIs within the CSDR can become unmanageable as more storage technologies are integrated. This can generate a significant amount of boilerplate code. Furthermore, each OData entity would require an abstract class that allows the CSDR to perform CRUD operations. One may also have to manually create code that allows you to write and execute queries for each data store. Hence, using JPA eliminates this complication while providing the freedom to switch between data stores without having to re-write the queries at lower, data access layer.

Managing consistency across multiple stores can be a complex process — especially when the stores do not provide adequate transaction mechanisms. In other words, the stores do not provide a reliable way to ensure ACID properties when handling requests. Although the CSDR leaves transaction management to the data stores, some consistency is required for the logical repository itself. To enforce such properties, the CSDR has to be explicitly programmed with those constraints. When done that way, the CSDR is prone to management inconsistencies. Using the JPA technology eliminates this problem as data manipulation occurs through a standardised API which satisfies consistency in data management.

Furthermore, JPA allows the CSDR to switch between similar and different storage technologies with minimal changes to the initially defined domain model. For example, the CSDR can switch from a MySQL to PostgreSQL data store, which both adopt relational models or from MongoDB to Neo4j, which adopt document and graph models respectively. However, the introduction of data storage systems at this layer forces a trade-off between flexibility and complexity. That is, the CSDR should support an additional storage technology if there is certainty that the existing stores can not handle the data access and modelling requirements.

Moreover, the CSDR exposes its data set to the FEs through the EDM. Therefore, when the CSDR is creating, updating or deleting data, it simply extracts the EDM entity, converts it to an equivalent JPA entity before forwarding the entity to the EM. Nevertheless, EDM schema changes can render some data stores obsolete. In other words, changing or updating a data model can result in the CSDR not utilising some portion of its data. JPA defines a Cascade Type mechanism that handles dependencies between entities. The mechanism provides a group of five conditions namely DETACH, MERGE, PERSIST, REMOVE, and REFRESH. This implies that changes made to the parent entity will affect the child entity. Therefore, a delete or update request sent through OData may trigger JPA Cascade Type conditions where defined.

However, using JPA also provides some limitations to our implementation. First, JPA is a framework for Java applications. Implementing polyglot persistence with a CSDR developed using a different language may require other persistence mechanisms. The CSDR adopts a JPA technology as a proof-of-concept that OData can assist in managing and exposing data gathered from heterogeneous sources. Second, implementing JPA requires technical competence in handling issues such as entity inheritance and relationship complexities. An adequate understanding of JPA is required to handle growing data sets with polyglot persistence integration effectively.

## 7.7 Summary

This chapter has presented the development of the CSDR prototype. The implementation was managed with Maven and it persists data across MySQL, Redis and MongoDB stores through the Kundera JPA framework. The techniques used for mapping between JPA models and EDM were also discussed. The prototype also implements request processors and query handlers, which are used to manage requests received from FEs. A simple access control mechanism was implemented for the prototype using the Java Servlet Filter interface. Furthermore, the data producer, schema manager and backend interface implementations were also presented. Finally, the implications of using JPA for polyglot persistence and exposing CSDR data with EDM were discussed. The next chapter presents the experiments that were carried out over a network with the aid of this unified data repository.

## Chapter 8

# Integration within an IMS Testbed

The previous chapter discussed the development of a CSDR prototype. Since RCS services can only interact with the CSDR through defined FE applications, it was deemed necessary to model some of these applications in order to evaluate this work. Further, it is also necessary to ensure that the CSDR and the FEs that have been developed can interwork in an IMS environment since RCS relies on IMS. Therefore, this chapter focuses on the deployment of the CSDR and its defined FEs within an IMS network testbed. The chapter starts out by presenting an overview of the testbed and its components. The next section presents the description, implementation and integration testing of an HSS FE. This is followed by that of the Telephony and Messaging FEs, which handle SIP-based RCS functionalities. Thereafter, the integration of the XCAP FE, which essentially handles XML data used by RCS services is discussed. The contents of the CSDR are interrogated toward the end of the chapter, to further emphasise the convergence of RCS subscriber data. The chapter concludes with a summary of the activities discussed.

## 8.1 Overview

To realise an experimental environment, an open source IMS implementation was utilised — the OpenIMSCore [32]. OpenIMSCore is currently distributed by Core Network Dynamics GmbH.<sup>1</sup> The software provides a software version of the P-CSCF, I-CSCF, S-CSCF, and HSS components of the Core IMS subsystem. The OpenIMSCore HSS is termed Fokus Fraunhofer HSS (FHoSS). In addition to the OpenIMSCore, clients were also used to interact with the FEs. These are discussed in subsequent sections.

Figure 8.1 illustrates the testbed environment which was set up on a Linux Ubuntu OS. The Restcomm SIP servlets technology [90] was also integrated with the testbed to realise SIP AS features. In essence, SIP servlets were used to build services, which respond to SIP requests over IMS. This will be discussed further in Section 8.3. Additionally, Restcomm jdiameter [89] was used to facilitate Diameter-based interactions in the network. Doubango<sup>2</sup> Boghe IMS [37] and IMSDroid [38] clients were used to simulate SIP user agents. The former works on a Windows desktop OS while the latter works on either an Android smartphone or tablet. This is to validate interoperability between RCS clients installed on different platforms.

The Postman REST client [84] was used to mimic the behaviour of an XCAP client toward the lightweight XCAP FE discussed in Section 8.4. Postman was also used to send HTTP requests during the execution of activities described in Section 8.5. Finally, the interactions between the testbed components were analysed using Wireshark (v2.2.0) [5], which is a network protocol analyser.

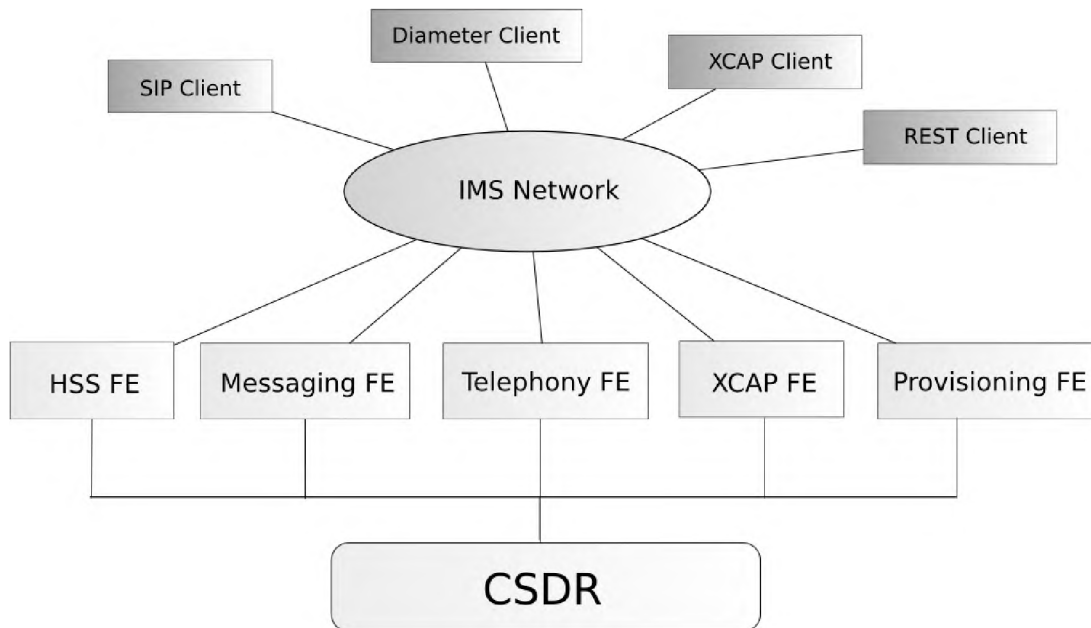


Figure 8.1: Overview of the experimental setup.

<sup>1</sup><http://www.corenetdynamics.com/>

<sup>2</sup>An open source 3GPP IMS/LTE framework for embedded systems [39]

## 8.2 HSS FE

The HSS keeps subscription related information for RCS services as shown in Figure 2.2. Transforming this repository into a stateless application that interacts with the CSDR was a non-trivial task. The FE implementation was derived from the FHoSS component of the OpenIMSCore. In other words, code from the FHoSS was re-used and re-engineered to achieve the HSS functionalities that were used to test the CSDR.

### 8.2.1 Description

The HSS FE supports two Diameter interfaces: Sh and Cx. The Cx interface is used by the I-CSCF and the S-CSCF in the network. The Sh interface allows an AS to query the HSS for subscriber data. Data exchanged over both interfaces use predefined XSDs and operations specified in [11] and [12] respectively. The Cx and Sh operations supported by the HSS FE are presented in Table 8.1

Table 8.1: Interface operations supported by the HSS FE.

Interface	Operation	Description
Cx	User-Authorization-Request (Cx-UAR)	This is used by the I-CSCF to check whether a specific IMS URI is allowed to roam.
Cx	Location-Info-Request (Cx-LIR)	This is also used by the I-CSCF to fetch the identity of a designated S-CSCF when a SIP call request has been received by the network.
Cx	Multimedia-Auth-Request (Cx-MAR)	The S-CSCF uses this message to authenticate a subscriber.
Cx	Server-Assignment-Request (Cx-SAR)	The S-CSCF uses this message to download the subscription profile from the HSS.
Sh	User-Data-Request (Sh-UDR)	A network service fetches subscriber data from the HSS through this message.

## 8.2.2 Implementation

Both interfaces were implemented with the aid of the Restcomm jdiagramer stack version 1.7.0.79. Restcomm provides a multiplexer which coordinates the deployment of multiple Diameter-based applications. The multiplexer exposes the Diameter stack which can be shared among multiple applications. This allows the HSS FE to support both Cx and Sh interfaces. The FE was deployed using an embedded Jetty AS (v9.3.9) [41]. Restcomm jdiagramer was configured through a config-server.xml file. Furthermore, the clients that interact with the HSS FE are considered diameter peers. Once the HSS FE has started, the S-CSCF and I-CSCF attempt to synchronise with the HSS FE by sending a Capabilities Exchange Request (CER). This is done to ensure that both the HSS and CSCFs support a common Diameter application identifier, which is depicted in Listing 8.1 as an `<ApplicationID>`. The listing also presents the interfaces that are defined in the config-server.xml. It is important to note that for each request sent to the HSS FE, a corresponding data query request is sent to the CSDR to fetch the pertinent data. Thus, allowing the HSS FE to perform its functions while its data resides in the CSDR.

Listing 8.1: Diameter interfaces in config-server.xml

```
<Configuration>
<LocalPeer>
  <URI value="aaa://192.168.23.4:3868" />
  <IPAddresses>
    <IPAddress value="192.168.23.4" />
  </IPAddresses>
</LocalPeer>

<Network>
<Peers>
  <Peer name="aaa://127.0.0.1:1812" /> <!-- Sh -->
  <Peer name="icscf.open-ims.test:3869" /> <!-- I-CSCF -->
  <Peer name="scscf.open-ims.test:3870" /> <!-- S-CSCF -->
</Peers>

<!-- CSCFs -->
<Realm name="open-ims.test" peers="icscf.open-ims.test,scscf.open-ims.test" local_action="LOCAL"
dynamic="false" exp_time="1">
  <ApplicationID>
    <VendorId value="10415" />
    <AuthApplId value="16777216" />
    <AcctApplId value="0" />
  </ApplicationID>
</Realm>

<!-- Sh App -->
<Realm name="open-ims.test" peers="127.0.0.1" local_action="LOCAL"
dynamic="false" exp_time="1">
  <ApplicationID>
    <VendorId value="10415" />
    <AuthApplId value="16777217" />
    <AcctApplId value="0" />
  </ApplicationID>
</Realm>
</Network>
</Configuration>
```

## 8.2.3 Results

### 8.2.3.1 Scenario 1 — RCS Client Registration

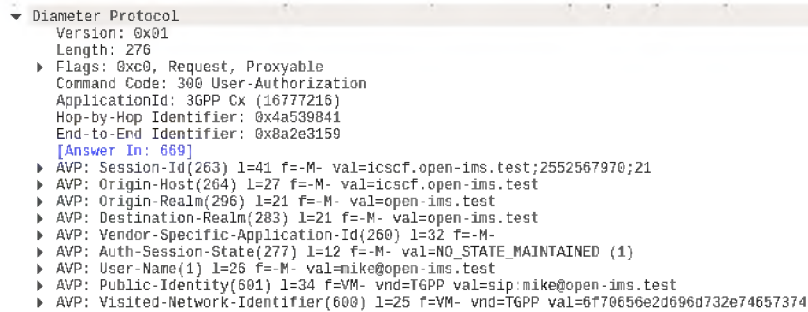
In this scenario, a subscriber tries to register with the network through an RCS Client (which is modelled as a pure SIP client). The subscriber enters their IMPI,

IMPU, password and realm into the client. The client sends the request to the P-CSCF and once the P-CSCF validates the request, it forwards the message to the I-CSCF for further processing. There are four Diameter requests executed during this registration process: UAR, MAR, SAR and LIR respectively.

## Cx-UAR

(Step<sub>1</sub>) The P-CSCF forwards the message to the I-CSCF.

(Step<sub>2</sub>) The I-CSCF contacts the HSS FE.



```

Diameter Protocol
  Version: 0x01
  Length: 276
  Flags: 0xc0, Request, Proxyable
  Command Code: 300 User-Authorization
  ApplicationId: 3GPP Cx (16777216)
  Hop-by-Hop Identifier: 0x4a539841
  End-to-End Identifier: 0x8a2e3159
  [Answer In: 669]
  AVP: Session-Id(263) l=41 f=-M- val=icscf.open-ims.test;2552567970;21
  AVP: Origin-Host(264) l=27 f=-M- val=icscf.open-ims.test
  AVP: Origin-Realm(296) l=21 f=-M- val=open-ims.test
  AVP: Destination-Realm(283) l=21 f=-M- val=open-ims.test
  AVP: Vendor-Specific-Application-Id(260) l=32 f=-M-
  AVP: Auth-Session-State(277) l=12 f=-M- val=NO_STATE_MAINTAINED (1)
  AVP: User-Name(1) l=26 f=-M- val=mike@open-ims.test
  AVP: Public-Identity(591) l=34 f=VM- vnd=T6PP val=sip:mike@open-ims.test
  AVP: Visited-Network-Identifier(600) l=25 f=VM- vnd=T6PP val=6f70656e20696d732e74657374
  
```

Figure 8.2: UAR from S-CSCF to HSS FE.

(Step<sub>3</sub>) The HSS FE sends a GET request to the CSDR for subscriber data using the specified IMPU. This is done to generate an answer to the UAR request.

(Step<sub>4</sub>) The CSDR responds with a JSON-formatted *RegisteredIdentity* OData entity that contains the required data.



```

JavaScript Object Notation: application/json
  Object
    Member Key: @odata.context
      String value: $metadata#RegisteredIdentities
      Key: @odata.context
    Member Key: value
      Array
        Object
          Member Key: ID
            Number value: 8
            Key: ID
          Member Key: Impi
            String value: mike@open-ims.test
            Key: Impi
          Member Key: Impu
            String value: sip:mike@open-ims.test
            Key: Impu
          Member Key: AuthKey
            String value: mike
            Key: AuthKey
          Member Key: AuthOp
            String value: 00000000000000000000000000000000
            Key: AuthOp
          Member Key: AuthAmf
            String value: 0000
            Key: AuthAmf
          Member Key: AuthSqn
            String value: 000000000000
            Key: AuthSqn
          Member Key: AuthScheme
            String value: Auth_Scheme_MD5
            Key: AuthScheme
          Member Key: AuthLineId
            Null value
            Key: AuthLineId
          Member Key: AuthIpAddress
            Null value
            Key: AuthIpAddress
  
```

Figure 8.3: HSS FE and CSDR interaction.

(Step<sub>5</sub>) The HSS FE then sends a Diameter success response to the I-CSCF.

```

▼ Diameter Protocol
  Version: 0x01
  Length: 144
  ▶ Flags: 0x40, Proxyable
  Command Code: 300 User-Authorization
  ApplicationId: 3GPP Cx (16777216)
  Hop-by-Hop Identifier: 0x4a53983e
  End-to-End Identifier: 0x8a2e3156
  [Request In: 255]
  [Response Time: 7.046161452 seconds]
  ▶ AVP: Session-Id(263) l=41 f=-M- val=icscf.open-ims.test;2552567970;18
  ▶ AVP: Vendor-Specific-Application-Id(260) l=32 f=-M-
  ▶ AVP: Result-Code(268) l=12 f=M- val=DIAMETER_SUCCESS (2001)
  ▶ AVP: Server-Name(602) l=35 f=VM- vnd=TGPP val=sip:scscf.open-ims.test

```

Figure 8.4: HSS FE response sent to the I-CSCF.

## Cx-MAR

(Step<sub>1</sub>) The I-CSCF forwards the SIP registration message to the S-CSCF.

(Step<sub>2</sub>) The S-CSCF sends a Cx-MAR message to the HSS FE.

```

▼ Diameter Protocol
  Version: 0x01
  Length: 336
  ▶ Flags: 0xc0, Request, Proxyable
  Command Code: 303 Multimedia-Auth
  ApplicationId: 3GPP Cx (16777216)
  Hop-by-Hop Identifier: 0x66373596
  End-to-End Identifier: 0x8a45a1d0
  [Answer In: 27702]
  ▶ AVP: Session-Id(263) l=40 f=-M- val=scscf.open-ims.test;896604324;16
  ▶ AVP: Origin-Host(264) l=27 f=-M- val=scscf.open-ims.test
  ▶ AVP: Origin-Realm(296) l=21 f=-M- val=open-ims.test
  ▶ AVP: Destination-Realm(283) l=21 f=-M- val=open-ims.test
  ▶ AVP: Vendor-Specific-Application-Id(260) l=32 f=-M-
  ▶ AVP: Auth-Session-State(277) l=12 f=-M- val=NO_STATE_MAINTAINED (1)
  ▶ AVP: Public-Identity(601) l=34 f=VM- vnd=TGPP val=sip:mike@open-ims.test
  ▶ AVP: User-Name(1) l=26 f=-M- val=mike@open-ims.test
  ▶ AVP: 3GPP-SIP-Number-Auth-Items(607) l=16 f=VM- vnd=TGPP val=1
  ▶ AVP: 3GPP-SIP-Auth-Data-Item(612) l=36 f=VM- vnd=TGPP
  ▶ AVP: Server-Name(602) l=40 f=VM- vnd=TGPP val=sip:scscf.open-ims.test;6060
  ▶ AVP Code: 602 Server-Name
  ▶ AVP Flags: 0xc0
  ▶ AVP Length: 40
  ▶ AVP Vendor Id: 3GPP (10415)
  ▶ Server-Name: sip:scscf.open-ims.test;6060

```

Figure 8.5: S-CSCF MAR request sent to HSS FE.

(Step<sub>3</sub>) The HSS FE queries the CSDR for subscriber data and gets a response.

(Step<sub>4</sub>) The HSS FE sends a MAR response message to the S-CSCF

```

▼ Diameter Protocol
  Version: 0x01
  Length: 284
  ▶ Flags: 0x40, Proxyable
  Command Code: 303 Multimedia-Auth
  ApplicationId: 3GPP Cx (16777216)
  Hop-by-Hop Identifier: 0x66373596
  End-to-End Identifier: 0x8a45a1d0
  [Request In: 27602]
  [Response Time: 0.068612955 seconds]
  ▶ AVP: Session-Id(263) l=40 f=-M- val=scscf.open-ims.test;896604324;16
  ▶ AVP: Vendor-Specific-Application-Id(260) l=32 f=-M-
  ▶ AVP: Result-Code(268) l=12 f=-M- val=DIAMETER_SUCCESS (2001)
  ▶ AVP: 3GPP-INSI(1) l=30 f=VM- vnd=TGPP val=mike@open-ims.test
  ▶ AVP: Public-Identity(601) l=34 f=VM- vnd=TGPP val=sip:mike@open-ims.test
  ▶ AVP: 3GPP-SIP-Auth-Data-Item(612) l=96 f=VM- vnd=TGPP
  ▶ AVP: 3GPP-SIP-Number-Auth-Items(607) l=16 f=VM- vnd=TGPP val=1

```

Figure 8.6: S-CSCF MAR request sent to HSS FE.



## Cx-SAR

(Step<sub>1</sub>) The S-CSCF sends a Cx-SAR message to the HSS FE.

```

▼ Diameter Protocol
  Version: 0x01
  Length: 316
  ▶ Flags: 0xc0, Request, Proxyable
  Command Code: 301 Server-Assignment
  ApplicationId: 3GPP Cx (16777216)
  Hop-by-Hop Identifier: 0x66373597
  End-to-End Identifier: 0x8a45a1d1
  [Answer In: 28338]
  ▶ AVP: Session-Id(263) l=40 f=-M- val=scscf.open-ims.test;896604324;17
  ▶ AVP: Origin-Host(264) l=27 f=-M- val=scscf.open-ims.test
  ▶ AVP: Origin-Realm(296) l=21 f=-M- val=open-ims.test
  ▶ AVP: Destination-Realm(283) l=21 f=-M- val=open-ims.test
  ▶ AVP: Vendor-Specific-Application-Id(260) l=32 f=-M-
  ▶ AVP: Auth-Session-State(277) l=12 f=-M- val=NO_STATE_MAINTAINED (1)
  ▶ AVP: Public-Identity(601) l=34 f=VM- vnd=TGPP val=sip:mike@open-ims.test
  ▶ AVP: Server-Name(602) l=40 f=VM- vnd=TGPP val=sip:scscf.open-ims.test:6060
  ▶ AVP: User-Name(1) l=26 f=-M- val=mike@open-ims.test
  ▼ AVP: Server-Assignment-Type(614) l=16 f=VM- vnd=TGPP val=RE_REGISTRATION (2)
    AVP Code: 614 Server-Assignment-Type
    ▶ AVP Flags: 0xc0
    AVP Length: 16
    AVP Vendor Id: 3GPP (10415)
    Server-Assignment-Type: RE_REGISTRATION (2)
  ▼ AVP: User-Data-Already-Available(624) l=16 f=VM- vnd=TGPP val=USER_DATA_ALREADY_AVAILABLE (1)
    AVP Code: 624 User-Data-Already-Available
    ▶ AVP Flags: 0xc0
    AVP Length: 16
    AVP Vendor Id: 3GPP (10415)
    User-Data-Already-Available: USER_DATA_ALREADY_AVAILABLE (1)

```

Figure 8.7: S-CSCF sends Cx-SAR message to the HSS FE.

(Step<sub>2</sub>) The HSS FE queries the CSDR for subscription profile and the CSDR responds.

(Step<sub>3</sub>) The HSS FE sends a SAR response to the S-CSCF.

```

▼ Diameter Protocol
  Version: 0x01
  Length: 1932
  ▶ Flags: 0x40, Proxyable
  Command Code: 301 Server-Assignment
  ApplicationId: 3GPP Cx (16777216)
  Hop-by-Hop Identifier: 0x66373597
  End-to-End Identifier: 0x8a45a1d1
  [Request In: 27978]
  [Response Time: 0.619009190 seconds]
  ▶ AVP: Session-Id(263) l=40 f=-M- val=scscf.open-ims.test;896604324;17
  ▶ AVP: Vendor-Specific-Application-Id(260) l=32 f=-M-
  ▶ AVP: Result-Code(268) l=12 f=-M- val=DIAMETER_SUCCESS (2001)
  ▶ AVP: 3GPP-IMS(1) l=30 f=VM- vnd=TGPP val=mike@open-ims.test
  ▼ AVP: Cx-User-Data(606) l=1795 f=VM- vnd=TGPP val=3c3f786d6c2076657273696f6e3d22312e302220656e636f...
    AVP Code: 606 Cx-User-Data
    ▶ AVP Flags: 0xc0
    AVP Length: 1795
    AVP Vendor Id: 3GPP (10415)
    Cx-User-Data: 3c3f786d6c2076657273696f6e3d22312e302220656e636f...
    ▼ eXtensible Markup Language
      ▶ <?xml
      ▼ <IMSSubscription>
        ▼ <PrivateID>
          mike@open-ims.test
        </PrivateID>
        ▼ <ServiceProfile>
          ▼ <PublicIdentity>
            ▼ <BarringIndication>
              false
            </BarringIndication>
            ▼ <Identity>
              sip:mike@open-ims.test
            </Identity>
          </PublicIdentity>
          ▼ <PublicIdentity>
            ▼ <BarringIndication>
              false
            </BarringIndication>
            ▼ <Identity>
              sip:mike2@open-ims.test
            </Identity>
          </PublicIdentity>
          ▼ <InitialFilterCriteria>
            ▼ <Priority>
              0
            </Priority>
          </InitialFilterCriteria>
        </ServiceProfile>
      </IMSSubscription>
    </Cx-User-Data>
  </AVP>

```

Figure 8.8: HSS FE sends SAR response to the S-CSCF.

(Step<sub>4</sub>) The RCS client is notified.

## Cx-LIR

(Step<sub>1</sub>) The I-CSCF sends a Cx-LIR message to the HSS FE.

(Step<sub>2</sub>) The HSS FE fetches the S-CSCF data for registered subscriber from the CSDR and sends it to the I-CSCF.

```

▼ Diameter Protocol
  Version: 0x01
  Length: 144
  ▶ Flags: 0x40, Proxyable
  Command Code: 302 Location-Info
  ApplicationId: 3GPP Cx (16777216)
  Hop-by-Hop Identifier: 0x4a539844
  End-to-End Identifier: 0x8a2e315c
  [Request In: 1846]
  [Response Time: 0.168176543 seconds]
  ▶ AVP: Session-Id(263) l=41 f=M- val=icscf.open-ims.test;2552567970;24
  ▶ AVP: Vendor-Specific-Application-Id(260) l=32 f=M-
  ▶ AVP: Result-Code(268) l=12 f=M- val=DIAMETER_SUCCESS (2001)
  ▼ AVP: Server-Name(602) l=35 f=VM- vnd=TGPP val=sip:scscf.open-ims.test
    AVP Code: 602 Server-Name
    ▶ AVP Flags: 0xc0
    AVP Length: 35
    AVP Vendor Id: 3GPP (10415)
    Server-Name: sip:scscf.open-ims.test
    Padding: 00

```

Figure 8.9: HSS FE LIR response sent to the I-CSCF.

## Scenario 2 — Querying the S-CSCF Association of an RCS Client

A messaging service wants to know about the status and S-CSCF name designated to a subscriber identity in the HSS. It composes a UDR message with Diameter Sh codes for *S-CSCFName* and *IMSUserStatus* data. Diameter Sh codes can be found in Appendix A.5.1.

(Step<sub>1</sub>) The service sends an Sh-UDR message to the HSS FE.

```

▼ Diameter Protocol
  Version: 0x01
  Length: 384
  ▶ Flags: 0x80, Request
  Command Code: 306 User-Data
  ApplicationId: 3GPP Sh (16777217)
  Hop-by-Hop Identifier: 0x76e2c285
  End-to-End Identifier: 0x06390002
  [Answer In: 4361]
  ▶ AVP: Session-Id(263) l=55 f=M- val=BadCustomSessionId;YesWeCanPassId;1475168360015
  ▶ AVP: Vendor-Specific-Application-Id(260) l=32 f=M-
  ▶ AVP: Destination-Realm(283) l=21 f=M- val=open-ims.test
  ▶ AVP: Destination-Host(293) l=20 f=M- val=192.168.23.4
  ▶ AVP: Origin-Host(264) l=17 f=M- val=127.0.0.1
  ▶ AVP: Origin-Realm(296) l=21 f=M- val=open-ims.test
  ▶ AVP: Data-Reference(703) l=16 f=VM- vnd=TGPP val=RepositoryData (0)
  ▶ AVP: Origin-Host(264) l=23 f=M- val=aaa://127.0.0.1
  ▶ AVP: Server-Name(602) l=44 f=VM- vnd=TGPP val=sip:messaging@open-ims.test;5080
  ▶ AVP: Data-Reference(703) l=16 f=VM- vnd=TGPP val=IMSUserState (11)
  ▶ AVP: Data-Reference(703) l=16 f=VM- vnd=TGPP val=S-CSCFName (12)
  ▶ AVP: Public-Identity(601) l=34 f=VM- vnd=TGPP val=sip:mike@open-ims.test
  ▶ AVP: Auth-Session-State(277) l=12 f=M- val=Unknown (2)
  ▶ AVP: Origin-Realm(296) l=21 f=M- val=open-ims.test

```

Figure 8.10: UDR message received by HSS FE.

(Step<sub>2</sub>) HSS FE queries the CSDR.

(Step<sub>3</sub>) CSDR responds with the pertinent data.

(Step<sub>4</sub>) HSS FE sends a response to the messaging service

```

▼ Diameter Protocol
  Version: 0x01
  Length: 340
  ▶ Flags: 0x00
  Command Code: 306 User-Data
  ApplicationId: 3GPP Sh (16777217)
  Hop-by-Hop Identifier: 0x76e2c285
  End-to-End Identifier: 0x06300002
  [Request In: 1289]
  [Response Time: 0.217818981 seconds]
  ▶ AVP: Session-Id(263) l=55 f=-M- val=BadCustomSessionId;YesWeCanPassId;1475168360015
  ▶ AVP: Vendor-Specific-Application-Id(260) l=32 f=-M-
  ▶ AVP: Result-Code(268) l=12 f=-M- val=DIAMETER_SUCCESS (2001)
  ▼ AVP: Sh-User-Data(702) l=220 f=VM- vnd=3GPP val=3c3f786d6c2076657273696f6e3d22312e302220656e636f...
    AVP Code: 702 Sh-User-Data
    ▶ AVP Flags: 0xc0
    AVP Length: 220
    AVP Vendor Id: 3GPP (10415)
    Sh-User-Data: 3c3f786d6c2076657273696f6e3d22312e302220656e636f...
  ▼ eXtensible Markup Language
    ▶ <?xml
    ▼ <Sh-Data>
      ▼ <Sh-IMS-Data>
        ▼ <SCSCFName>
          sip:scscf@open-ims.test
        </SCSCFName>
        ▼ <IMSUserState>
          1
        </IMSUserState>
      </Sh-IMS-Data>
    </Sh-Data>

```

Figure 8.11: HSS responds to the messaging service.

## 8.3 Telephony and Messaging FEs

As discussed in Section 2.7, the messaging and telephony services can handle chat, SMS and VVoIP conversations within RCS. In addition, the file transfer service works closely with messaging service. According to the RCS specification [52], a messaging AS can provide both services and outsource the management of files to another repository. This is possible because both services use MSRP for information exchange over the network and can be required when a CFS is not deployed by an MNO. Thus, the telephony and messaging FEs were created to coordinate and generate data from RCS messaging, file transfer and VVoIP calls placed over IMS.

### 8.3.1 Description

IMSDroid and Boghe IMS client were used to generate conversations between subscribers. Both FEs (Telephony and Messaging) generate similar sets of data: call and message log, and session-related data. These data sets are represented as *CallDetailRecord* and *ClientSessionData* entities within the domain model presented in Figure 7.2. The CSDR stores these data sets in MySQL and Redis respectively. The services were deployed on a bundling of Restcomm SIP servlets 4.0.95 API and Wildfly 10 AS technology. The platform allows for the deployment of portable and distributable SIP and Java-based services [90]. Multiple SIP servlets can be deployed on this platform. Thus, Restcomm uses a Default Application Router (DAR) to dispatch incoming SIP requests to the appropriate SIP servlet deployed on the AS. The DAR uses configurations for both FEs, which are stored different `mobicents-dar.properties` file.

### 8.3.2 Implementation

For simplicity, the telephony service was configured to explicitly handle SIP INVITE messages which can be used for chat, file transfer, and VVoIP. Contrarily, the messaging FE was configured to explicitly handle SIP MESSAGE requests to mimic the behaviour of a Standalone Message Service. Using this approach, SIP INVITE requests were used to generate data for real-time communications while SIP MESSAGE requests were used to generate RCS standalone message log and session data. Furthermore, both FEs were provisioned on the IMS as enablers for SIP services that are used by registered RCS subscribers. Therefore, the address of these FEs can be found in the iFC portion of subscription profiles that the S-CSCF downloads from the HSS FE.

The Telephony FE `mobicents-dar.properties` file contains the following:

```
INVITE: ("com.inted.as.telephony.Main", "DAR:From", "
    TERMINATING", "", "NO_ROUTE", "0")
```

The Messaging FE `mobicents-dar.properties` file contains the following:

```
MESSAGE: ("com.inted.as.messaging.Main", "DAR:From", "
    TERMINATING", "", "NO_ROUTE", "0")
```

### 8.3.3 Results

#### Scenario 1

Subscriber Mike places a voice call to Tessa, who is also registered on the network.

(*Step<sub>1</sub>*) Mike's client sends a SIP INVITE request toward the network.

(*Step<sub>2</sub>*) The network routes the request to Tessa.

(*Step<sub>3</sub>*) Tessa accepts the call.

```

▼ Session Initiation Protocol (200)
  ▶ Status-Line: SIP/2.0 200 OK
  ▼ Message Header
    ▶ Via: SIP/2.0/UDP 192.168.23.4:4060;branch=z9hG4bK59ea.069c6d16.0
    ▼ Via: SIP/2.0/UDP 192.168.23.2:54220;branch=z9hG4bK824085621;rport=54220
      Transport: UDP
      Sent-by Address: 192.168.23.2
      Sent-by port: 54220
      Branch: z9hG4bK824085621
      RPort: 54220
    ▶ From: <sip:mike@open-ims.test>;tag=626677066
    ▶ To: <sip:tessa@open-ims.test>;tag=475777425
    ▶ Contact: <sip:tessa@192.168.23.1:53539;transport=udp>
    Call-ID: ddbf0562-4513-bebb-6426-8db845cfd25
    ▼ CSeq: 320161950 PRACK
      Sequence Number: 320161950
      Method: PRACK
      Content-Length: 0
```

Figure 8.12: Tessa accepts the voice call.

(Step<sub>4</sub>) The Telephony FE updates call log in the CSDR with Mike’s most recent conversation.

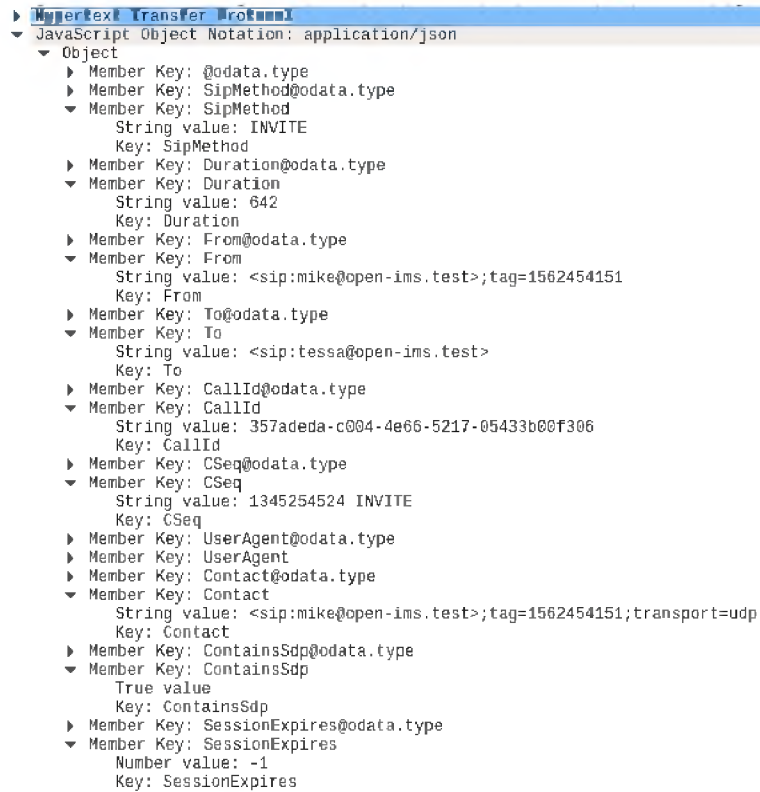


Figure 8.13: Telephony FE adds Mike’s recent conversation to the CSDR call log.

(Step<sub>5</sub>) The Telephony FE updates Mike’s session data in the CSDR.

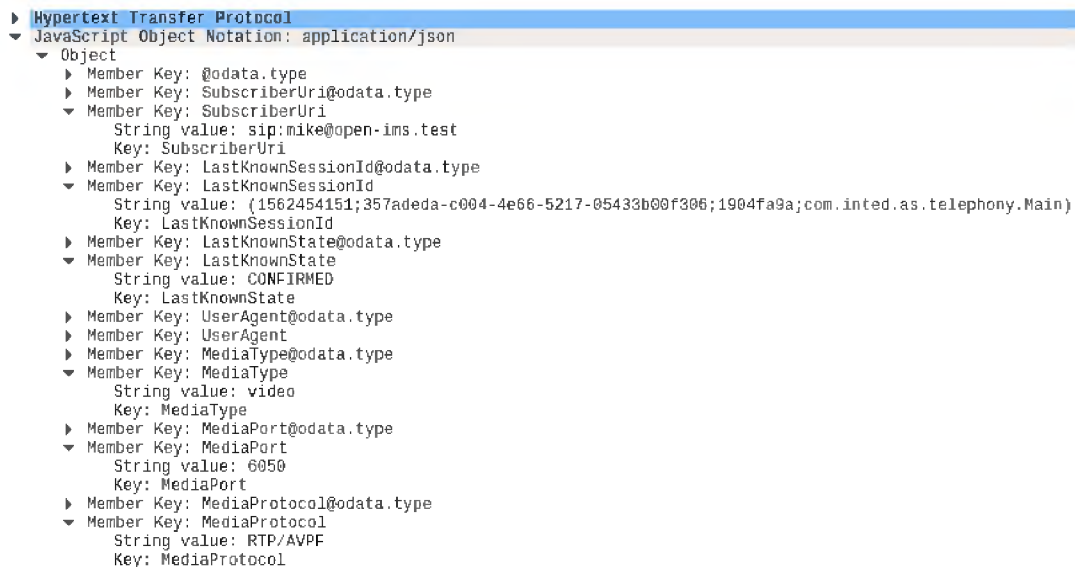


Figure 8.14: Telephony FE updates Mike’s session data in the CSDR.

## Scenario 2

Tessa sends an SMS to Mike.

(Step<sub>1</sub>) Tessa's client forwards the message to the network.

```

▼ Session Initiation Protocol (MESSAGE)
  ▶ Request-Line: MESSAGE sip:mike@open-ims.test SIP/2.0
  ▼ Message Header
    ▶ Route: <sip:messaging@open-ims.test:5080;lr>, <sip:ismark@scscf.open-ims.test:6060;lr;s=2;h=-1;d=1;a=7369763a6d696b655406f79656e2d696d732e74657374>
    ▶ Via: SIP/2.0/UDP 192.168.23.4:6060;branch=z9hG4bKad6f.7963df22.0
    ▶ Via: SIP/2.0/UDP 192.168.23.4:6060;branch=z9hG4bKad6f.6963df22.1
    ▶ Via: SIP/2.0/UDP 192.168.23.4:4060;branch=z9hG4bKad6f.4f9a24f2.0
    ▶ Via: SIP/2.0/UDP 192.168.23.1:53539;branch=z9hG4bK-476687359;rport=53539
    ▶ From: <sip:tessa@open-ims.test>;tag=476677711
    ▶ To: <sip:mike@open-ims.test>
    ▶ Call-ID: eba9hebb-11c3-1342-a1b5-4006724eba8
    ▼ CSeq: 4837 MESSAGE
      ▶ Sequence Number: 4837
      ▶ Method: MESSAGE
      ▶ Content-Length: 19
      ▶ Max-Forwards: 14
      ▶ Accept-Contact: ";g.oma.sip-im"
      ▶ Accept-Contact: ";language=en,fr"
      ▶ Content-Type: text/plain
      ▶ Allow: INVITE, ACK, CANCEL, BYE, MESSAGE, OPTIONS, NOTIFY, PRACK, UPDATE, REFER
      ▶ Privacy: none
      ▶ P-Access-Network-Info: ADSL;utran-cell-id-3gpp=00000000
      ▶ User-Agent: IM-client/OMA1.0 Bogue-Win32/v2.0.106.1013
      ▶ P-Asserted-Identity: <sip:tessa@open-ims.test>
      ▶ P-Charging-Vector: icid-value="P-CSCFabc000000057ee098a0000046";icid-generated-at=192.168.23.4;orig-icid="open-ims.test"
  ▼ Message Body
    ▼ Line-based text data: text/plain
      Hi Michael
  
```

Figure 8.15: Tessa sends an SMS to Mike.

(Step<sub>2</sub>) The network routes the message to Mike.

(Step<sub>3</sub>) The Messaging FE updates Tessa's session data.

```

▼ Hypertext Transfer Protocol
  ▼ JavaScript Object Notation: application/json
    ▼ Object
      ▶ Member Key: @odata.type
      ▶ Member Key: SubscriberUri@odata.type
      ▼ Member Key: SubscriberUri
        String value: sip:tessa@open-ims.test
        Key: SubscriberUri
      ▶ Member Key: LastKnownSessionId@odata.type
      ▶ Member Key: LastKnownSessionId
      ▶ Member Key: LastKnownState@odata.type
      ▶ Member Key: LastKnownState
      ▶ Member Key: UserAgent@odata.type
      ▶ Member Key: UserAgent
      ▶ Member Key: MediaType@odata.type
      ▼ Member Key: MediaType
        String value: message
        Key: MediaType
      ▶ Member Key: MediaPort@odata.type
      ▼ Member Key: MediaPort
        String value: 53727
        Key: MediaPort
      ▶ Member Key: MediaProtocol@odata.type
      ▼ Member Key: MediaProtocol
        String value: TCP/MSRP
        Key: MediaProtocol
    
```

Figure 8.16: Messaging FE updates Tessa's session data in the CSDR.

## Scenario 3

Tessa sends a file to Mike.

(Step<sub>1</sub>) Tessa's client forwards the file transfer request as a SIP INVITE to the network.

```

▼ Session Initiation Protocol (INVITE)
  ▶ Request-Line: INVITE sip:mike@open-ims.test SIP/2.0
  ▶ Message Header
  ▼ Message Body
    ▼ Session Description Protocol
      Session Description Protocol Version (v): 0
      ▶ Owner/Creator, Session Id (o): doubango 1983 678901 IN IP4 192.168.23.1
      Session Name (s): -
      ▶ Connection Information (c): IN IP4 192.168.23.1
      ▶ Time Description, active time (t): 0 0
      ▶ Media Description, name and address (m): message 34478 TCP/MSRP *
      ▶ Connection Information (c): IN IP4 192.168.23.1
      ▶ Media Attribute (a): path:msrp://192.168.23.1:34478/476944138;tcp
      ▶ Media Attribute (a): connection:new
      ▶ Media Attribute (a): setup:actpass
      ▶ Media Attribute (a): accept-types:message/CPIM application/octet-stream
      ▶ Media Attribute (a): accept-wrapped-types:application/octet-stream image/jpeg image/gif image/bmp image/png
      ▶ Media Attribute (a): sendonly
      ▶ Media Attribute (a): file-selector:name:"20140823_092951.jpg" type:application/octet-stream size:1168250
      ▶ Media Attribute (a): file-transfer-id:476968686
      ▶ Media Attribute (a): file-disposition:attachment
      ▶ Media Attribute (a): file-icon:cid:test@doubango.org

```

Figure 8.17: SIP File Transfer Request from Tessa.

(Step<sub>2</sub>) Mike accepts the request.

(Step<sub>3</sub>) The file is sent to Mike over the network.

(Step<sub>4</sub>) Telephony FE updates Tessa's session data in the CSDR.

```

▶ Hypertext Transfer Protocol
▼ JavaScript Object Notation: application/json
  ▼ Object
    ▶ Member Key: @odata.type
    ▶ Member Key: SubscriberUri@odata.type
    ▼ Member Key: SubscriberUri
      String value: sip:tessa@open-ims.test
      Key: SubscriberUri
    ▶ Member Key: LastKnownSessionId@odata.type
    ▼ Member Key: LastKnownSessionId
      String value: {476966147;bee87f25-cc4f-e145-bbhc-dc7c75bbf9a0;349ad82f;com.inted.as.telephony.Main}
      Key: LastKnownSessionId
    ▶ Member Key: LastKnownState@odata.type
    ▶ Member Key: LastKnownState
    ▶ Member Key: UserAgent@odata.type
    ▶ Member Key: UserAgent
    ▶ Member Key: MediaType@odata.type
    ▶ Member Key: MediaType
    ▶ Member Key: MediaPort@odata.type
    ▼ Member Key: MediaPort
      String value: 53727
      Key: MediaPort
    ▶ Member Key: MediaProtocol@odata.type
    ▼ Member Key: MediaProtocol
      String value: TCP/MSRP
      Key: MediaProtocol

```

Figure 8.18: Telephony FE updates Tessa's session data with the file transfer activity.

## 8.4 XCAP FE

XCAP, as previously discussed, is an HTTP-based protocol. As such, simple HTTP verbs can be used to execute CRUD operations on XCAP document. Operations supported by XCAP are similar to those defined in Section 4.5. However, the use of the XPath language allows the XDMS to navigate the contents of each XML document [92]. In other words, XPath allows clients to execute CRUD operations on entire documents or individual elements of it.

### 8.4.1 Description

The XCAP FE is designed to mimic a simplistic behaviour of an XDMS and it uses the basic XPath URI construction rule for handling an XML document. This follows the work done in [59]. Choosing this approach eliminates the possibility of executing CRUD operations on the contents of the XCAP document since the FE is used for testing the CSDR. Therefore, it supports two basic HTTP operations: creating and fetching XML documents that are stored in the CSDR. To create an XML document in the XCAP FE, the HTTP URI is constructed by the client as follows:

```
PUT <XCAP-Root>/auid/sub-tree/xuid/document-name
```

To fetch an XCAP document, the HTTP URI is as follows:

```
GET <XCAP-Root>/auid/sub-tree/xuid/document-name
```

- **<XCAP-Root>** denotes the address of the XCAP FE.
- **auid** denotes the identity of the AppUsage bound to the XML document.
- **sub-tree** denotes the scope of the document which can be either “global” or “users”.
- **xuid** denotes the identity of the owner of the document. That is, a subscriber’s IMS URI.
- **document-name** is used to identify the document being managed.

### 8.4.2 Implementation

The FE also embeds the Jetty AS as a standalone application. Postman was used as an XCAP client for this experiment. The XCAP FE supports a ‘presence-lists’ AppUsage. This list allows a subscriber to manage SPI subscriptions of their contacts through a single XML document. The presence-lists AppUsage defines the following Multipurpose Internet Mail Extensions (MIME) type: ‘application/rls-services+xml’. This value is specified by the XCAP client when sending a request to the FE. More importantly, the XCAP FE demonstrates how XML documents can be stored and retrieved from the CSDR.

### 8.4.3 Results

#### Scenario 1

An XCAP client creates a buddy list in the XCAP FE.



(Step<sub>1</sub>) The client sends a PUT request to the XCAP FE.

```
[Full request URI: http://localhost:8282/xcap-server-fe/resource-lists/users/sip:mike@open-ims.test/buddy-list.xml]
[HTTP request 1/2]
[Response in frame: 80385]
[Next request in frame: 89246]
File Data: 984 bytes
▼ Extensible Markup Language
  ▶ <?xml
  ▼ <resource-lists
    xmlns="urn:ietf:params:xml:ns:resource-lists"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      ▼ <list
        name="friends">
          ▶ <entry
          ▶ <entry-ref
          ▼ <list
            name="close-friends">
              ▶ <display-name>
              ▼ <entry
                uri="sip:tessa@open-ims.test">
                  ▶ <display-name>
                  </entry>
              ▼ <entry
                uri="sip:mary@open-ims.test">
                  ▶ <display-name>
                  </entry>
              ▼ <entry
                uri="sip:alice@open-ims.test">
                  ▶ <display-name>
                  </entry>
              ▼ <entry
                uri="sip:bob@open-ims.test">
                  ▶ <display-name>
                  </entry>
            </list>
          </list>
        </resource-lists>
```

Figure 8.19: Creating an XCAP document.

(Step<sub>2</sub>) The XCAP FE parses the URL and extracts the components.

(Step<sub>3</sub>) The XCAP FE sends a POST request to the CSDR.

```
▼ Hypertext Transfer Protocol
  ▼ POST /csdr/svc/XcapDocuments HTTP/1.1\r\n
    ▶ [Expert Info (Chat/Sequence): POST /csdr/svc/XcapDocuments HTTP/1.1\r\n]
    Request Method: POST
    Request URI: /csdr/svc/XcapDocuments
    Request Version: HTTP/1.1
    Accept: application/json;odata.metadata=minimal\r\n
    Content-Type: application/json;odata.metadata=full\r\n
    OData-MaxVersion: 4.0\r\n
    OData-Version: 4.0\r\n
    Transfer-Encoding: chunked\r\n
    Host: localhost:8080\r\n
    Connection: Keep-Alive\r\n
    User-Agent: Apache-Clingo/4.0.0-beta-02\r\n
    \r\n
    [Full request URI: http://localhost:8080/csdr/svc/XcapDocuments]
    [HTTP request 1/1]
    [Response in frame: 80295]
    ▶ HTTP chunked response
    File Data: 1341 bytes
  ▼ JavaScript Object Notation: application/json
    ▼ Object
      ▼ Member Key: @odata.type
        String value: #com.inted.csdr.XcapDocument
        Key: @odata.type
      ▶ Member Key: Auid@odata.type
      ▶ Member Key: Auid
      ▶ Member Key: Xui@odata.type
      ▶ Member Key: Xui
      ▶ Member Key: DocumentName@odata.type
      ▶ Member Key: DocumentName
      ▶ Member Key: Data@odata.type
      ▶ Member Key: Data
      ▶ Member Key: isUpdated@odata.type
      ▶ Member Key: isUpdated
```

Figure 8.20: Creating an XCAP document in CSDR.

(Step<sub>4</sub>) The CSDR creates the document and forwards a 201 Created response.

(Step<sub>5</sub>) The XCAP FE forwards the response to the client.

## Scenario 2

An XCAP client fetches an XML document from the XCAP FE.

(Step<sub>1</sub>) The client sends a GET request to the XCAP FE.

```

▼ Hypertext Transfer Protocol
  ▼ GET /xcap-server-fe/resource-lists/users/sip:mike@open-ims.test/buddy-list.xml HTTP/1.1\r\n
    ▶ [Expert Info (Chat/Sequence): GET /xcap-server-fe/resource-lists/users/sip:mike@open-ims.test/buddy-list.xml HTTP/1.1\r\n]
      Request Method: GET
      Request URI: /xcap-server-fe/resource-lists/users/sip:mike@open-ims.test/buddy-list.xml
      Request Version: HTTP/1.1
      Host: localhost:8282\r\n
      Connection: keep-alive\r\n
      Postman-Token: 33a3de99-85ae-9362-a3e6-f798b8cb5f28\r\n
      Cache-Control: no-cache\r\n
      User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/53.0.2785.116 Safari/537.36\r\n
      Content-Type: application/rls-services+xml\r\n
      Accept: */*\r\n
      Accept-Encoding: gzip, deflate, sdch\r\n
      Accept-Language: en-US,en;q=0.8\r\n
      \r\n
      [Full request URI: http://localhost:8282/xcap-server-fe/resource-lists/users/sip:mike@open-ims.test/buddy-list.xml]
      [HTTP request 2/2]
      [Prev request in frame: 77058]
      [Response in frame: 89289]
  
```

Figure 8.21: Fetching an XCAP document.

(Step<sub>2</sub>) The XCAP FE parses the request and queries the CSDR for the XML document.

(Step<sub>3</sub>) The CSDR fetches the XCAP document and forwards it to the XCAP FE.

(Step<sub>4</sub>) The XCAP FE sends the document to the client.

```

▼ Hypertext Transfer Protocol
  ▼ HTTP/1.1 200 OK\r\n
    ▶ [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
      Request Version: HTTP/1.1
      Status Code: 200
      Response Phrase: OK
      Date: Sat, 01 Oct 2016 07:00:24 GMT\r\n
      Content-Type: application/rls-services+xml; charset=iso-8859-1\r\n
      Content-Length: 984\r\n
      Server: Jetty(9.3.9.M0)\r\n
      \r\n
      [HTTP response 2/2]
      [Time since request: 2.118280865 seconds]
      [Prev request in frame: 77058]
      [Prev response in frame: 80305]
      [Request in frame: 89246]
      File Data: 984 bytes
  ▼ eXtensible Markup Language
    ▶ <?xml
    ▼ <resource-lists
      xmlns="urn:ietf:params:xml:ns:resource-lists"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        ▼ <list
          name="friends">
            ▶ <entry
              ▼ <entry-ref
                ref="resource-lists/users/sip:mike@open-ims.test/">
                  ▼ <list
                    name="close-friends">
                      ▶ <display-name>
                        ▶ <entry
                          ▶ <entry
                          ▶ <entry
                        </list>
                      </list>
                    </list>
                  </list>
                </entry-ref>
              </entry>
            </list>
          </resource-lists>
        </list>
      </resource-lists>
    </?xml>
  
```

Figure 8.22: Document sent to the XCAP client.

## 8.5 Engaging the CSDR

The consolidated data set can be interrogated through both a PFE and backend interface. Postman was used to interact directly with the CSDR in order to demonstrate the behaviour of a PFE toward the CSDR. This section presents three scenarios where the PFE interacts with the CSDR.

### Scenario 1

A PFE fetches the entity set for subscription profiles stored in the CSDR and specifies the attributes to be displayed using the \$select query option.

(*Message<sub>1</sub>*) Request

```
GET /csdr/svc/SubscriptionProfiles?$select=ID,Name,Impi,ScscfName HTTP/1.1
Host: localhost:8080
Cache-Control: no-cache
Postman-Token: a490e027-df1e-1aee-0861-3aa367a6c709
```

(*Message<sub>2</sub>*) Response

```
{
  "@odata.context": "$metadata#SubscriptionProfiles(ID,Name,Impi,ScscfName)",
  "value": [
    {
      "ID": 1,
      "Name": "Mike",
      "Impi": "mike@open-ims.test",
      "ScscfName": "sip:scscf@open-ims.test"
    },
    {
      "ID": 2,
      "Name": "Tessa",
      "Impi": "tessa@open-ims.test",
      "ScscfName": "sip:scscf@open-ims.test"
    },
    {
      "ID": 3,
      "Name": "Alice",
      "Impi": "alice@open-ims.test",
      "ScscfName": "sip:scscf@open-ims.test"
    },
    {
      "ID": 4,
      "Name": "Mary",
      "Impi": "mary@open-ims.test",
      "ScscfName": "sip:scscf@open-ims.test"
    },
    {
      "ID": 5,
      "Name": "Bob",
      "Impi": "bob@open-ims.test",
      "ScscfName": "sip:scscf@open-ims.test"
    }
  ]
}
```

## Scenario 2

A PFE fetches the IMPUs, which belong to Tessa using the \$expand query option.

### (Message<sub>1</sub>) Request

```
GET /csdr/svc/SubscriptionProfiles(2)?$expand=Impus HTTP/1.1
Host: localhost:8080
Cache-Control: no-cache
Postman-Token: 86635f50-ea49-733b-c44b-5d1878b5e989
```

### (Message<sub>2</sub>) Response

```
{
  "@odata.context": "$metadata#SubscriptionProfiles/$entity",
  "ID": 2,
  "Sid": "57ec7a74e4b06d073b074a78",
  "Name": "Tessa",
  "Impi": "tessa@open-ims.test",
  "ScscfName": "sip:scscf@open-ims.test",
  "ChargingInfo": {
    "PrimaryCcf": null,
    "SecondaryCcf": null,
    "PrimaryEcfc": null,
    "SecondaryEcfc": null
  },
  "Dsai": "Inactive",
  "Impus": [
    {
      "ID": 1,
      "Sid": "57ec7a74e4b06d073b074a79",
      "SipUri": "sip:tessa@open-ims.test",
      "IdentityType": "PublicUserIdentity",
      "ImsUserState": "AuthenticationPending",
      "BarringIndication": false,
      "CanRegister": true,
      "DisplayName": "TESSA 1",
      "PsiActivation": "Inactive"
    },
    {
      "ID": 2,
      "Sid": "57ec7a74e4b06d073b074a7a",
      "SipUri": "sip:tessa2@open-ims.test",
      "IdentityType": "PublicUserIdentity",
      "ImsUserState": "AuthenticationPending",
      "BarringIndication": false,
      "CanRegister": true,
      "DisplayName": "TESSA 2",
      "PsiActivation": "Inactive"
    }
  ]
}
```

## Scenario 3

A PFE checks the CDR for all calls that have been placed by Tessa using the \$select, \$filter and \$contains query options.

### (Message<sub>1</sub>) Request

```
GET /csdr/svc/CallDetailRecords?$select=ID,CSeq,From,To,Contact,Duration&$filter=contains(From,'tessa') HTTP/1.1
Host: localhost:8080
Cache-Control: no-cache
Postman-Token: d6592901-6827-27b0-da6a-f0aaf1f2568e
```

### (Message<sub>2</sub>) Response

```
{
  "@odata.context": "$metadata#CallDetailRecords(ID,CSeq,To,From,Duration,Contact)",
  "value": [
    {
      "ID": 1,
      "CSeq": "339740171 INVITE",
      "To": "<sip:mike@open-ims.test>;tag=1510714595",
      "From": "<sip:tessa@open-ims.test>",
      "Duration": "621",
      "Contact": "<sip:tessa@192.168.28.1:60704;transport=udp>"
    },
    {
      "ID": 4,
      "CSeq": "28510 INVITE",
      "To": "<sip:mike@open-ims.test>",
      "From": "<sip:tessa@open-ims.test>;tag=385216151",
      "Duration": "132",
      "Contact": "<sip:tessa@open-ims.test>;tag=385216151;transport=udp"
    },
    {
      "ID": 9,
      "CSeq": "32409 INVITE",
      "To": "<sip:mike@open-ims.test>",
      "From": "<sip:tessa@open-ims.test>;tag=475273224",
      "Duration": "30",
      "Contact": "<sip:tessa@open-ims.test>;tag=475273224;transport=udp"
    },
    {
      "ID": 12,
      "CSeq": "18882 INVITE",
      "To": "<sip:mike@open-ims.test>",
      "From": "<sip:tessa@open-ims.test>;tag=476966147",
      "Duration": "69",
      "Contact": "<sip:tessa@open-ims.test>;tag=476966147;transport=udp"
    }
  ]
}
```

## 8.6 Summary

This chapter has shown the experiments that were carried out in a prototypical IMS testbed while using the CSDR as a unique repository within the network. The testbed and its components were created using purely open source tools. The experiments demonstrated the interactions between FE implementations for the HSS, XCAP, Telephony, and Messaging Services and the CSDR. Wireshark was used to analyse the traffic between the testbed components while the Doubango and IMSDroid clients and Postman REST client were used to generate SIP and HTTP traffic respectively. The CSDR exposed heterogeneous data sets through OData while persisting to MySQL, Redis and MongoDB at the underlying data access level. Thus, the chapter shows that different types of subscriber data can feature in the CSDR. Additionally, these activities have shown that the CSDR can support multiple FEs and also provide a converged view of data through a PFE. The next chapter concludes this thesis.

## Chapter 9

## Conclusion

This thesis has investigated the creation of a converged data repository for RCS services. It has summarised and analysed arguments from literature that have been produced by organisations such as GSMA, OMA, and 3GPP. A repository called CSDR was designed and integrated within a software-based, prototypical IMS network to manage heterogeneous RCS subscriber data sets. This chapter outlines the objectives that have been met, presents a summary of the main contributions and suggests several ideas for possible improvements on this work. The chapter starts out by discussing the realised objectives, which is then followed by the thesis contributions and suggestions for improvements in few areas of this work.

## 9.1 Achieved Objectives

The discussion in this section analyses the degree to which the research objectives stated in Section 1.4 have been met by the thesis.

### 9.1.1 Review of RCS specifications and its data management policies

The thesis has reviewed the GSMA and OMA specifications that RCS adopts. The literature revealed the efforts made by GSMA to drive RCS toward global adoption, which is impacted by two main factors: attractive service models and operator interoperability. Although ways in which MNOs can achieve interoperability have been defined for the RCS specifications, offering attractive services to subscribers can be enhanced by gaining insight into subscriber data. This was difficult to obtain with the way application and data services have been defined in the specifications while leveraging IMS. These specifications also showed how each component of the framework interoperates while being enabled by the distinct application and data services.

### 9.1.2 Review of the UDC standard to extract relevant requirements

The second objective was to investigate the main requirements of the UDC standard which are: a standardised architecture for a consolidated data repository and data manipulation requirements ranging from CRUD to subscribe/notify operations. Furthermore, 3GPP proposed the usage of LDAP and SOAP protocols as candidates for a unified data access interface toward the repository. The integration of both protocols raised concerns which lead to the contribution toward the Ud reference point as discussed in Section 9.2.

### 9.1.3 Investigation of data store design practices

The third objective was to investigate and document the different data service design and modelling techniques, which revealed some viable options which were considered when building the converged repository. The discussion highlighted a number of data models, schema and transaction management approaches, common persistence and fault tolerance techniques used by existing storage technologies. Additionally, using the CAP theorem when deciding among some of these factors. This lead to the adoption of a polyglot persistence model for managing heterogeneous data by leveraging the strengths of the integrated data stores as opposed to using a single data store.

### 9.1.4 Creating a Converged Repository

The fourth objective was to design and implement a prototype of the unified repository, which the author has named the Converged Subscriber Data Repository (CSDR). AFE and PFE interactions with the CSDR and the behaviours of the CSDR when managing CRUD and subscribe/notify requests were also defined. The repository adopts a polyglot persistence architecture for heterogeneous subscriber data sets. These datasets can be accessed through the OData protocol. Thus, the CSDR provides flexible schema management across underlying data stores through OData.

### 9.1.5 Evaluation of the design

The last objective of the thesis was to prove that the repository design could cater for heterogeneous data by integrating the prototype within an existing IMS network testbed. Multiple FE applications were developed to interact with the unified repository. Thus, allowing distinct FEs to query the repository for heterogeneous data within the network. Furthermore, different types of FEs and protocols were used to interact with FE implementations thus demonstrating that the existing network service interfaces can still be maintained with their corresponding protocols when a storage facility such as the CSDR is introduced.

## 9.2 Thesis Contributions

RCS provides a platform that enables inter-operator support, unifies client applications, centrally manages subscriber messaging data and exposes a subscriber's presence information through a unified phone book. This unification in communication lacked a central repository to manage data sets for services beyond messaging, thus, making it difficult to obtain a unified view of subscriber data in the network. The CSDR has been designed and implemented as a "centralised store" for RCS subscriber data within an IMS network. The use of this repository can enhance RCS services being offered by MNOs, as new service models can be derived from insights gained from the consolidation of subscriber data within the network. For instance, network services and functional entities can query the repository for relevant user data. Hence, the CSDR provides a structured and standardised approach to the management for heterogeneous RCS data in the network.

The Ud reference point provides access to the converged repository as defined by the UDC standard. 3GPP recommended the combination of LDAP and SOAP protocols for this reference point. LDAP was primarily designated for CRUD operations and SOAP for subscribe/notify operations. Adopting OData as the single protocol for the reference point allows a converged data repository to leverage the CRUD and callback mechanisms offered by the protocol. Seeing that



OData provides callback capability, it can provide the functionalities required of SOAP. Hence, OData supports both categories of operations, and this makes it a strong alternative to its counterparts. In consequence, the need to use multiple protocols over the Ud reference point is eliminated as OData provides a single-protocol alternative. Furthermore, as MNOs are embracing RESTful APIs, OData provides a way to expose data from different sources across the network using a standardised RESTful interface. In essence, disparate network data sources can expose their data using OData as a common communications protocol. Thus, the work related in this thesis provides a viable alternative to the LDAP and SOAP protocols for the realisation of the Ud interface.

Polyglot persistence provides a data storage architecture where distinct data sets are persisted in suitable storage technologies. Hence, data services can adopt multiple underlying stores for persistence. However, the complexity of managing these data sets arises when the services try to define a generic schema for encapsulating data retrieved from underlying stores. OData provides the EDM, which acts as a wrapper over existing data storage technologies. Since the OData is data-store-agnostic, it can expose data sets used by an underlying data store through the flexible and consistent EDM. Furthermore, as models used by the underlying stores change, a data service may need to redefine its schema. This becomes complex to manage data stored in each store within the service as schemas evolve. Nevertheless, with the CSDR exposing polyglot-persisted data through OData, this thesis has shown that a service schema built with EDM can provide a consistent view of data residing in underlying storage technologies. Thus, schema evolution of complex data services that support polyglot persistence can be managed by providing a conceptual view of these data sets through OData.

### 9.3 Future Work

Further work on the data management approach presented in this thesis could involve achieving polyglot persistence through data service convergence. In other words, defining a data service that comprises other data services. Currently, the PPI specified in the design expects connections to different data stores as demonstrated with the prototype's adoption of JPA. For CSDR implementations that do not adopt JPA, PPI algorithms and interfaces have to be devised to facilitate polyglot persistence. However, the introduction of an OData service convergence framework can enable the CSDR to integrate with underlying data stores as OData producers. This is illustrated in Figure 9.1. This will eliminate the need for PPI interface frameworks such as JPA, which are technology-based. Consequently, the CSDR can consist of a number of individual data services unified through a single EDM. This can have implications for complex data management decisions with existing and future network infrastructures.

As use-cases for the CSDR continue to evolve based on RCS services, more sophisticated techniques may be required to effectively manage data through the OData protocol. This may be achieved by introducing extensions in the form of

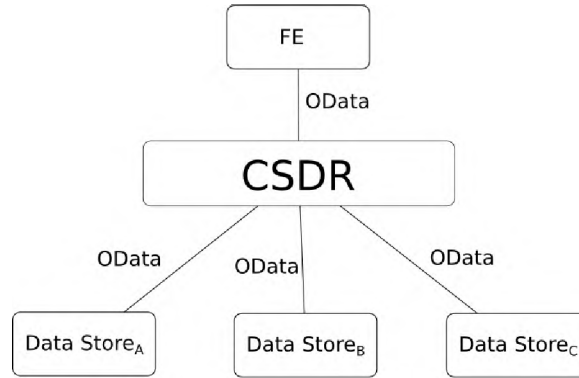


Figure 9.1: OData Service Convergence.

query and aggregation extensions to the protocol. In essence, proposing frameworks that give the CSDR the ability to query and aggregate data from external systems before providing a converged view. Say an RCS client was integrated with an ICSP application. Some data generated while using the services can be aggregated and stored by the CSDR as required. This can increase the depth of insight gained through the CSDR.

Furthermore, Kundera provides a JPA ONM framework for Java-based services. The PPI implementation with JPA guarantees that relationship between parent-child data sets can be managed in a consistent manner. However, systems that do not adopt JPA as the PPI for the CSDR, require synchronisation techniques to ensure data consistency across multiple stores. For example, a portion of subscriber data which was persisted across multiple stores should be reliably deleted across those stores when required. Thus, synchronisation mechanisms are required if the CSDR implementation supports polyglot persistence without an ONM framework.

## 9.4 Summary

This thesis has presented the design of a candidate repository that can be used to unify RCS subscriber data. It has shown that MNOs need not interrogate multiple repositories for subscriber data since a single one can provide shared access to heterogeneous data over the network. Therefore, the thesis proposes the use of this repository, which exposes its data sets using a standardised protocol while providing flexible querying capabilities to its clients.

# References

- [1] JavaServerFaces.org. Available at: <http://www.javaserverfaces.org/>. [Accessed: 27 May, 2016].
- [2] Kundera: A JPA 2.1 compliant polyglot object-datastore mapping library for NoSQL datastores. Available at: <https://github.com/impetus-opensource/Kundera>. [Accessed: 20 June, 2016].
- [3] Maven - Welcome to Apache Maven. Available at: <https://maven.apache.org/>. [Accessed: 26 January, 2016].
- [4] nosql-databases.org. Available at: <http://www.nosql-databases.org/>. [Accessed: 11 January, 2016].
- [5] Wireshark · go deep. Available at: <https://www.wireshark.org/>. [Accessed: 15 August, 2016].
- [6] 3GPP. *TR 23.845: Study on User Data Convergence (UDC) evolution*, December 2015. Release 13.
- [7] 3GPP. *TR 29.935: Study on the User Data Convergence (UDC) Data Model*, December 2015. Release 13.
- [8] 3GPP. *TS 22.101: Service Principles*, September 2015. Release 14.
- [9] 3GPP. *TS 22.985: Service Requirements for User Data Convergence (UDC)*, December 2015. Release 13.
- [10] 3GPP. *TS 23.335: Service Requirements for User Data Convergence (UDC); Technical realization and information flows*, December 2015. Release 13.
- [11] 3GPP. *TS 29.228: IP Multimedia (IM) Subsystem Cx and Dx Interfaces; Signalling flows and message contents*, December 2015. Release 13.
- [12] 3GPP. *TS 29.328: IP Multimedia (IM) Subsystem Sh Interface; Signalling flows and message contents*, December 2015. Release 13.
- [13] 3GPP. *TS 29.335: User Data Convergence (UDC); User Data Repository Access Protocol over the Ud interface; Stage 3*, December 2015. Release 13.
- [14] 3GPP. *TS 32.181: User Data Convergence (UDC); Framework for model handling and management*, January 2016. Release 13.

- 
- [15] 3GPP. *TS 32.182: User Data Convergence (UDC); Common Baseline Information Model (CBIM)*, January 2016. Release 13.
  - [16] ABADI, D. J. Consistency tradeoffs in modern distributed database system design. *Computer-IEEE Computer Magazine* 45, 2 (2012), 37.
  - [17] AHO, A. V., AND ULLMAN, J. D. *Foundations of computer science: C edition*. 1995.
  - [18] APACHE SOFTWARE FOUNDATION. Apache Olingo Library. Available at: <https://olingo.apache.org>. [Accessed: 6 January, 2016].
  - [19] BAR-SINAI, M. Big data technology literature review. *Communications of the ACM* 13, 7 (2015), 422–426.
  - [20] BELLAVISTA, P., CORRADI, A., AND FOSCHINI, L. IMS-based presence service with enhanced scalability and guaranteed QoS for interdomain enterprise mobility. *IEEE Wireless Communications* 16, 3 (2009), 16–23.
  - [21] BERNERS-LEE, T., FIELDING, R., FRYSTYK, H., GETTYS, J., LEACH, P., MASINTER, L., AND MOGUL, J. RFC 2616: Hypertext Transfer Protocol–HTTP/1.1.
  - [22] BETZ, H., GROPENGIESSER, F., HOSE, K., AND SATTLER, K.-U. Learning from the history of distributed query processing: a heretic view on linked data management. In *Proceedings of the Third International Conference on Consuming Linked Data-Volume 905* (2012), CEUR-WS.org, pp. 15–26.
  - [23] BREWER, E. CAP twelve years later: How the "rules" have changed. *Computer* 45, 2 (2012), 23–29.
  - [24] CABIBBO, L. ONDM: an object-nosql datastore mapper. *Faculty of Engineering, Roma Tre University*. Retrieved April 21st (2013).
  - [25] CAMARILLO, G., KAUPPINEN, T., KUPARINEN, M., AND IVARS, I. M. Towards an innovation oriented IP Multimedia Subsystem . *IEEE Communications Magazine* 45, 3 (2007), 130–136.
  - [26] CATTELL, R. Scalable SQL and NoSQL data stores. *Acm Sigmod Record* 39, 4 (2011), 12–27.
  - [27] CHAPPELL, D. Introducing OData: Data access for the web, the cloud, mobile devices, and more. *Microsoft Whitepaper, May* (2011).
  - [28] CHODOROW, K. *MongoDB: the definitive guide*. O'Reilly Media, Inc., 2013.
  - [29] CODD, E. F. A relational model of data for large shared data banks. *Communications of the ACM* 13, 6 (1970), 377–387.
  - [30] COMVERSE. Making RCS Happen. Available at: <http://www.slideshare.net/comverseinc/rcs-vision-whitepaper18forweb>. [Accessed: 13 December, 2015].

- [31] COPELAND, R. *Converging NGN wireline and mobile 3G networks with IMS: converging NGN and 3G mobile*, vol. 7. CRC Press, 2008.
- [32] CORE NETWORK DYNAMICS GMBH. OpenIMS - The Open Source IMS Core Project. Available at: <http://www.openimscore.org>. [Accessed: 12 May, 2016].
- [33] CORONEL, C., AND MORRIS, S. *Database systems: design, implementation, & management*. Cengage Learning, 2016.
- [34] DA XU, L. Enterprise systems: state-of-the-art and future trends. *IEEE Transactions on Industrial Informatics* 7, 4 (2011), 630–640.
- [35] DELAC, G., BUDISELIC, I., ZUZAK, I., SKULIBER, I., AND STEFANEC, T. A methodology for SIP and SOAP integration using application-specific protocol conversion. *ACM Transactions on the Web (TWEB)* 6, 4 (2012), 15.
- [36] DELANEY, J. Rcs and joyn: Keeping operators at the center of communications. Technical Report HW62U, December 2012.
- [37] DOUBANGO. DoubangoTelecom/boghe: IMS/RCS client for WP8, Surface and Desktop with support for CUDA, Intel Quick Sync, DXVA2. Available at: <https://github.com/DoubangoTelecom/boghe>. [Accessed: 7 February, 2016].
- [38] DOUBANGO. DoubangoTelecom/imsdroid: High Quality Video SIP/IMS client for Google Android. Available at: <https://github.com/DoubangoTelecom/imsdroid>. [Accessed: 7 February, 2016].
- [39] DOUBANGO TELECOM. Doubango - open source 3GPP IMS/LTE framework for embedded systems. Available at: <https://www.doubango.org/>. [Accessed: 7 February, 2016].
- [40] DUSSEAULT, L., AND SNELL, J. RFC 5789: PATCH method for HTTP.
- [41] ECLIPSE FOUNDATION. Jetty - Servlet Engine and HTTP Server. Available at: <http://www.eclipse.org/jetty/>. [Accessed: 27 July, 2016].
- [42] ELLIOT, B., BLOOD, S., AND KRAUS, D. Magic quadrant for unified communications. *Gartner RAS Core Research Note G 160407* (2011).
- [43] EVANS, D. An introduction to Unified Communications: challenges and opportunities. In *Aslib Proceedings* (2004), Emerald Group Publishing Limited, pp. 308–314.
- [44] FAJARDO, V., ARKKO, J., LOUGHNEY, J., AND ZORN, G. RFC 6733: Diameter Base Protocol.
- [45] FIELDING, R., AND RESCHKE, J. RFC 7230: Hypertext Transfer Protocol (HTTP/1.1); Message Syntax and Routing.
- [46] GESSERT, F., FRIEDRICH, S., WINGERATH, W., SCHAARSCHMIDT, M., AND RITTER, N. Towards a scalable and unified REST API for cloud data stores. In *GI-Jahrestagung* (2014), pp. 723–734.

- [47] GSMA. Act now to implement RCS. Available at: <http://www.gsma.com/network2020/wp-content/uploads/2014/12/Evaluate-RCS-Today-Dec-2014.pdf>. [Accessed: 7 December, 2014].
- [48] GSMA. IMS-based interoperability in South Korea. Available at: <http://www.gsma.com/network2020/wp-content/uploads/2012/10/RCSCaseStudy-SouthKorea.pdf>. [Accessed: 17 April, 2015].
- [49] GSMA. Rich Communication Services. Available at: <http://www.gsma.com/network2020/technology/rcs/>. [Accessed: 25 March, 2015].
- [50] GSMA. Rich Communication Services. Available at: <http://www.gsma.com/network2020/technology/enriched-calling-with-rs/>, 2012. [Accessed: 25, March 2015].
- [51] GSMA. *Joyn Blackbird Product Definition Document*, January 2014. Version 3.0.
- [52] GSMA. *Rich Communication Suite 5.3 Advanced Communications Services and Client Specification*, February 2015. Version 6.0.
- [53] GSMA. *Rich Communication Suite 5.3 Endorsement of OMA CPM 2.0 Message Storage*, February 2015. Version 5.0.
- [54] GSMA. *Rich Communication Suite 5.3 Endorsement of OMA SIP SIMPLE IM 2.0*, February 2015. Version 4.0.
- [55] GSMA. *Service Provider Device Configuration*, February 2015. Version 2.0.
- [56] HAN, J., HAIHONG, E., LE, G., AND DU, J. Survey on NoSQL database. In *Pervasive computing and applications (ICPCA), 2011 6th international conference on* (2011), IEEE, pp. 363–366.
- [57] HENRY, K., LIU, Q., AND PASQUEREAU, S. Rich Communication Suite. In *Intelligence in Next Generation Networks, 2009. ICIN 2009. 13th International Conference on* (2009), IEEE, pp. 1–6.
- [58] HUHS, M. N., AND SINGH, M. P. Service-Oriented Computing: Key concepts and principles. *IEEE Internet computing* 9, 1 (2005), 75–81.
- [59] HYUN, W., PARK, S., LEE, I., AND KANG, S. A study on design and implement of xcap server. In *2006 8th International Conference Advanced Communication Technology* (2006), vol. 1, IEEE, pp. 4–pp.
- [60] ISLAM, S., AND GRÉGOIRE, J.-C. User-centric service provisioning for IMS. In *Proceedings of the 6th International Conference on Mobile Technology, Application & Systems* (2009), ACM, p. 5.
- [61] JBOSS. Wildfly Homepage. Available at: <http://wildfly.org/>. [Accessed: 28 July, 2016].
- [62] LASKEY, K. B., AND LASKEY, K. Service Oriented Architecture. *Wiley Interdisciplinary Reviews: Computational Statistics* 1, 1 (2009), 101–105.

- [63] LIN, M., AND ARIAS, J. A. Rich Communication Suite: The challenge and opportunity for MNOs. In *Intelligence in Next Generation Networks (ICIN), 2011 15th International Conference on* (2011), IEEE, pp. 187–190.
- [64] LINTHICUM, D. S. Defining, Designing, and Implementing SOA-Based Data Services. White paper, David S. Linthicum LLC, 2009.
- [65] MAGEDANZ, T., BLUM, N., AND DUTKOWSKI, S. Evolution of SOA concepts in telecommunications. *Computer* 40, 11 (2007), 46–50.
- [66] MONGODB. MongoDB for GIANT ideas. Available at: <https://www.mongodb.com/>. [Accessed: 23 July, 2016].
- [67] NAYAK, A., PORIYA, A., AND POOJARY, D. Type of NOSQL databases and its comparison with relational databases. *International Journal of Applied Information Systems* 5, 4 (2013), 16–19.
- [68] NURSEITOV, N., PAULSON, M., REYNOLDS, R., AND IZURIETA, C. Comparison of JSON and XML data interchange formats: A case study. *Caine 2009* (2009), 157–162.
- [69] OASIS. Open Data Protocol (OData) Technical Committee. Available at: [https://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=odata](https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=odata). [Accessed: 14 March, 2016].
- [70] OASIS. *[OData-Atom-Format-v4.0]: OData Atom Format Version 4.0*, November 2013.
- [71] OASIS. *[OData-Data-Agg-v4.0]: OData Extension for Data Aggregation Version 4.0*, November 2015.
- [72] OASIS. *[OData-JSON-Format-v4.0]: OData JSON Format Version 4.0 Plus Errata 03*, June 2016.
- [73] OASIS. *[OData-Part1]: OData Version 4.0. Part 1: Protocol Plus Errata 03*, March 2016.
- [74] OASIS. *[OData-Part2]: OData Version 4.0. Part 2: URL Conventions Plus Errata 03*, March 2016.
- [75] OASIS. *[OData-Part3]: OData Version 4.0. Part 3: Common Schema Definition Language (CSDL) Plus Errata 03*, March 2016.
- [76] ODATA.ORG. OData Ecosystem. Available at: <http://www.odata.org/ecosystem>. [Accessed: 6 January, 2016].
- [77] OLSON, M. A., BOSTIC, K., AND SELTZER, M. I. Berkeley DB. In *USENIX Annual Technical Conference, FREENIX Track* (1999), pp. 183–191.
- [78] OMA. *XML Document Management Architecture*, April 2012. Version 2.1.
- [79] OMA. *User Plane Location Protocol*, September 2014. Version 2.1.

- [80] PAPAOGLOU, M. P., TRAVERSO, P., DUSTDAR, S., AND LEYMAN, F. Service-Oriented Computing: A research roadmap. *International Journal of Cooperative Information Systems* 17, 02 (2008), 223–255.
- [81] PAPAOGLOU, M. P., AND VAN DEN HEUVEL, W.-J. Service Oriented Architectures: approaches, technologies and research issues. *The VLDB journal* 16, 3 (2007), 389–415.
- [82] PAUTASSO, C., AND WILDE, E. RESTful web services: Principles, patterns, emerging technologies. In *Proceedings of the 19th International Conference on World Wide Web* (2010), WWW '10, ACM, pp. 1359–1360.
- [83] PFEIL, M. What is persistence and why does it matter. Available at: <http://www.datastax.com/dev/blog/what-persistence-and-why-does-it-matter>. [Accessed: 19 August, 2016].
- [84] POSTMAN. Postman | Supercharge your API workflow. Available at: <https://www.getpostman.com/>. [Accessed: 11 July, 2016].
- [85] PRIME TEK. PrimeFaces. Available at: <http://www.primefaces.org/>. [Accessed: 20 July, 2016].
- [86] PRITCHETT, D. BASE: An ACID alternative. *Queue* 6, 3 (2008), 48–55.
- [87] REDIS.IO. Redis. Available at: <http://www.redis.io>. [Accessed: 06 May, 2016].
- [88] RESCHKE, J., AND FIELDING, R. RFC 7231: Hypertext Transfer Protocol (HTTP/1.1); Semantics and Content.
- [89] RESTCOMM. RestComm/jdiameter: RestComm Diameter Stack and Services. Available at: <https://github.com/RestComm/jdiameter>. [Accessed: 23 August, 2016].
- [90] RESTCOMM. RestComm/sip-servlets: Leading SIP - IMS - WebRTC Application Server. Available at: <https://github.com/RestComm/sip-servlets>. [Accessed: 23 August, 2016].
- [91] RIEMER, K., AND TAING, S. Unified Communications. *Business & Information Systems Engineering* 1, 4 (2009), 326–330.
- [92] ROSENBERG, J. RFC 4825: The Extensible Markup Language Configuration Access Protocol.
- [93] ROSENBERG, J., SCHULZRINNE, H., CAMARILLO, G., JOHNSTON, A., PETERSON, J., SPARKS, R., HANDLEY, M., AND SCHOOLER, E. RFC 3261: SIP - Session Initiation Protocol.
- [94] ROTH, M., AND TAN, W.-C. Data Integration and Data Exchange: It's Really About Time. In *6th Biennial Conference on Innovative Data Systems Research (CIDR)* (2013), Citeseer.



- [95] SADALAGE, P. J., AND FOWLER, M. *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. Pearson Education, 2012.
- [96] SCHERZINGER, S., KLETTKE, M., AND STÖRL, U. Managing schema evolution in NoSQL data stores. *arXiv preprint arXiv:1308.0514* (2013).
- [97] SCHRAM, A., AND ANDERSON, K. M. MySQL to NoSQL: data modeling challenges in supporting scalability. In *Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity* (2012), ACM, pp. 191–202.
- [98] SELLAMI, R., BHIRI, S., AND DEFUDE, B. ODBAPI: a unified REST API for relational and NoSQL data stores. In *2014 IEEE International Congress on Big Data* (2014), IEEE, pp. 653–660.
- [99] SERMERSHEIM, J. RFC 4511: Lightweight Directory Access Protocol (LDAP); The Protocol.
- [100] SOGUNLE, O., TSIETSI, M., AND TERZOLI, A. Adopting the OData protocol as a Ud reference point for user data convergence. In *Southern Africa Telecommunication Networks and Applications Conference* (2016), SATNAC, pp. 402–407.
- [101] STÖRL, U., HAUF, T., KLETTKE, M., SCHERZINGER, S., AND REGENSBURG, O. Schemaless NoSQL Data Stores-Object-NoSQL Mappers to the Rescue? In *BTW* (2015), pp. 579–599.
- [102] STRAUCH, C., SITES, U.-L. S., AND KRIHA, W. NoSQL databases. *Lecture Notes, Stuttgart Media University* (2011).
- [103] TRIVEDI, N., AND JAIN, A. Implementation Challenges in Rich Communication Suite-enhanced (RCS-e). In *Networks, 2013. ICN 2013. Twelfth International Conference on* (2013), IARA, pp. 132–136.
- [104] VARGAS-SOLAR, G. Polyglot persistence and multi-cloud data management solutions. In *Tutorial Talk in EDBT Summer School on Data all around Big, Linked, Open* (2013).
- [105] WANG, X., SCHULZRINNE, H., KANDLUR, D., AND VERMA, D. Measurement and analysis of LDAP performance. *IEEE/ACM Transactions on Networking (TON)* 16, 1 (2008), 232–243.
- [106] ZEILENGA, K. RFC 4512: Lightweight Directory Access Protocol (LDAP); Directory Information Models.

# Appendix A

## Additional Tables

### A.1 RCS Device Configuration

When an RCS subscriber wants to register for the first time, the subscription profile containing valid subscriber identity are stored in the network. Thereafter, the RCS client is configured with the correct settings. Upon completion of this process, the registration procedure is invoked in the network. First time registration comprises two main activities:

- Register.
- Establish; The service finds a subset among the subscriber's existing contacts who also use RCS.

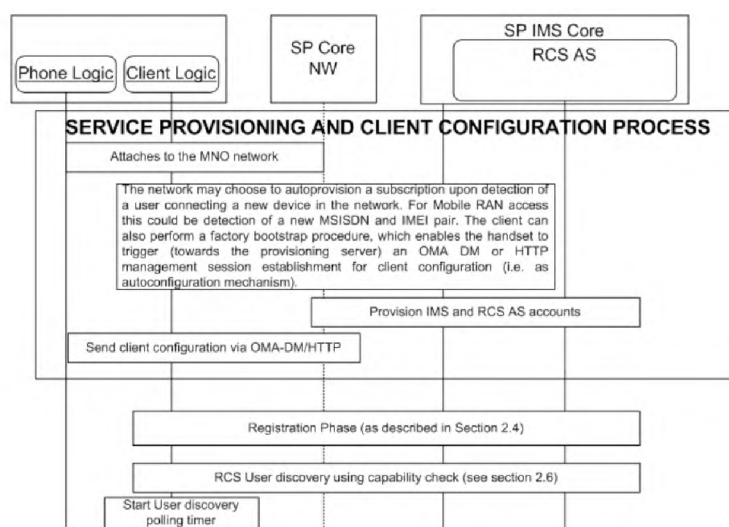


Figure A.1: Sequence Diagram for first time startup of an RCS device. Source: [52].

## A.2 Device Configuration Parameters

Table A.1: RCS device configuration parameters. Source: [52].

Parameter	Description	Mandatory	Format
<code>rcs_state</code>	<p>It contains one of the following values: <math>\{-4, -3, -2, -1, 0, \text{positive integers} \dots\}</math></p> <ul style="list-style-type: none"> <li>• A positive value denotes the most recent version of the configuration document.</li> <li>• 0 denotes that the configuration is obsolete and may require an update.</li> <li>• -1 indicates that the MNO has disabled RCS services on the UE or Client.</li> <li>• -2 indicates that RCS is disabled on the UE but configuration can be manually triggered by a subscriber.</li> <li>• -3 indicates that RCS has not been configured on the UE.</li> <li>• -4 indicates that the subscriber has explicitly disabled the RCS services on the UE.</li> </ul>	Yes	Integer
<code>rcs_version</code>	Identifies the version of RCS, which is supported by the Client	Yes	Case-Sensitive String
<code>rcs_profile</code>	Identifies the set of RCS services supported by the Client	No	Case-Sensitive String
<code>client_vendor</code>	Identifies the vendor providing the Client	Yes	Case-Sensitive String
<code>client_version</code>	Identifies the version of the Client itself	Yes	Case-Sensitive String

default_sms_app	<p>It contains one of the following values: {0, 1, 2}</p> <ul style="list-style-type: none"> <li>• 0 indicates that the native operating system does not allow the subscriber to select an SMS application on the Client or the application can not be found.</li> <li>• 1 indicates the RCS messaging client is selected as the default SMS application.</li> <li>• 2 indicates that another messaging client is the default SMS application.</li> </ul>	<p>Conditional</p> <p>– Yes, for Clients supporting SMS,</p>	Integer
-----------------	---	--	---------

## A.3 EDM Data Types

This section provides an overall view of primitive data types present within the OData's EDM.

Table A.2: EDM Primitive Data Types.

Type	Description
Edm.Binary	Binary data
Edm.Boolean	Binary-valued logic
Edm.Byte	Date without a time-zone offset
Edm.DateTimeOffset	Date and time with a time-zone offset, no leap seconds
Edm.Decimal	Numeric values with fixed precision and scale
Edm.Double	Floating-point number (15-17 decimal digits)
Edm.Duration	Signed duration in days, hours, minutes, and (sub)seconds
Edm.Guid	16-byte (128-bit) unique identifier
Edm.Int16	Signed 16-bit integer
Edm.Int32	Signed 32-bit integer
Edm.Int64	Signed 64-bit integer
Edm.SByte	Signed 8-bit integer
Edm.Single	floating-point number (6-9 decimal digits)
Edm.Stream	Binary data stream
Edm.String	Sequence of UTF-8 characters
Edm.TimeOfDay	Clock time 00:00-23:59:59.999999999999
Edm.Geography	Abstract base type for all Geography types
Edm.GeographyPoint	A point in a round-earth coordinate system

Edm.GeographyLineString	Line string in a round-earth coordinate system
Edm.GeographyPolygon	Polygon in a round-earth coordinate system
Edm.GeographyMultiPoint	Collection of points in a round-earth coordinate system
Edm.GeographyMultiLineString	Collection of line strings in a round-earth coordinate system
Edm.GeographyMultiPolygon	Collection of polygons in a round-earth coordinate system
Edm.GeographyCollection	Collection of arbitrary Geography values
Edm.Geometry	Abstract base type for all Geometry types
Edm.GeometryPoint	Point in a flat-earth coordinate system
Edm.GeometryLineString	Line string in a flat-earth coordinate system
Edm.GeometryPolygon	Polygon in a flat-earth coordinate system
Edm.GeometryMultiPoint	Collection of points in a flat-earth coordinate system
Edm.GeometryMultiLineString	Collection of line strings in a flat-earth coordinate system
Edm.GeometryMultiPolygon	Collection of polygons in a flat-earth coordinate system
Edm.GeometryCollection	Collection of arbitrary Geometry values

## A.4 Olingo Processors

This section presents the Olingo's list of the currently supported EDM processors for OData version 4. It is important to note that these processors are implemented as Java Interfaces within the Library.

Table A.3: Olingo EDM Processors.

Processor	Description
ActionComplexCollection Processor	It handles an action request and returns a Collection of Complex Types.
ActionComplex Processor	It handles an action request and returns a Complex Type.
ActionEntityCollection Processor	It handles an action request and returns an Entity Collection.
ActionEntity Processor	It handles an action request and returns an Entity Type
ActionPrimitiveCollection Processor	It handles an action request and returns a Collection of Primitive Types.
ActionPrimitive Processor	It handles an action request and returns a Primitive Type.
ActionVoid Processor	It handles an action request and returns no Data Type.
Batch Processor	It handles a single instance of an Entity Type.
ComplexCollection Processor	It handles Complex Type instances.
Complex Processor	It handles an instance of a Complex Type

CountComplexCollection Processor	Can be used to count a collection of complex properties. For example, counting the number of Complex Types within the EDM.
CountEntityCollection Processor	Can be used to count a collection of Entities. For example, number of Entities in an Entity Set.
CountPrimitiveCollection Processor	Can be used to count a collection of primitive types. For example, counting number of elements in a set of type <code>Edm.String</code> .
Delta Processor	It handles a single instance of a Delta response.
EntityCollection Processor	It handles Entity Collections.
Entity Processor	It handles an instance of an Entity Type.
Error Processor	It handles the any error that occurs by any of the processors within the Library.
MediaEntity Processor	It handles Media Entity Types
Metadata Processor	It handles the Metadata Document
PrimitiveCollection Processor	It handles a collection of Primitive Types (Strings, Integers, and Boolean, among others).
Primitive Processor	It handles an instance of a Primitive Type.
PrimitiveValue Processor	It handles the raw value of a Primitive Type.
Processor	Is the generic processor from which others are derived. Thus, it is the base-interface for all processors.
ReferenceCollection Processor	It handles a collection of Entity references.
Reference Processor	It handles a single instance of an Entity reference.
ServiceDocument Processor	It handles the Service Document.

## A.5 Diameter Command Codes

This section presents the command codes defined for Diameter Sh and Cx interfaces. It also presents the AVP codes for defined for the Sh-UDR Data reference command.

### A.5.1 Sh Codes

The table below shows the different command codes for requests and response messages sent through Sh interface.

Table A.4: Diameter Sh Codes. Source: [12].

Command Name	Code
User-Data-Request (UDR)	306
User-Data-Answer (UDA)	306
Profile-Update-Request (PUR)	307

Profile-Update-Answer (PUA)	307
Subscribe-Notifications-Request (SNR)	308
Subscribe-Notifications-Answer (SNA)	308
Push-Notification-Request (PNR)	309
Push-Notification-Answer (PNA)	309

### A.5.2 Cx Codes

The table below shows the different command codes for requests and response messages sent through Cx interface.

Table A.5: Diameter Cx Codes. Source: [11].

Command Name	Code
User-Authorization-Request (UAR)	300
User-Authorization-Answer (UAA)	300
Server-Assignment-Request (SAR)	301
Server-Assignment-Answer (SAA)	301
Location-Info-Request (LIR)	302
Location-Info-Answer (UAR)	302
Multimedia-Auth-Request (MAR)	303
Multimedia-Auth-Answer (MAA)	303
Registration-Termination-Request (RTR)	304
Registration-Termination-Answer (RTA)	304
Push-Profile-Request (PPR)	305
Push-Profile-Answer (PPA)	305

### A.5.3 Sh Data-Reference AVP Codes

The Data-Reference AVP is used when making a request for specific user data through the Sh interface. The table below shows the possible values of the Data-Reference AVP, which is of an Enumerated data type.

Table A.6: Diameter Sh Data Reference Codes. Source: [12].

Data Value	Code
RepositoryData	0
IMSPublicIdentity	10
IMSUserState	11
S-CSCFName	12
InitialFilterCriteria	13
LocationInformation	14

---

APPENDIX A. ADDITIONAL TABLES

---

UserState	15
ChargingInformation	16
MSISDN	17
PSIActivation	18
DSAI	19
ServiceLevelTraceInfo	21
IPAddressSecureBindingInformation	22
ServicePriorityLevel	23
SMSRegistrationInfo	24
UEReachabilityForIP	25
TADSInformation	26
STN-SR	27
UE-SRVCC-Capability	28
ExtendedPriority	29
CSRN	30
ReferenceLocationInformation	31
IMSI	32
IMSPriateUserIdentity	33



# Appendix B

## Additional Figures

### B.1 OMA XDM Architecture

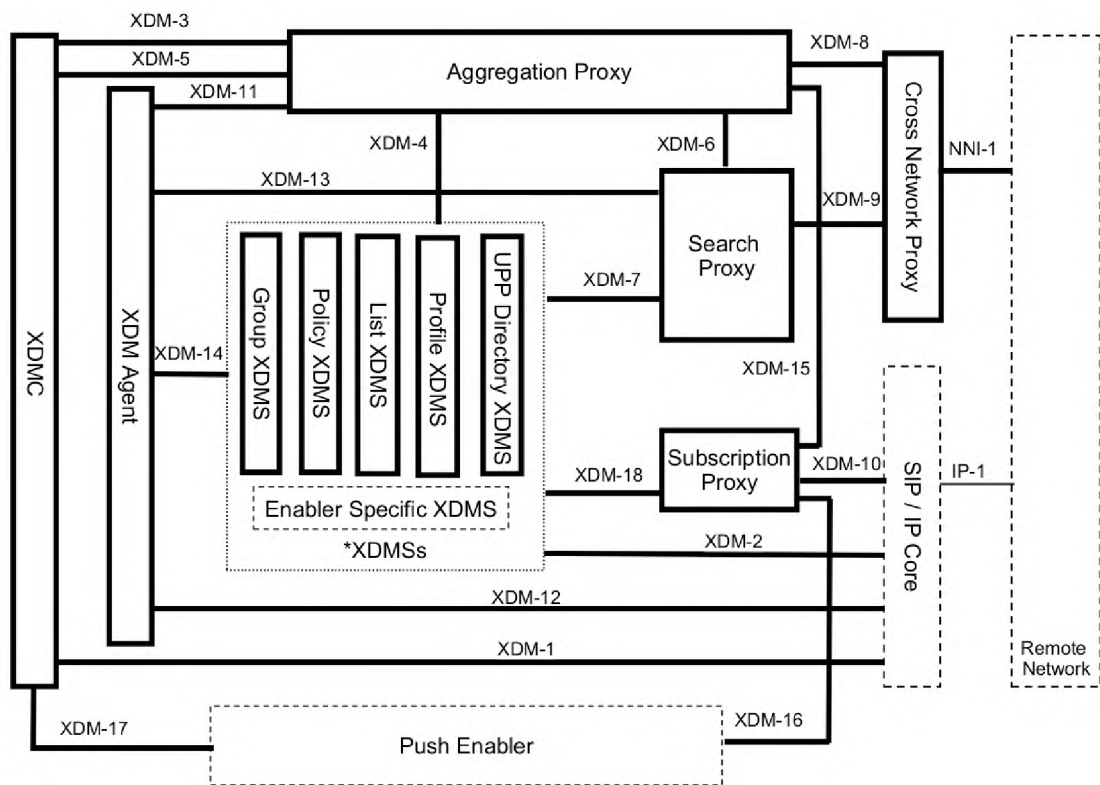


Figure B.1: OMA XDM Architecture. Source: [78].

B.2 HSS FE DM

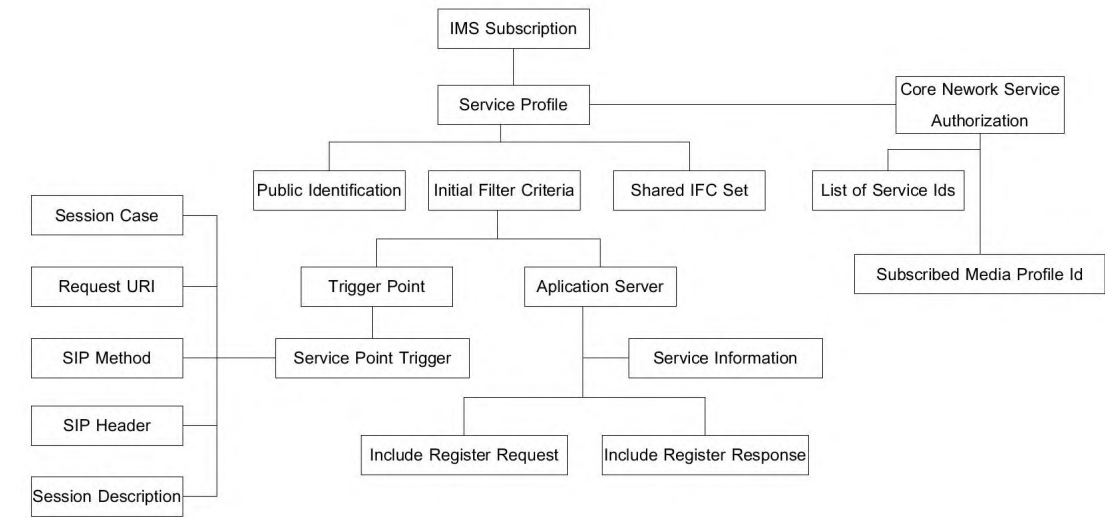


Figure B.2: Cx Data.

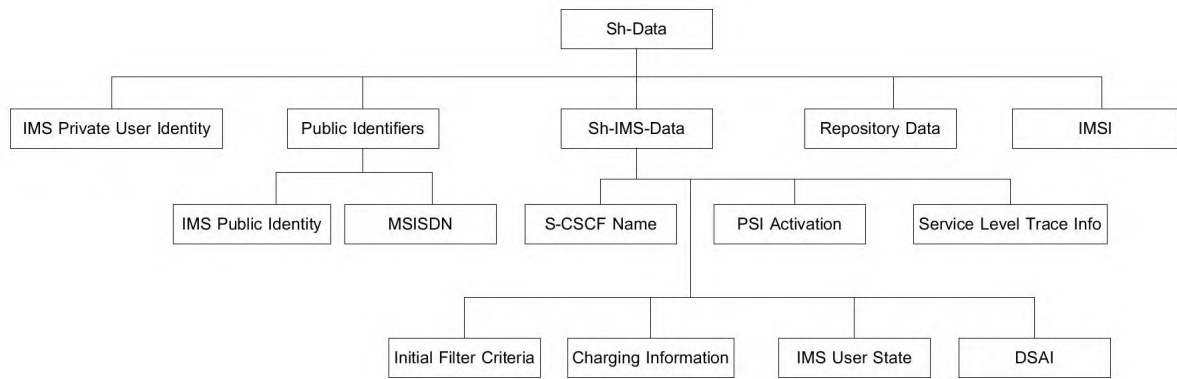


Figure B.3: Sh Data.

B.3 XCAP DM

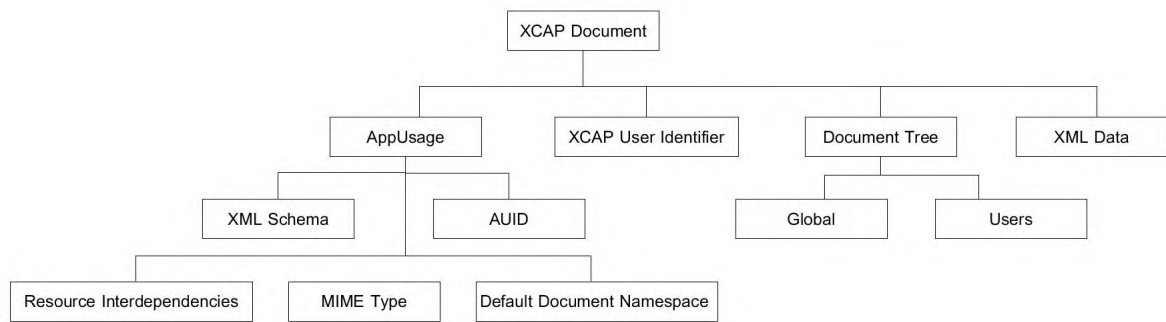


Figure B.4: XCAP Document.

## B.4 Provisioning the CSDR

The data sets managed by the CSDR can be categorised into two: Statically-created and Dynamic-generated data sets. The subscription profile and XCAP documents are in the former category while the latter consists of the log and session-related data. This section discusses how profiles and documents can be created in the CSDR. Thus, it is organised as follows: First, a brief discussion on how FEs are registered. This is followed by the creation of subscription profiles. Thereafter, the creation of XCAP documents. Finally, the section discusses the use of the various query handlers which have been supported by the CSDR.

### B.4.1 Registering the FEs on the CSDR

To register the aforementioned FEs in the CSDR, the backend registration interface is used. This is because it is easier to use when compared to passing the parameters as JSON documents on a REST client interface. The following information is required for the registration:

- FE Name.
- FE IP Address.
- Assigned OData Namespace.

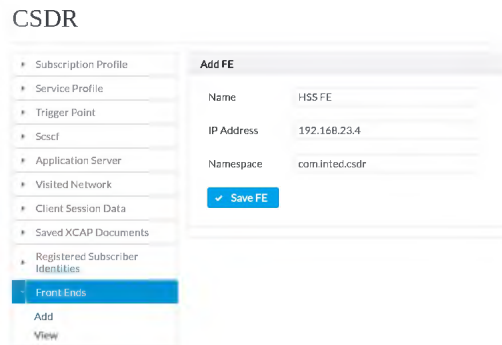


Figure B.5: Snapshot of FE registration page.

### B.4.2 Creating Subscription Profiles

To configure the subscription profiles in the CSDR, the Service Profile which consists of the iFCs, among others was created.

CSDR

Subscription Profile

Add

View

Service Profile

Add

View

Trigger Point

Add

View

Scscf

Add

View

Add Scscf

Name

SCSCF 1

SIP URI

sip:scscf@open-ims.test

Diameter URI

Save SCSCF

Figure B.6: Snapshot of Add SCSCF page.

CSDR

Subscription Profile

Service Profile

Trigger Point

Scscf

Application Server

Visited Network

Client Session Data

Saved XCAP Documents

Registered Subscriber Identities

Front Ends

Add

View

Add Application Server

Name

messaging

SIP URI

sip:messaging@open-ims.test

Diameter URI

Application Server - HSS FE Interaction

Sh Permissions	UDR	PUR
Allow Request	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Aliases-Repository-Data	<input type="checkbox"/>	<input type="checkbox"/>
Charging-Info	<input type="checkbox"/>	<input type="checkbox"/>
DSAI	<input type="checkbox"/>	<input type="checkbox"/>
IFC	<input type="checkbox"/>	<input type="checkbox"/>
IMPU	<input checked="" type="checkbox"/>	<input type="checkbox"/>
IMS-User-State	<input type="checkbox"/>	<input type="checkbox"/>
PSI Activation	<input type="checkbox"/>	<input type="checkbox"/>

Save Application Server

Figure B.7: Snapshot of Add Application Server page.

CSDR

Subscription Profile

Service Profile

Trigger Point

Scscf

Application Server

Visited Network

Client Session Data

Saved XCAP Documents

Registered Subscriber Identities

Front Ends

Add

View

Add Trigger Point

Name

messaging TP

Condition Type CNF

DISJUNCTIVE NORMAL FORM

Number of attached SPT Groups

0

Remove Groups

Service Point Triggers

☐ SIP\_METHOD

MESSAGE

Remove

Add SPT

Attach SPT Group

Save Trigger Point

Figure B.8: Snapshot of Add Trigger Point page.

CSDR

Subscription Profile

Service Profile

Trigger Point

Scscf

Application Server

Visited Network

Client Session Data

Saved XCAP Documents

Registered Subscriber Identities

Front Ends

Add Service Profile

Name

sp 1

Attached IFCs

Priority	IFC Name	Application Server	Trigger Point	Profile Part Indicator	
0	ifc 1	telephony	telephony TP	ANY	<div>- Remove</div>
1	ifc2	messaging	messaging TP	ANY	<div>- Remove</div>

Attach IFC to Service Profile

IFC Name

Application Server

telephony

Trigger Point

telephony TP

Profile Part Indicator

ANY

+ Attach

Save Service Profile

Figure B.9: Snapshot of Add Service Profile page.

CSDR

Subscription Profile

Service Profile

Trigger Point

Scscf

Add Visited Network

Name

open-ims.test

Save Visited Network

Figure B.10: Snapshot of Add Visited Network page.

Attached IMS Public Identities

Identity Type	SIP URI	Can Register	Is Barred	Display Name	Visited Network Identifier	
PUBLIC_USER_IDENTITY	slp:mike3@open-ims.test	true	false	INACTIVE	open-ims.test	<div>- Remove</div>

Attach IMS Public Identity

SIP URI

slp:mike3@open-ims.test

Identity Type

PUBLIC\_USER\_IDENTITY

Barring

False

Can Register

True

Display Name

MIKE 2

PSI Activation

INACTIVE

Visited Network Identifier

open-ims.test

+ Attach

Save Subscription Profile

Figure B.11: Snapshot of Add Subscription Profile with IMPU page.

CSDR

Subscription Profile

Add

View

Service Profile

Add

View

Trigger Point

Add

View

Scscf

Add

View

Application Server

Visited Network

Add

View

Name

Mike

DSAI Value

INACTIVE

Service Profile

sp 1

Scscf

SCSCF 1

Private Identity

URI \*

mike@open-ims.test

Secret Key (Password)

mike

Attached IMS Public Identities

Identity Type	SIP URI	Can Register	Is Barred
No records found.			

Figure B.12: Snapshot of Add Subscription Profile with IMPI page.

## B.5 Redis

Snapshots of the contents and startup display are presented as follows.

```

06:54:16 Sep 06:54:43.518 # WARNING: The TCP backlog setting of 511 cannot be enforced because /proc/sys/net/
06:54:16 Sep 06:54:43.518 # You requested maxclients of 10000 requiring at least 10032 max file descriptors
06:54:16 Sep 06:54:43.518 # Server can't set maximum open files to 10032 because of OS error: Operation not
06:54:16 Sep 06:54:43.518 # Current maximum open files is 4096, maxclients has been reduced to 4064 to comp

Redis 3.2.1 (00000000/0) 64 bit
Running in standalone mode
Port: 6379
PID: 5254

http://redis.io

06:54:16 Sep 06:54:43.520 # WARNING: The TCP backlog setting of 511 cannot be enforced because /proc/sys/net/
06:54:16 Sep 06:54:43.520 # Server started, Redis version 3.2.1
06:54:16 Sep 06:54:43.520 # WARNING overcommit_memory is set to 0! Background save may fail under low mem
06:54:16 Sep 06:54:43.520 # the command 'sysctl vm.overcommit_memory=1' for this to take effect.
06:54:16 Sep 06:54:43.520 # WARNING you have Transparent Huge Pages (THP) support enabled in your kernel.
06:54:16 Sep 06:54:43.529 * DB loaded from disk: 0.009 seconds
06:54:16 Sep 06:54:43.529 * The server is now ready to accept connections on port 6379
  
```

Figure B.13: Starting up a Redis Server

```

127.0.0.1:6379> hgetall ClientSessionData:sip:tessa@open-ims.test
1) "mediaType"
2) "message"
3) "lastKnownSessionId"
4) "(475273224;c1dbf239-bcb9-746c-1547-8dfb8581ac9b;dbf56c1f;com.intel.as.telephony.Main)"
5) "mediaPort"
6) "54275"
7) "lastKnownState"
8) "CONFIRMED"
9) "Id"
10) "slp:tessa@open-ims.test"
11) "mediaProtocol"
12) "TCP/MSRP"
13) "userAgent"
14) "well"
  
```

Figure B.14: Session data generated while Tessa was sharing a file.

# Appendix C

## Configuration Files

### C.1 HSS FE

#### C.1.1 Jetty AS

The default configuration file for Jetty is `jetty.xml`, which is placed at the same directory as the Maven POM document.

Listing C.1: HSS FE `jetty.xml` configuration file.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configure id="Server" class="org.eclipse.jetty.server.Server"
>
  <New id="httpConfig" class="org.eclipse.jetty.server.
    HttpConfiguration">
    <Set name="secureScheme">https</Set>
    <Set name="securePort">
      <Property name="jetty.secure.port" default="
        8443" />
    </Set>
    <Set name="outputBufferSize">32768</Set>
    <Set name="requestHeaderSize">8192</Set>
    <Set name="responseHeaderSize">8192</Set>
    <Set name="sendServerVersion">true</Set>
    <Set name="sendDateHeader">false</Set>
    <Set name="headerCacheSize">512</Set>
  </New>
</Configure>
```

#### C.1.2 Deployment Descriptor

The HSS FE uses the `web.xml` file to describe how it is to be deployed by Jetty AS.



Listing C.2: HSS FE web.xml configuration file.

```
<web-app>
  <display-name>Archetype Created Web Application</display-name>
  <servlet>
    <servlet-name>HSS</servlet-name>
    <display-name>HSS</display-name>
    <description></description>
    <servlet-class>com.inted.as.hss.fe.HSS</servlet-class>
    <load-on-startup>0</load-on-startup>
  </servlet>
  <listener>
    <listener-class>com.inted.as.hss.fe.AppDaemon</listener-class>
  </listener>
  <servlet-mapping>
    <servlet-name>HSS</servlet-name>
    <url-pattern>/HSS</url-pattern>
  </servlet-mapping>
</web-app>
```

## C.2 XCAP FE

The XCAP FE also uses the web.xml service descriptor.

Listing C.3: XCAP FE web.xml configuration file.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:web="http://xmlns.jcp.org/xml/ns/javaee">
  <display-name>Archetype Created Web Application</display-name>
  <servlet>
    <description></description>
    <display-name>XcapServer</display-name>
    <servlet-name>XcapServer</servlet-name>
    <servlet-class>com.inted.xcap.server.fe.XcapServer</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>XcapServer</servlet-name>
    <url-pattern>/*</url-pattern>
  </servlet-mapping>
</web-app>
```

## C.3 Telephony FE

This FE uses the sip.xml file instead of the web.xml since it is a SIP application. The web.xml are used by services that respond to HTTP requests.

Listing C.4: Telephony FE sip.xml configuration file.

```
<?xml version="1.0" encoding="UTF-8"?>
<sip-app>
  <!-- Archetype Created SIP Application -->
  <app-name>com.inted.as.telephony.Main</app-name>

  <servlet-selection>
    <main-servlet>
      MainServlet
    </main-servlet>
  </servlet-selection>

  <servlet>
    <servlet-name>MainServlet</servlet-name>
    <display-name>MainServlet</display-name>
    <description>Telco Telephony Front End Service
    </description>
    <servlet-class>
      com.inted.as.telephony.Main
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
</sip-app>
```

## C.4 Messaging FE

Listing C.5: Messaging FE sip.xml configuration file.

```
<?xml version="1.0" encoding="UTF-8"?>
<sip-app>
  <!-- Archetype Created SIP Application -->
  <app-name>com.inted.as.messaging.Main</app-name>

  <servlet-selection>
    <main-servlet>
      MainServlet
    </main-servlet>
  </servlet-selection>

  <servlet>
    <servlet-name>MainServlet</servlet-name>
    <display-name>MainServlet</display-name>
```

```

        <description>Telco Messaging Front End Service
        </description>
        <servlet-class>
            com.inted.as.messaging.Main
        </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
</sip-app>

```

## C.5 CSDR

### C.5.1 Deployment Descriptor

Listing C.6: CSDR web.xml configuration file.

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"      xsi:
    schemaLocation="http://java.sun.com/xml/ns/javaee http://
    java.sun.com/xml/ns/javaee/web-app_3_0.xsd"      version=
    "3.0">
    <display-name>The CSDR - Consolidated Subscriber Data
        Repository</display-name>
    <context-param>
        <param-name>javax.faces.PROJECT_STAGE</param-
            name>
        <param-value>Development</param-value>
    </context-param>
    <context-param>
        <param-name>primefaces.THEME</param-name>
        <param-value>omega</param-value>
    </context-param>
    <welcome-file-list>
        <welcome-file>index.xhtml</welcome-file>
    </welcome-file-list>
    <servlet>
        <servlet-name>CsdrApp</servlet-name>
        <servlet-class>com.inted.csdr.web.CsdrApp</
            servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>CsdrApp</servlet-name>
        <url-pattern>/svc/*</url-pattern>
    </servlet-mapping>
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>

```

```
        <servlet-class>javax.faces.webapp.FacesServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.xhtml</url-pattern>
    </servlet-mapping>

    <filter>
        <filter-name>AuthFilter</filter-name>
        <filter-class>com.inted.csdr.auth.AuthFilter</
        filter-class>
    </filter>

    <filter-mapping>
        <filter-name>AuthFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
</web-app>
```

## C.5.2 POM File

Listing C.7: CSDR pom.xml configuration file.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="
    http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
        http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.inted</groupId>
    <artifactId>csdr</artifactId>
    <packaging>war</packaging>
    <version>1.0</version>
    <name>csdr</name>
    <url>http://maven.apache.org</url>

    <properties>

        <!-- java version -->
        <java-version>1.8</java-version>

        <!-- JUnit -->
        <junit-version>4.12</junit-version>

        <!-- HTTP Servlet -->
        <javax.version>2.5</javax.version>

        <!-- ODATA -->
```

```
<odata.version>4.2.0</odata.version>

<!-- SLF4J LOGGER -->
<slf4j.version>1.7.13</slf4j.version>

<!-- Kundera -->
<kundera.version>3.4</kundera.version>

<!-- MySQL -->
<mysql.version>5.1.20</mysql.version>

<!-- Maven compiler plugin -->
<compiler-plugin-version>3.3</compiler-plugin-
    version>

<vaadin.version>7.6.2</vaadin.version>
<vaadin.plugin.version>${vaadin.version}</
    vaadin.plugin.version>

</properties>

<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>${junit-version}</version>
        <scope>provided</scope>
    </dependency>

    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>servlet-api</artifactId>
        <version>${javax.version}</version>
        <scope>provided</scope>
    </dependency>

    <!-- Olingo OData -->
    <dependency>
        <groupId>org.apache.olingo</groupId>
        <artifactId>odata-server-api</
            artifactId>
        <version>${odata.version}</version>
    </dependency>

    <dependency>
        <groupId>org.apache.olingo</groupId>
        <artifactId>odata-server-core</
            artifactId>
        <version>${odata.version}</version>
    </dependency>
</dependencies>
```

```
<dependency>
  <groupId>org.apache.olingo</groupId>
  <artifactId>odata-commons-core</
    artifactId>
  <version>${odata.version}</version>
</dependency>

<dependency>
  <groupId>org.apache.olingo</groupId>
  <artifactId>odata-commons-api</
    artifactId>
  <version>${odata.version}</version>
</dependency>

<!-- Slf4j logger -->
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-api</artifactId>
  <version>${slf4j.version}</version>
</dependency>

<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-simple</artifactId>
  <version>${slf4j.version}</version>
</dependency>

<!-- Kundera for MongoDB -->
<dependency>
  <groupId>com.impetus.kundera.client</
    groupId>
  <artifactId>kundera-mongo</artifactId>
  <version>${kundera.version}</version>
</dependency>

<!-- Kundera for Redis -->
<dependency>
  <groupId>com.impetus.kundera.client</
    groupId>
  <artifactId>kundera-redis</artifactId>
  <version>${kundera.version}</version>
</dependency>

<!-- Kundera for RDBMS -->
<dependency>
  <groupId>com.impetus.kundera.client</
    groupId>
  <artifactId>kundera-rdbms</artifactId>
  <version>${kundera.version}</version>
</dependency>
```

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</
    artifactId>
  <version>${mysql.version}</version>
</dependency>

<dependency>
  <groupId>com.impetus.kundera.core</
    groupId>
  <artifactId>fallback-impl</artifactId>
  <version>${kundera.version}</version>
</dependency>
<dependency>
  <groupId>org.primefaces</groupId>
  <artifactId>primefaces</artifactId>
  <version>6.0</version>
</dependency>
<dependency>
  <groupId>com.sun.faces</groupId>
  <artifactId>jsf-api</artifactId>
  <version>2.2.13</version>
</dependency>
<dependency>
  <groupId>org.primefaces.extensions</
    groupId>
  <artifactId>all-themes</artifactId>
  <version>1.0.8</version>
  <type>pom</type>
</dependency>
</dependencies>

<build>
  <finalName>csdr</finalName>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.
        plugins</groupId>
      <artifactId>maven-compiler-
        plugin</artifactId>
      <version>${compiler-plugin-
        version}</version>
      <configuration>
        <source>${java-version}
          </source>
        <target>${java-version}
          </target>
      </configuration>
    </plugin>
  </plugins>
</build>
```

```
        </build>  
</project>
```