Towards an Architecture for Real-Time Heterogeneous Data Dissemination

Raymond Peterkin

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering
School of Electrical Engineering and Computer Science
University of Ottawa

I hereby declare that I am the sole author of this thesis.

I authorize the University of Ottawa to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Raymond Peterkin

I further authorize the University of Ottawa to reproduce this thesis by photocopying or other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Raymond Peterkin

ABSTRACT

The popularity of the Internet and cellular networks in recent years has led to widespread use of these networks for data dissemination. Advancements in wireless communications networks like Long Term Evolution (LTE) and 5G networks indicate a trend towards higher speed mobile networks accessed by users with increasingly sophisticated mobile devices. These advancements do not include the means for users to leverage increased wireless network transmission rates by disseminating heterogeneous data. The ability to share heterogeneous data entirely through mobile networks illustrates a communications methodology where end users determine what data will be shared with no limitations based on content. Advances in this domain have introduced many aspects of this methodology on a global scale but have not addressed how heterogeneous data can be defined and distributed by end users.

This thesis proposes a communications architecture for the dissemination of heterogeneous data in real-time. The implementation of this communications architecture is based on the IP Multimedia Subsystem (IMS) [90] and proposes modifications to the Session Initiation Protocol (SIP) [98] to permit the distribution of heterogeneous data through synchronous and asynchronous data transmission. A communications protocol is introduced defining a broker, data consumers, data producers and a message format. Algorithms for session management and packet processing are provided in an embedded system called the Reconfigurable Multimedia Collaborative System (RMCS) [58]. Results for the communications architecture demonstrate the successful generation and transmission of heterogeneous data in real-time using the proposed modifications to SIP. The successful operation of packet generation algorithms and related functions in hardware is also provided to demonstrate the means through which this architecture could be provided through hardware for optimal performance and to facilitate its implementation into mobile devices.

ACKNOWLEDGMENT

I wish to express my sincere gratitude to my supervisor, Dr. Dan Ionescu, for his constant guidance, encouragement, and support throughout my research. It would not have been possible to finish this long-term work without his support.

I would also like to express my sincere thanks to the members of my advisory committee and thesis examiners for their encouragement, discussions and suggestions.

I also wish to thank my colleagues and friends in the Network Computing Control and Technologies (NCCT) laboratory, namely Mohamed Abou-Gabal and Fadi El-Hassan, for their help and encouragement.

Finally, I would like to thank my parents (Raymond and Joan), sister (Leanne) and wife (Allison) for their support, patience, kindness, inspiration and love.

Contents

List of Figures

List of Tables

Notation

Various symbols, superscripts, subscripts, and abbreviations used frequently in this proposal are summarized below.

Acronyms and Definitions

3G	3rd Generation
AOR	Address of Record
CSCF	Call Session Control Function
DDS	Data Distribution Services
DNS	Domain Name System
EDGE	Enhanced Data rates for GSM Evolution
FPGA	Field Programmable Gate Array
GSM	Global System for Mobile Communications
HSS	Home Subscriber Server
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
IM	Instant Message
IMS	IP Multimedia System
ISP	Internet Service Provider
KQML	Knowledge Query and Manipulation Language
LDAP	Lightweight Directory Access Protocol
MAC	Message Authentication Code

OSI	Open System Interconnection
PIDF	Presence Information Data Format
PSTN	Public Switched Telephone Network
RFID	Radio Frequency Identification
RLS	Resource List Server
RMCS	Reconfigurable Multimedia Collaborative System
SA	Security Association
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SDP	Session Description Protocol
SIP	Session Initiation Protocol
SQL	Structured Query Language
SQTL	Sensor Query and Tasking Language
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UA	User Agent
UAC	User Agent Client
UAS	User Agent Server
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

WSDL	Web Services Description Language
XCAP	XML Configuration Access Protocol
XML	Extensible Markup Language
XSD	XML Schema Document
XSLT	Extensible Stylesheet Language Transformation

Chapter 1

Introduction

1.1 Motivation

As Internet usage has increased in recent years, the dissemination of heterogeneous data across public networks has become increasingly commonplace. Heterogeneous data may be disseminated between a diverse collection of data producers and data consumers across multiple networks. Personal computers (PCs) were traditionally used to consume and produce data through the Internet using the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP). New technologies have emerged as the popularity of data dissemination has increased. Autonomous robotic systems and unmanned aerial vehicles (UAVs) are used in industrial applications and research projects to produce data. Increased dissemination of heterogeneous data has been a contributing factor to the increased popularity of big data. Big data is a term used to describe a collection of data sufficiently large and complex that data management and processing are difficult with traditional methods. Numerous technical challenges exist in the domain of big data due to the complex and diverse nature of the data collected including data acquisition, storage, transfer and analysis. Furthermore, the widespread popularity of mobile networks has lead to the extensive use of smart phones and tablet computers as data consumption devices. As new technologies have emerged to facilitate network-based data dissemination, no single solution has been developed accounting for the heterogeneity of data producers, data consumers and networks.

A well known approach for the dissemination of real-time data between multiple parties is

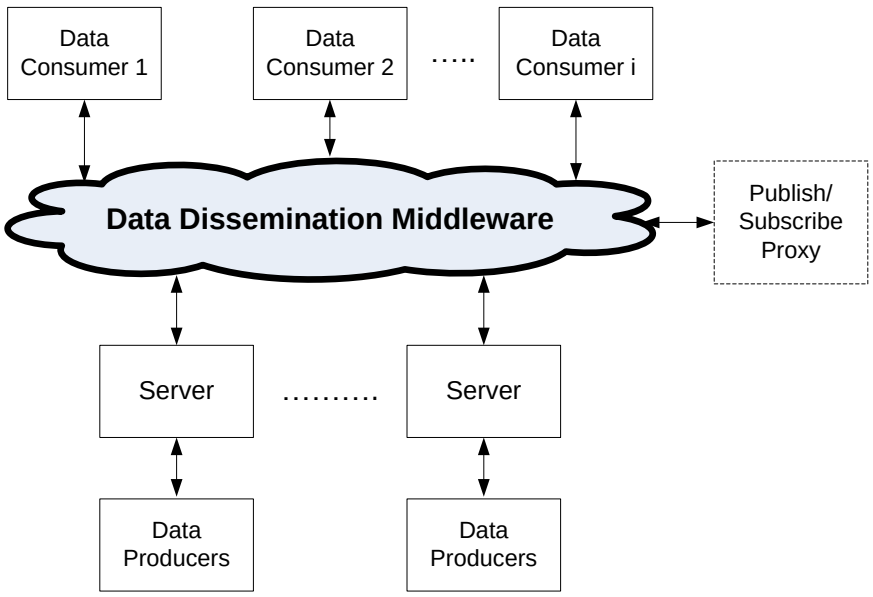

Figure 1.1: Middleware Connecting Data Producers and Data Consumers.

illustrated in Fig. 1.1. Data consumers are connected to data sources through a middleware application. This middleware application provides the means for each data producer and data consumer to exchange data across a network and potentially other services. Data consumers are connected directly to the middleware while data producers are connected through servers that must be properly configured to accept requests through the middleware and generate responses. Middleware architectures implementing a strict client/server methodology do not decouple data producers from their servers and do not permit real-time data requests based on content. Therefore publish/subscribe mechanisms are often implemented through a proxy to decouple all data consumers and data producers from their connection points. This approach to data dissemination does not permit the dissemination of real-time data from mobile data producers.

The Data Distribution Service (DDS) [83] is a specification for distributed systems permitting the data available from data producers to be published in the network so subscriptions may be created through data consumers for real-time data. Additional services are provided through DDS including data definition, message addressing, message delivery and flow control. All the facilities required to disseminate real-time data are provided through DDS but no mechanisms are defined for mobile network integration or for mobile data pro-

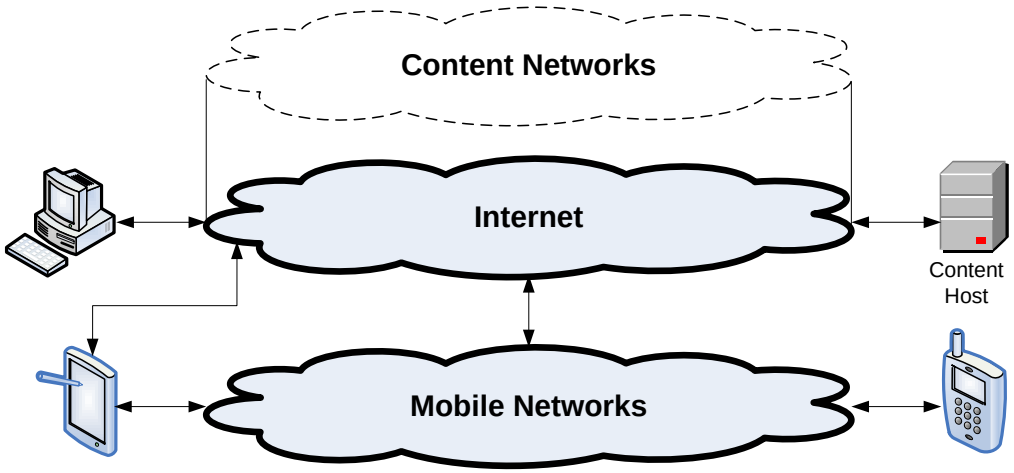

Figure 1.2: Data Dissemination through the Internet and Mobile Networks.

ducers.

Mobile networks have become increasingly sophisticated in recent years. 3G networks provide support for multimedia services while transmitting data at peak throughput of 3.1 Mbps and 700 Kbps for digital broadband and packet data respectively. 3.5G networks have an increased peak throughput of 14.4Mbps. Throughput rates are sufficiently increased in modern 4G networks to permit high definition streaming with peak rates of 100 - 300 Mbps. The trend of increased throughput in mobile networks is expected to increased with 5G networks in the future with the potential of peak rates in the range of gigabits per second. Increased throughput and new multimedia streaming services in mobile networks will provide the means to transmit large amounts of heterogeneous data between data consumers and data producers in real-time.

Content networks [22] are a collection of overlays to improve delivery and service provisioning for all Internet-based content through the use of web caches, content adaptation and other facilities. The integration of content networks and mobile networking technologies is illustrated in Fig. 1.2.

The combination of content networks and middleware based solutions do not permit all possible data producers and data consumers to be integrated across the Internet and mobile networks for heterogeneous data dissemination. The primary motivation of this thesis is to introduce a communications architecture integrating heterogeneous data producers and data

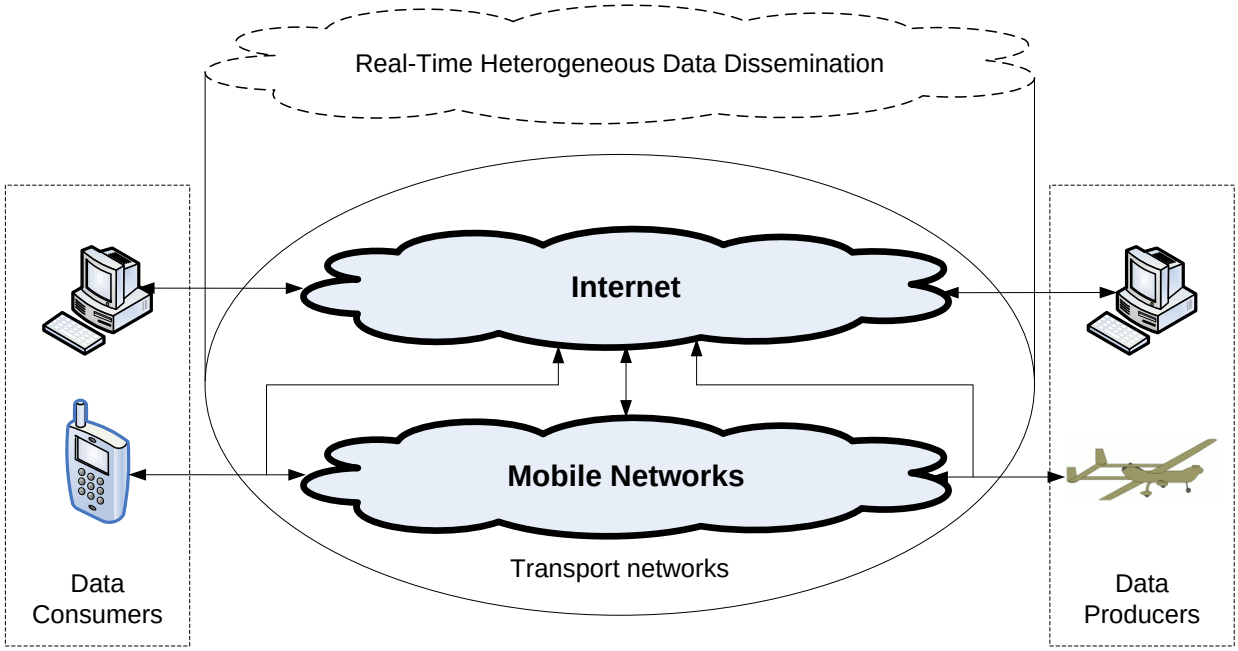

Figure 1.3: A Communications Architecture for Real-Time Heterogeneous Data Dissemination.

consumers across the Internet and mobile networks. Such an architecture would provide the communications infrastructure necessary to exchange all types of data in real-time. A high level description of this architecture is given in Fig. 1.3.

In contrast with the approach used by traditional networks, the proposed architecture would permit the definition and transmission of arbitrarily defined data for consumption by any device connected to the Internet or a mobile network. Where content networks are used primarily to make existing data more readily available to existing users, this architecture would make it possible to make data previously unavailable for transmission across these media in as ubiquitous a manner by providing the means to define and transmit heterogeneous data in real-time and making them part of the communications network infrastructure.

1.2 A Brief Review

The increased use of mobile networks in recent years has led to numerous development efforts to integrate them with IP networks for service deployment and integration. A framework

integrating these networks could leverage the existing infrastructure of IP networks across the world to provide several domains of convergence to any individual with a communications device using a cellular network of their choice. The IP Multimedia Subsystem (IMS) has emerged as a widespread approach to combine fixed and mobile networks to enhance the functionality available to users through their mobile devices.

IMS is an architecture for the convergence of mobile and fixed networks permitting the convergence of audio, video and text based data on those networks [90]. From the perspective of a mobile device user, the convergence provided through IMS allows for the delivery of a fully customized user experience from multiple locations and devices. Users registered with an IMS network will gain access to all their services at the same time. Those services include push to talk, voice and video communications, instant messaging, online gaming and presence which allows them to communicate their status to other users available on the network. An efficient and robust implementation of the IMS communications mechanisms is necessary to access all these services and others at the same time with one device. Such mechanisms are traditionally implemented in software and deployed on communications devices. If the necessary IMS communications protocols were deployed through hardware, IMS operations could be performed more efficiently and be easily integrated into mobile devices.

Numerous networks exist providing backbone type services like PSTN, PLMN, and IP networks. Each one of these networks provides access to users through specific communications protocols. Various wireless networks also exist but are not compatible with wired, backbone networks. IMS effectively combines all these networks into a unified core network through which end users may gain access through multiple devices as shown in Fig. 1.4.

The network convergence achieved through IMS permits the convergence of end user devices to access a unified core network through any device accessible to an individual network. The services provided on each of the individual networks can all be accessed directly through IMS. Therefore, users have unprecedented customization options to combine elements of several services to meet their needs using an embedded device.

IMS provides the mechanisms for users to publish their presence to other individuals and subscribe to the presence of others. Presence may be defined as the dynamic profile of an individual. Users wishing to transmit updates to their presence to others may write

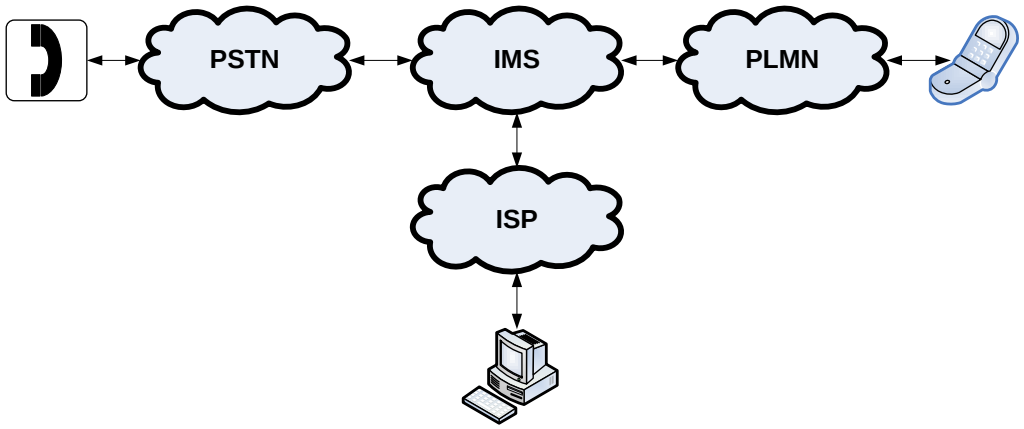

Figure 1.4: Network Integration with IMS.

their updating profile to a presence server which is broadcast to individuals to have created successful subscriptions to this information. Individuals subscribing to the presence of several individuals may write a friend list to a Resource List Server (RLS) and subscribe to the entire list. Other types of data may be written to an RLS including access policy for the individuals publishing their presence and presence management so perform tasks like transmitting a default presence value when a user is offline.

IMS is based on a wide range of protocols developed substantially through the Internet Engineering Task Force (IETF). IMS is based substantially on the Session Initiation Protocol (SIP) and Session Description Protocol (SDP). SIP is a protocol used to establish, modify and terminate sessions between two users for point-to-point communications. Only one type of data can be exchanged during a single session. Therefore, users who want to exchange different types of data simultaneously would require multiple sessions for data transmission and other mechanisms to synchronize data as required. Session information is described with SDP and it is passed in the payload of a SIP packet. SIP features that may be used directly for mobile sensor network communication include the following:

- Portability: The location of an end user may be changed while using a single identifier. Users may use this feature to seamlessly change their location if they must quickly monitor a mobile sensor network from another device.

- Subscription: A user may wish to initiate asynchronous data exchange for the purpose

of receiving automatic updates when certain occur. Subscriptions remove the need to continuously perform synchronous requests when real-time updates would be more suitable. This feature permits users to receive updates when a mobile sensor network or its corresponding environment have changed.

Numerous extensions have been developed for SIP to provide features related to distributed communications. Extensions that can be used in the control of mobile sensor networks include the following:

- Instant Messaging: An extension exists allowing users to transmit a text based message to another individual without requiring a session. This feature permits synchronous data exchange between a mobile sensor network and its clients.

- Notification: An extension exists allowing users to notify subscribed participants of changes through a subscription using an arbitrary data format. This feature was used in SIP to permit the transmission of presence updates.

The presence service available in IMS through SIP and various servers provides the basic mechanisms for users to transmit asynchronous information through subscriptions and write configuration information in the network to request data from groups of users. These features present the basic mechanisms to transmit data from a group of data producers to data consumers. However these features are provided through an address based perspective and do not permit end users to publish or request data based on the description of content. Customized data types required for data producers cannot be defined using the presence service of IMS. Therefore no mechanisms are available for an end user to publish heterogeneous data. End users may subscribe to the presence of several individuals but may not subscribe to a subset of data defined across a set of end users. Furthermore, no mechanisms exist for end users to request the subset of an individual's presence but it would be necessary for a data consumer to request a subset of data from a data producer.

Custom applications can be integrated into IMS to provide functionality that would otherwise not be available. However the communications protocols used in IMS limit the types of data that can be disseminated. Therefore, heterogeneous data dissemination cannot

be provided through IMS using its existing facilities. Additionally, custom applications cannot be implemented in a distributed manner so the distribution of data from mobile data producers is not possible.

Where there has been some work in establishing communications platforms for sharing real-time data, primarily in the domain of sensor networks, there has been extremely limited work done in this domain as it relates to mobile networks. The increasing popularity of these devices and growing capacity of their networks presents an opportunity for a global middleware solution where individuals can define and share completely heterogeneous information with the ability to request any global subset of information and dictate how their data can be shared with no discernible delay. The absence of these features in communications networks, particularly mobile networks, present the basis of the contribution for this thesis.

1.3 Contributions

The contribution of this thesis is the introduction of a communications architecture to disseminate heterogeneous data in real-time across divergent networks. This contribution is made through the design and development of different algorithms and protocols to facilitate the transmission of heterogeneous data through IMS networks. Algorithms were developed to permit the dissemination of heterogeneous data using synchronous and asynchronous communication. A communications protocol was designed and implemented permitting participants to transmit and request an arbitrary subset of heterogeneous data available in the network. This protocol also provides the means for the asynchronous distribution of heterogeneous data availability. An FPGA-based system is used to transmit and receive data using SIP to ensure that heterogeneous data may be sent through IMS networks in real-time.

1.4 Thesis Outline

Chapter 2 is the literature overview which provides a detailed description of relevant issues and research pertaining to the dissemination of data between data producers and data consumers. Important issues and common communications frameworks for data dissemina-

tion are identified and contributions in this field are summarized to describe the problem addressed by the contributions of this thesis.

Chapter 3 provides a high level description of the communications architecture. The high level description includes requirements, specifications and the identification of the components required for data dissemination. IMS and SIP are described in detail to provide justification for their use as the communications framework of the architecture.

Chapter 4 describes the data dissemination requirements for the communications architecture. Session management features are identified to illustrate how application level sessions are mapped to the sessions of the communications protocol of the architecture. Numerous use cases are defined to illustrate the functionality of the architecture in all circumstances identified for real-time heterogeneous data dissemination.

Chapter 5 describes the protocol required for the communications architecture. The protocol defines a data consumer, data producer and the broker as data dissemination components. Algorithms and sub system components are illustrated in significant detail using finite state machines and architectural diagrams. Message formats are provided describing the requests, responses and alerts defined for heterogeneous data dissemination.

Chapter 6 defines the communications framework used to exchange data between all components in the architecture. This chapter describes how the high level protocol defined in the previous chapter data is successfully mapped to SIP so data can be successfully passed through an IMS network. The contributions made to the subscription and notification mechanisms of SIP are clearly identified.

Chapter 7 describes the Reconfigurable Multimedia Collaborative System (RMCS), an embedded system used to perform all tasks related to SIP for the communications architecture. The RMCS contains the functionality required for session management, SIP packet processing, subscriptions and instant messaging. Resource utilization and simulation results for the hardware component of the RMCS are also provided in this chapter.

Chapter 8 describes illustrates the results observed from the operation of the communications architecture. An experimental testbed was created by connecting different RMCS together. Each RMCS was controlled by applications implementing the functionality of a data consumer, data producer and broker. Data was disseminated successfully between the

data consumer and the data producer through the broker by exchanging the appropriate sequence of SIP packets. Latency values observed during data dissemination were noted to help identify circumstances where the communications architecture demonstrates its best and worst performance.

Chapter 9 concludes this thesis and proposes future research plans.

Chapter 2

Literature Review

This section describes the literature review for this thesis. The first section provides an overview for data dissemination by describing how it is performed through middleware and the important issues that are addressed while transmitting data from an arbitrary collection of sources to destinations. The literature describes several domains where data dissemination techniques have been applied. These domains are described in a separate section including sensor networks, publish/subscribe architectures, service oriented architectures and context aware applications. A comprehensive review of middleware implementations in the literature are provided and divided to multiple sections to identify research areas of interest for data dissemination solutions including dissemination data to mobile clients, transmission of multimedia data, network control, security and the use of multicast techniques. The final conclusion of this review is that data dissemination methods described in the literature are application specific and software based.

2.1 Data Dissemination Middleware

2.1.1 Definition

Middleware may be broadly defined as software that facilitates data exchange between multiple applications in the same environment. As described in [50], numerous middleware applications have been developed for traditional systems. Corba [84] is a well known mid-

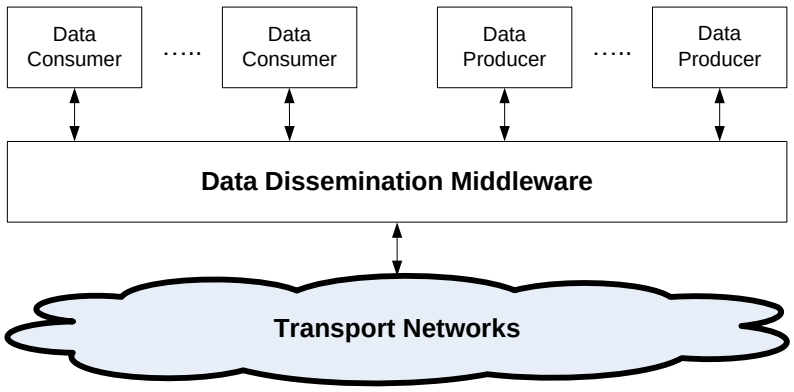

Figure 2.1: Data Dissemination Middleware.

dleware system for data exchange where the environment remains static. Jini [25] provides a protocol for client applications to discover services and manage client-server connections. The LIME middleware [79] provides an application programming interface for mobile ad hoc components. Limbo [21] and FarGo [54] can modify the order of data exchanges or relocate components as required to fulfill requirements for bandwidth availability. Using Odyssey [82] users can subscribe to changes in the data rate of the underlying network while the Spectra [31] component of Aura [39] can be used to monitor network conditions. Middleware may be classified as a specific type depending on the nature of the applications exchanging data and the means used to exchange data. The type of middleware required to exchange data between clusters of data producers and numerous data producers may be described as data dissemination middleware. The relationship between data consumers, data producers and the data dissemination middleware necessary to facilitate their data exchange is illustrated in Fig. 2.1.

2.1.2 Issues

Data Producers

The type of data disseminated between data producers and data consumers may impact other components of the middleware. Many middleware solutions described in the literature refer to the dissemination of documents or single pieces of data that do not change, can be searched by name and whose contents may be directly queried depending on their format.

Caching techniques may also be implemented for document retrieval in middleware solutions based on the frequency of their retrieval by data consumers.

Real-time data do not share the same characteristics as documents insofar as the values generated by real-time data sources are only considered current the moment they are received by data consumers. This type of data cannot be cached or searched due to its frequent changes. Real-time data representing a sequence of values at a specific point in time (typically represented by a collection of strings, numbers and images with metadata to explain their collective format) may be serialized for data dissemination.

One issue to address with real-time data sources is the availability of data while it is being disseminated to data consumers. It is entirely foreseeable that real-time data may become unavailable for significant periods of time depending on the nature of the source, particularly in situations where data is generated from a collection of sensors. Data dissemination middleware transmitting real-time data may be further characterized by the features available to alert data consumers if data sources should become unavailable and the ability to adjust existing data sessions if necessary.

Multimedia data (i.e. audio and video data) is real-time data representing information gathered over a period of time and is not transmitted with any form of metadata to explain its format. A session is typically established in advance to negotiate data formats before multimedia data is transmitted. One type of data (audio or video) is transmitted using a single multimedia session therefore synchronization for audio and video streams must be performed by data consumers if those facilities are not available in the data dissemination middleware.

Data Consumers

Components used to receive data transmitted through data dissemination middleware are highly dependent on the mechanisms used to establish a physical connection to the network. Some data dissemination middleware solutions do not specifically describe a data consumer component but instead describe a high level component intended to act as an interface to the end user. Many middleware implementations described in the literature implement data consumer functionality on PCs connected to middleware through network controller

interfaces or wireless LANs. This category of data consumers are stationary and do not permit users to receive data in different physical locations.

Mobile clients require some mechanism to send and receive data using a wireless communication protocol. Certain data dissemination middleware solutions support receiving middleware data with mobile clients implementing protocols like Bluetooth or WiFi. The range of these clients extends several metres and permit data transmission in areas that are typically enclosed by a single building. Cellular networks would be required to transmit data across a larger geographical area spanning several kilometres.

Embedded communications devices like smart phones and tablets are required as data consumers for data dissemination middleware using cellular networks. Using these devices to receive significant amounts of data presents challenges that do not exist with clients like PC. One such challenge are the relatively small number of processing resources available compared to high performance machines like PCs. Processing significant amounts of data directly from a network in real-time may present a performance bottleneck delaying the presentation of data to the user of the device. Implementing the functionality required to receive and process middleware data using hardware would help to alleviate this bottleneck on mobile communications devices.

Data Transmission

Data may be disseminated from data producers to data consumers using either a push or pull mechanism. Pulling data is a reference to polling data where a data consumer will make an explicit request for data and receive a response with the requested data or some indicating that the data is unavailable. Pushing data refers to data producers transmitting data to an arbitrary subset of data consumers as soon as data becomes available under a specific set of circumstances. Those circumstances could be the availability of data at any time or after a specific sequence of events has occurred like timeouts or when certain data values have been observed. The conditions through which data is pushed from data producers must be specified by the data consumers through an initial request typically submitted in the form of a subscription. The conditions of the subscription dictate how long, how often and under what circumstances data is pushed when it becomes available. Data may be pushed

to multiple recipients periodically (after the occurrence of evenly spaced, time based events) or aperiodically (after the occurrence of unevenly spaced events not necessarily related to time).

When data is pulled or pushed through data dissemination middleware, it may be transmitted to a single recipient or multiple recipients at the same time. The transmission of data from one entity to one other entity through a network is called unicast transmission. Multicast (the transmission of a single piece of data to a subset of available recipients) is a commonly used method of data dissemination. The work in [17] describes how data can be multicast using push and pull methods. Multicast push refers to the repeated dissemination of data from a data producer to multiple data consumers without explicit requests. The most commonly used approaches to data dissemination are based on multicast. This method of multicast is considered most appropriate when the data transmitted is extremely popular among a large number of data consumers for a significant period of time. A repetitive multicast push would lead to an unnecessarily high amount of network traffic. Therefore a multicast pull approach to data dissemination is most appropriate where users requests data that is multicast to all the individuals in a multicast group to aggregate client requests.

Performance

One important issue is scheduling which determines the policies used to resolve contention among multiple requests and select the order and frequency of data multicast. Collectively, these policies determine the performance of the data dissemination middleware as one combination of policies will allow for the optimal transmission of data between a collection of data producers and consumers. The most commonly used measure of performance for data dissemination is the latency perceived by data consumers. This latency measures the time required for a request to be satisfied. The scheduling policies used by data dissemination middleware must adapt to accommodate changing network conditions so latency can be minimized as much as possible. Data dissemination middleware solutions must also implement policies for data consistency and cache replacement as required to ensure that all data received by data consumers represents the most current and up to date information available through the network. However scheduling policies most directly affect the latency

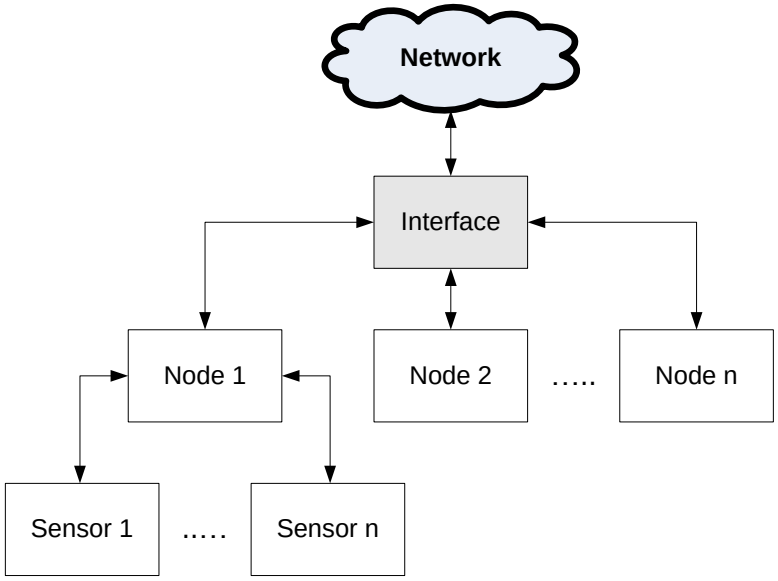

Figure 2.2: Description of a Sensor Network.

experienced by data consumers in receiving data through middleware.

2.2 Data Dissemination Methodologies

2.2.1 Sensor Networks

Very often in the literature, sensors are the primary sources of data disseminated in a communications architecture between data producers and data consumers. Abstractly speaking, a sensor is a device capable of measuring a physical quantity and transforming it into a format that can be correctly interpreted by an instrument, namely a computer or digital device. Sensors are deployed on sensor networks through nodes that provide a high level interface to control every device. In this context, a sensor network may be defined as a collection of nodes with a single communications interface. A high level description of a sensor network is shown in Fig. 2.2.

The functionality provided by the interface is the means to interact with every sensor in the sensor network. Additional features implemented in the sensor network may also be accessed through the interface. While these features are application specific, the ability to retrieve a subset of sensor data through a publicly deployed sensor network is critical

for any operation that may be performed by an end user. The work in [44] outlines the following challenges in the development of data dissemination architectures for wireless sensor networks:

- Limited power and resources among sensors nodes.
- Accommodating application growth in the network.
- Hardware heterogeneity among sensor nodes.
- Dynamic network organization.
- Knowledge of application requirements.
- Data aggregation.
- Quality of Service.
- Security.

2.2.2 Context Awareness

Information may also be disseminated directly by an end user to describe a specific element of their environment. Architectures facilitating this kind of communication are generally described as context aware applications. Context information defines all information that is unique to a user that may be of interest to others like location, user preferences, environment or connectivity preferences[40] as illustrated in 2.3.

2.2.3 Publish/Subscribe Pattern

Publish/subscribe is a messaging pattern whereby entities do not transmit messages directly to their recipients. Instead messages are divided into classes of interest. Entities interested in receiving data create subscriptions to the appropriate set of classes to receive data of interest. Entities generating data are described as publishers while entities requesting data are described as subscribers. Publishers and subscribers are decoupled by an intermediary

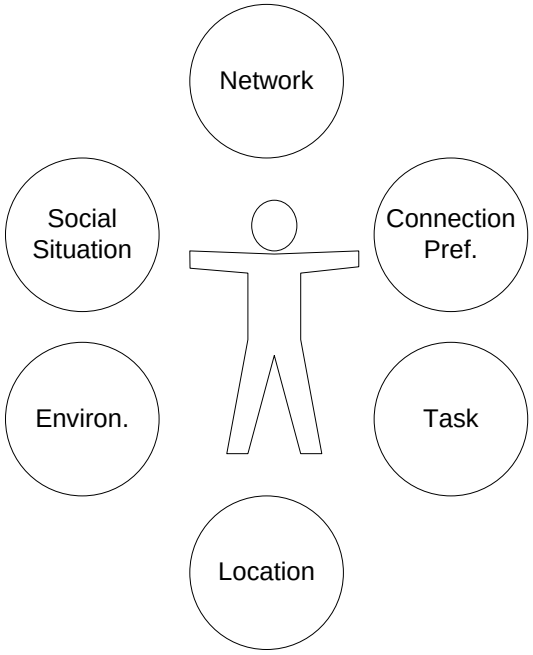

Figure 2.3: Description of Context Data.

such that the collection of publishers have no idea how many subscribers have requested data and vice versa.

In many publish/subscribe systems, publishers post messages to an intermediary message broker and subscribers register subscriptions with that broker as illustrated in Fig. 2.4.

Messages are normally stored at the broker before they are forwarded to the appropriate set of subscribers. Publish/subscribe systems may be characterized as topic based or content based. Topics refers to logical channels defined by publishers. All messages are published to topics and subscribers to a topic will receive data as it arrives. In content based systems, data is delivered to subscribers if it matches a set of criteria specified by the subscriber.

Data dissemination middleware can be implemented using a publish/subscribe architecture permitting clusters of data consumers to disseminate data to multiple data consumers. The roles of publishers and subscribers would be filled by data consumers and clusters respectively. A broker can be implemented to filter data requested by data consumers and forwarded based in a content-based system. Furthermore additional functionality can be implemented into the broker to allow a network administrator to implement scheduling policies for minimizing latency observed by the data consumers.

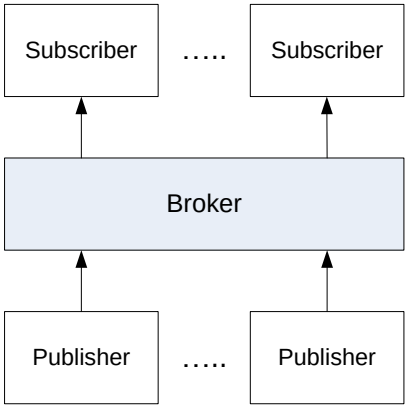

Figure 2.4: Publish/Subscribe Architecture Description.

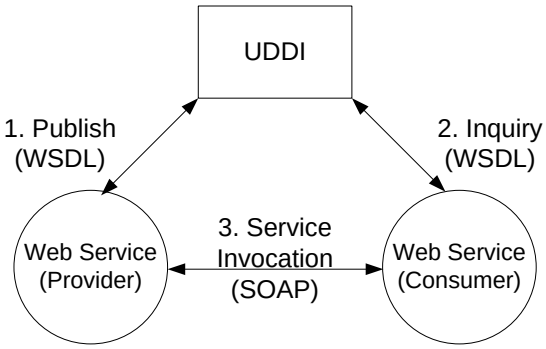

Figure 2.5: Web Services Description.

2.2.4 Web Services

Web services may be defined as a specific combination of technologies designed to facilitate interaction between component based software systems across a network. A high level description of a system implemented using web services is provided in Fig. 2.5.

Each component is described as a service and is used to implement an arbitrary piece of functionality and deployed on a network. Services interact with each other by exchanging messages using the Simple Object Access Protocol (SOAP). A service invoking a request is described as a consumer while the service responding to the request is described as the producer. Services may act as producers or consumers. All services publish information about themselves in a Universal Distribution, Discovery and Interoperability (UDDI) registry using the Web Services Discovery Language (WSDL). This information is retrieved by consumer

services so producers may be queried properly. Web services represent one paradigm through which data dissemination middleware may be developed, especially where data sources are distributed in several different locations.

2.3 Middleware Implementations

2.3.1 Mobile Clients

The integration of sensor applications into IMS through a gateway is discussed in [20] through the presentation of a general architecture for this purpose. Sensor applications are integrated into IMS through a gateway. This gateway contains a communications interface allowing for data to be exchanged with an arbitrary set of sensor nodes (SN). SNs may be equipped with wired communications interfaces like UART or Ethernet but they are likely enabled with the ability to use wireless communications protocols (i.e. Zigbee [1], Bluetooth [7]), especially in the case of mobile sensor nodes. A sensor application contains the intelligence to retrieve sensor data as required and perform any other tasks requiring SN interaction. In this architecture a WiMax [34] modem is used to exchange data with an IMS core but abstractly speaking any communications protocol may be used so long as the necessary mechanisms are available in IMS. Data is transferred through the IMS core and delivered to applications servers, which also act as the high level clients for the sensor application in the gateway.

The design and implementation of a comprehensive gateway to integrate wireless sensor networks with IMS is described in [5]. The gateway is divided into layers for connectivity (establishing and maintaining physical connections to the sensor and 3G [109] networks) and abstraction (transferring data from the wireless sensor network to IMS after formatting).

The work in [4] describes a presence-based architecture for integrating sensor applications with IMS. This architecture is similar to the work described in [5]. The wireless sensor network is connected to a gateway. However the gateway is connected to an extended presence server (PS) and presentity presence proxy. Therefore any data forwarded from the wireless sensor network must be described in a format recognizable to a presence server, namely PIDF. Sensor data is forwarded to end users through the PS or a watcher presence

proxy.

Architectures presented in [5] and [4] provided a detailed description of a gateway implementation for sensor networking applications with IMS that could be easily extended to mobile sensor networks. However they introduce a dependency for all data to be formatted with PIDF and transferred through presence servers before reaching end users. While this approach allows for integration to IMS through its presence facilities, it does not appear necessary to describe sensor data for presentation to an end user. Furthermore no mechanisms are described for the end user to change the configuration of the sensor network in real-time, store sensor data for later retrieval or actuate devices. These features are best provided with a data model described with XML and transported through IMS directly to end users without presence servers.

The work in [3] proposes the design and implementation of a sensor network using IMS. A sensor network gateway registers to the network by transmitting a SIP REGISTER message to the P-CSCF of the IMS network. Once the gateway is registered an XML Document Management Server (XDMS) subscribes to the gateway, effectively becoming an end user to the sensor network. When sensor data is available it is transmitted to the gateway and subsequently forwarded to the XDMS through a SIP NOTIFY message.

A health monitoring system using sensors deployed through IMS is described in [91]. This work is based on the system described in [3]. Electocardiogram (ECG) sensors are installed throughout a medical facility to monitor the condition of various patients. These sensors transmit information to other system components using Radio Frequency (RF). Critical information is transferred to a client using an RTP session established with SIP. Event notification is performed throughout the system by using the SUBSCRIBE/NOTIFY framework of SIP to transmit event information directly from a user to a service provider or an interested party like the family member of a user. An XML-based data management system was implemented to persistently store user information using XDMS. Authorized users access the data using XML Configuration Access Protocol (XCAP) or a SIP SUBSCRIBE message with XQuery.

Examples of sensor network deployment in IMS exist in the literature. The work in [65] describes the deployment of a sensor network with IMS for the purpose of collecting sensor

information to match satellite data using location information. The publication describes a field server (FS) as a sensor node with a set of sensors capable of collecting information about an outdoor environment like temperature, humidity, soil moisture, leaf wetness and carbon dioxide concentration. An FS is also equipped with a camera and WLAN functionality. Numerous FS are deployed and connected through a wireless communication module to a gateway. This gateway is connected to a grid system through IMS where field data can be compared with satellite data by retrieving it from an external data monitoring system. Data comparisons are based on GPS information.

AT&T Mobile Network (AMN) [60] is a middleware platform that allows limited mobile devices to communicate with each other and to securely access tactical corporate and Internet content and services. AMN consists of a set of distributed and collaborative component containers. Gateways are used to handle protocol interfaces to mobile devices and interaction with corporate authentication services aligned with the multiple independent layers of security used in the tactical environment while servers provide direct connections to these services through specialized components that access information based on device user profiles. Message queues are used to exchange data between gateways and servers. A database is deployed in the network to provide operational information.

The Sentient Communication Architecture for Pervasive Environments (SCAPE) [76] is an architecture designed to address a specific set of concerns in disseminating information to mobile clients. One concern is described as a 'bridge' between objects in the real-world and their virtual representation in a computer environment. Techniques used to read information about real-world objects may be classified based on their scalability (can this mechanism be used with large numbers of objects), ability to identify individual objects, ease of use and reader technology. After an object has been given a virtual representation, useful data must be extracted about the object, typically in the form of some ontology. Information must then be converted into a presentable format for individuals. Typical mechanisms for presenting user data are based on XML and XLST style sheets. Adequate security measure must be implemented through techniques to perform authorization and authentication and middleware must be implemented to connect data sources to various clients.

Rebeca [78] described middleware architecture based on the publish-subscribe paradigm

used to disseminate data to mobile clients. Rebeca is a research prototype of a notification service that supports a set of content-based routing algorithms and advertisements. The default data model of Rebeca describes notifications in name/value pairs. Subscriptions are made through the creation of filters that impose constraints on single name/value pairs requested by a data consumer. Rebeca implements functionality to address a specific set of issues regarding data dissemination including wireless connection reliability, durable subscriptions, queuing policies and limiting data rates.

Rebeca provides the means for users to access past notifications as described in [77] and [19]. Rebeca can replay stored notifications if they match a subscription made after their original publication. In this manner, clients may pull past notifications from their histories in an aperiodic manner. When creating a subscription a consumer attaches a replay specification to the subscription describing what past notifications they want to see (for example, all notifications within a specific period of time). The extended subscription is propagated to all concerned histories. When a history receives a subscription, it republishes those stored notifications that match the subscription and replay specification.

2.3.2 Data Retrieval

This category of sensor networking middleware describes solutions developed to retrieve data across a distributed set of sensors using a middleware solution. The work in [8] introduces the concept of a device database system where distributed query execution techniques are applied to leverage the computing capabilities of devices and reduce total network communication. SINA is implemented with components for hierarchical clustering, attribute-based naming and location awareness. Sensors are aggregated to form groups (or clusters) based on their power levels and proximity to each other. The process to generate clusters of sensors may be performed recursively so a hierarchy of clusters are created for data retrieval. Within each cluster there is a node designated as the cluster head responsible for performing functions related to information filtering, fusion and aggregation. If no clusters are formed through the clustering algorithm, all nodes deployed on SINA are considered part of one single cluster. It is presumed that location information will be available for each sensor node deployed on SINA using GPS or some other mechanism to facilitate attribute-based naming for data

queries and location awareness.

Data queries in SINA are performed using the sensor querying and tasking language (SQTL) and sensor execution environment (SEE) illustrated in [59]. Sensor nodes are connected to each other or to a front end, which acts as a gateway. The front end transmits messages into the sensor network on behalf of an end user and forwards responses when necessary. Sensor nodes are considered autonomous and collaborative to reduce the processing burden on the front end.

SQTL messages are decoded by the sensor execution environment (SEE) on each sensor node. Each sensor node is assumed to be executing an arbitrary set of applications that can be manipulated through SQTL message using application IDs to identify the specific application and primitive commands like start, stop, execute, suspend and resume to identify the operation of interest. The functionality of the applications on each sensor node is known in advance to the end user.

Data Service Middleware (DSWare) [69] is a fully-featured sensor networking middleware solution. In addition to the available data service abstraction mechanism, other features are available including data storage, sensor clustering (described as group management), event detection and data subscription. The system permits the retrieval of data across multiple sensors using a SQL like language.

The framework of DSWare consists of several distributed components providing functionality for data subscriptions, event detection, data storage, group management, data caching and scheduling. The Data Storage Component in DSWare provides similar mechanisms to store information according to its semantics with data lookup using two hash algorithms robustness through data duplication to multiple nodes. Data is cached through by duplicating data on multiple locations over the network routing path.

2.3.3 Network Control Solutions

This category of sensor networking middleware describes solutions developed to control network resources as data is disseminated from a collection of sensors. Automatic Service Composition (AutoSeC) is introduced in [45] and it is used to manage resources in a sensor network by providing access control for applications so that QoS requests are maintained.

This approach is similar to middleware for standard networks because resource constraints are met on a per-sensor basis. Techniques to retrieve data about resource utilization are specific to the sensor network. A family for information collection policies using AutoSec is introduced in [46].

MiLAN (Middleware Linking Applications and Networks) [50] is a middleware solution permitting network applications to specify their QoS requirements. MiLAN adjusts the characteristics of the network over time so the QoS requirements of network applications can be satisfied. In addition to QoS requirements, MiLAN receives information about application priority, sensor availability and channel bandwidth that is used to make decisions that will affect the perceived QoS to applications requesting data.

Milan presents an API through which a high level application presents its requirements regarding sensors of interest that may be available. Network-level functionality is abstracted away through a sequence of commands available to determine all available information about sensors deployed in the network. These commands are specifically used to request or manipulate information regarding network data channels, remote network control and network-specific local network control. Data channel and remote network control functionality interact directly with the transport protocol of the network stack in question like TCP, UDP in an IP network or L2CAP in a Bluetooth network. Functionality for network-specific local network control is used to interact with protocols related to routing, media access control and the physical layer as much as possible in the network stack.

The work in [14] presents a middleware-based algorithm for configuring and managing bandwidth in a sensor network based on a P2P paradigm. A master Bandwidth Mediation Point (BMP) is established that owns all the available bandwidth except for a fraction allocated for signalling traffic.

If a node requests more bandwidth, a query is generated at the node and transmitted to all other nodes. The system is analogous to a file-sharing system exchanging bandwidth instead of files. Each node sends some bandwidth back to the requesting node according to availability or a predetermined network traffic distribution plan. The distribution of bandwidth throughout the network causes a shift in allocation from a client-server approach, where the BMP distributes bandwidth to all requesting nodes to a more hybrid approach

where the BMP, along with nodes directly connected to the BMP, distribute bandwidth through successive requests. As additional bandwidth requests are made, more bandwidth is distributed to nodes beyond the BMP and its closest nodes creating a peer-to-peer bandwidth distribution mechanism.

DCM-Arch [67] is an integrated architecture for handling data, control and management issues in scalable wireless sensor networks. The architecture is divided into four tiers of network infrastructure. The lowest tier describes the routing and network infrastructure used to transmit and receive data through the network when data is exchanged with other networked components. This tier is connected to a tier implementing functionality for data dissemination, query dissemination and code distribution as required. Components for these tasks are integrated into information reduction techniques used to minimize the data required for transmission through the network. The data, control and management functionality is implemented in a third tier and divided into separate planes while the highest tier is used by applications to provide an interface to end users.

The data plane is used to perform tasks related to information storage, reduction, and dissemination in wireless sensor networks. Wireless sensor networks are perceived to be databases and users interact with sensor networks by executing queries in a SQL type language. Using this model, data is written to the wireless sensor network using the data dissemination protocol in the tier below the data plane. Additionally, data is retrieved from the wireless sensor network using the query dissemination protocol from the same plane. The control plane deals with the lightweight sensor adaptation and reconfiguration middleware to cope with the dynamic changes of the environment. The management plane provides the context management functions, e.g. for collaboration and reasoning, to allow sensors, coordinators, and sinks to make good decisions about adaptation and reconfiguration.

2.3.4 Service Oriented Solutions

This category if sensor networking middleware solutions describe mechanisms to integrate or create distributed sensor networks using a service oriented approach. The work in [100] describes a gateway deployed on a network used to integrate users, sensors and services at different locations. The Java-based gateway is used to implement potentially several services

connecting clients to small network-attached sensor-actuator modules (SAMs). Clients may implement interfaces using Jini, Java applets or HTML to receive exchange data with one or many deployed services. SAMs are registered into the gateway and associated with a specific service. Each service contains a SAM proxy used to process queued requests from clients and exchange data with a SAM. Events are generated by the proxy so data can be transmitted to clients. Data is exchanged between clients and services using remote method invocation (RMI) or HTTP.

A layered and compartmentalized middleware architecture was developed for the Gator Tech Smart House [51] containing separate layers for physical devices, sensor platforms, services, knowledge, context management and applications. The physical layer consists of the various devices and appliances the occupants of a house would use like lamps, a television, smoke detectors, thermostats, etc. This layer is directly connected to a sensor platform layer abstracting physical devices into low level services implemented as Open Services Gateway Initiative (OSGi) services. The service layers implements an OSGi framework and high level composite services to exchange data with sensors connected through the sensor platform layer. Clients use the application layer to access the composite services of the service layer. The creation and management of context information is facilitated through a context management layer used to exchange data with the application layer and service layer as required. Service registration and discovery are available through a knowledge layer.

Some research implementing service oriented solutions for various problems describe mechanisms permitting extensions for sensor networks. For example, the network topology service grid described in [110] outlines a Network Information and Monitoring Service (NIMS) provides the means to manipulate network control plane resources like network topology, link capacity occupation and communication delay. The measurement methodologies can be applied for network sensors, defined as entities capable of producing data regarding network metrics.

Mires [102] provides an asynchronous communication model for wireless sensor networks which are event driven in most cases. Mires' architecture includes components for the publication of data, subscription of data, routing and additional features like data aggregation. The publish-subscribe service coordinates the communication between middleware services.

Sensor nodes advertise their sensed data through the publish service of Mires. Mires routes the advertised messages using a multi-hop routing algorithm. A user application is used to subscribe to topics of interest. The publish-subscribe service also maintains the topics list and the subscribed application to marshal the right topic to the related application. Mires sends only messages referring to subscribed topics to reduce network traffic and energy consumption. A data aggregation service permits users to aggregate data where possible.

Publish/subscribe based contributions have been made based on Mires including Mires++ [80], a clustering algorithm to create energy-aware clusters for in-network data processing. The clustering algorithm is considered a new service in Mires. The clustering algorithm considers location information to facilitate boundary control.

A message broker to facilitate integration in sensor networks is presented in [38]. This broker is implemented in Java and uses the MQTT protocol [56] to format messages. The communication system of the broker is a runtime environment for the execution of protocol stacks. The communication subsystem of the broker consists of a layered sequence of protocol modules.

The work in [40] describe a middleware solution for the dissemination of context information based on the Services Oriented Architecture (SOA) and implemented using XML Web Services technologies. The middleware provides the base functionalities for mobile devices to publish their context information, and in addition subscribe to the context information published by their mobile peers. The publish/subscribe middleware is connected to numerous web services where each web service is used monitor multiple sensors. Context information from each sensor is saved into a database that is accessed by the web service. Web services transmit information about the context described by the sensors through notifications when subscriptions have been created. Clients connect to the middleware through an application layer used to submit subscriptions for various contexts.

The web services publish different methods so data may be exchanged with the middleware component of the architecture. The published methods subscribe and unsubscribe handle input and output messages. The Notification operation only sends out an output message. All operations are connected by a Simple Object Access Protocol (SOAP) port type to three bindings. The default binding is the Hypertext Transfer Protocol (HTTP), in

addition, a User Datagram Protocol (UDP), and a reliable UDP binding has been developed in order to improve the performance of SOAP in mobile networks.

A rule parser, rule evaluator are also required for the web service to perform its tasks. The subscribe and unsubscribe methods described above are used to add and delete new rules respectively. If the rule syntax is valid, the subscribe method returns an acknowledgment message to the context requester containing a unique rule identifier, otherwise an error message is transmitted. A valid rule will be saved into a list of rules. In a parallel process this list is evaluated by the rule evaluator. Each time the rule applies, a notification message with the event as parameter is send to a context requester. The prototype [41] was implemented using J2ME based on its widespread availability on mobile devices. Open source implementations of SOAP and XML that have been optimized for J2ME are used for the implementation of the prototype.

2.3.5 Dynamic and Mobile Code

Middleware architectures have been developed to support mobile code to reduce network load, facilitate parallel processing, create adaptable environments and enhance fault tolerance. The work in [9] presents a mobile code daemon based on the core network protocol Remote Executation and Action Protocol (REAP). REAP is used to pass data between nodes deployed on a network that may contain sensors. Each REAP message contains an instruction and payload used to execute a specific sequence of commands where it is received. Each instructions contains requests for code, resources, references and threads as required in addition to migration data.

Impala [71] is a middleware system and API for sensor application adaptivity and updates. It proposes a runtime system that acts as a lightweight event and device manager for each mobile wireless sensor node in the system. Impala has been built as part of the ZebraNet[62] effort, in which sensing nodes are placed on freeranging wildlife to perform long-term migration studies on a collection of animals in an ecosystem. Impala provides an interface for on-the-fly application adaptation in order to improve the performance, energy-efficiency, and reliability of the software system.

The highest layer of Impala contains all the application protocols and programs for Ze-

braNet. These applications use various mechanisms to collectively gather environmental information and route it to a centralized base using peer-to-peer transmission. One application is executing at a time. A lower layer is implemented with three middleware agents: an Application Adapter, the Application Updater, and the Event Filter. The Application Adapter is used to adapt the application protocols to different runtime conditions to improve performance, energy-efficiency and robustness. Software updates are propagated through the Application Updater to nodes where Impala components are installed. Event filters are used to capture that occurs in the system and pass this information to Application Adapters or the application layer as required. Multiple events are processed sequentially within a limited amount of time to limit performance bottlenecks. The following events are defined in Impala:

- Limited power and resources among sensors nodes. Timer events are generated when a signal is generated by a timer a specific period of time has elapsed. Timers may be owned by the active application, Application Adapter and Application Updaters.
- Packet events are generated when a network packet has arrived. Packets may be transmitted from application to application or from updater to updater. The intended packet recipient process the event.
- 'Send Done' events are generated when a network packet has been transmitted successfully or unsuccessfully. These events permit asynchronous data transmission and are processed by the entity transmitting data through the network.
- Data events are generated when a sensing device has data that is prepared to be read. These events are processed by the active application.
- Device events are generated when a device failure has been detected in Impala and they are processed by an Application Adapter.

Agilla [32] is a mobile agent middleware designed to support self-adaptive applications in wireless sensor networks. Agilla provides a programming model in which applications consist of evolving communities of agents that share a wireless sensor network. Coordination among the agents and access to physical resources are supported by tuple spaces (a type of shared

memory in which data is structured as tuples and accessed via pattern-matching). A tuple space in Agilla does not span multiple nodes to avoid the overhead of keeping it consistent in a dynamic environment and to ensure scalability. If a shared data abstraction spanning multiple nodes is necessary, it must be built on top of primitive system operations. Tuple spaces can be accessed by agents through local and remote operations, and is augmented with reactions that enable agents to efficiently respond to changes in the tuple space state. Nodes are referenced by their location. Agilla agents are written in an assembly like language.

The middle contains the core Agilla middleware components. The middleware consists of several components that are orchestrated by an Agilla engine, which is the Virtual Machine (VM) kernel that controls the concurrent execution of all agents on a node. It implements a round-robin scheduling policy. The Agilla engine also handles agent arrival and departure. An agent is divided into multiple instructions mapped into several messages for transmission between nodes. The bottom layer of Agilla is the operating system based on TinyOS[52].

Middleware for the dissemination of satellite images using software agents is described in [108]. A software prototype named Genie was developed based on the utilization of different types of agents representing different individuals or organizations. The architecture includes a ground system used to support customer order formulation, manages customer orders, tasks satellites to perform image collections, downloads satellite telemetry and imagery, maintain image archives, and deliver satellite images after the appropriate packaging and image processing has occurred. The ground system is divided into separate modules called the customer service center (CSC) for collecting and managing customer orders, the satellite tasking element (STE) to plan and schedule satellite operations and the communications and image processing module (CIP) for image retrieval, processing and archiving. The objective of the middleware was to permit users to select and search multiple distributed and heterogenous data sources in response to a single query received through the CSC. These data sources could be passive (i.e. image archive databases), active (taskable satellites) or independent services implemented to process data.

Each information processing component was implemented as a software agent. Data was exchanged between agents using the Knowledge Query and Manipulation Language [30]. Each KQML message indicates the requested action to be performed. Standard requests

include tell (tell another agent something), ask (ask another agent something), reply (reply to an ask message), achieve (ask another agent to achieve something), advertise (advertise the ability to provide information or services), and broker (ask an intermediary agent to find and task some other agent to handle a request).

Three categories of agents were implemented into the middleware. Provider agents are used to offer data or value added data-processing services on behalf of information providers. Consumer agents are used to assist define and meet their information requirements. Facilitator agents are used to help consumers agents and provider agents discover each other. The Genie prototype was implemented with provider agents for two passive data sources and one active satellite. Facilitator agents were used to act as a high level image search broker and an intermediary to match provider agents with a consumer agent implemented as an advanced graphical user interface.

MobEyes [68] is a middleware architecture designed for urban monitoring and exploits node mobility to diffuse sensed data summaries among neighbor vehicles and to create an index to query monitoring data. Data acquisition is performed through vehicular sensor networks, a collection of vehicles equipped with heterogeneous collections of sensors moving through an environment with mobility patterns due to street layouts, junctions, and speed limitations. MobEyes exploits wireless enabled vehicles that are equipped with video cameras and a variety of sensors to perform event sensing, processing/classification of sensed data, and intervehicle ad hoc message routing. Sensed data is kept in each vehicle using mobile storage. Onboard processing features are used to extract features of interest. Mobile nodes periodically generate data summaries with extracted features and context information such as timestamps and positioning coordinates, whereas mobile agents (e.g., police patrolling cars) move and opportunistically harvest summaries as needed from neighbor vehicles. MobEyes adopts VSN custom designed protocols for summary diffusion/harvesting that exploits intrinsic vehicle mobility and simple single-hop intervehicle communications. In this manner, MobEyes harvesting agents can create a low-cost opportunistic index to query the distributed sensed data storage.

MobEyes was implemented according to a component-based architecture. The MobEyes Sensor Interface (MSI) is a component used to retrieve information directly from a collection

of sensors deployed on a vehicle. Raw sensor data is retrieved from the MSI using a higher level component described as the MobEyes Data Processor (MDP). This sensor data is turned into chunks which include metadata (i.e. vehicle identification information, timestamps, etc.) and identifiable vehicle features of interest. The highest level component of a sensor node is the MobEyes Difussion/Harvesting Processor (MDHP) which selectively retrieves and disseminates data summaries produced by the MDP to external components. The main goal of the MDHP process is to create a highly distributed and scalable index that allows police agents to place queries into the huge urban monitoring database without attempting to combine this index in a centralized location. The MobEyes prorotype is build on the Java Standard Edition virtual machine. Additional details about the implementation of the MobEyes prototype are provided in [6].

2.3.6 Application Specific Middleware

The multitude of data types that can be processed from sensor networks easily permits the development of data specific middleware that permits the dissemination and processing of a limited set of data types. Limiting the data types of consideration allows specific client requirements to be fulfilled so long as they are related to the data supported by the system. One such middleware architecture is described in [15] where a context-aware middleware architecture was developed to control home appliances. The context-aware middleware uses OSGi(Open Service Gateway Initial) as the framework of the home network. A context-aware agent is used to read six values (pulse, body temperature, facial expression, room temperature, time and location) to create a context that is used to determine the preferences of a user and adjust the conditions of the environment accordingly.

The NavMote [29] is an assisted pedestrian dead reckoning system incorporating middleware to facilitate communication with a collection of sensors. The NavMote gathers information about pedestrian motion from an integrated magnetic compass and accelerometers. This data is compressed and transferred from a sensor network to an information center where the data are processed into an estimate of the pedestrian trajectory based on a dead reckoning algorithm. The middleware layer of the NavMote system contains five components: A component to determine if the NavMote is in the vicinity of a sensor network to transmit

the data, a backbone containing a spanning tree to facilitate efficient data transport through the network, clock synchronization, localization and network telemetry.

EgoSpaces [63] describes agent-based middleware to facilitate the development of context-aware applications. This middleware connections an application layer, through which clients receive data, to a network abstraction layer. Support packages were developed for EgoSpaces to perform functions related to network discovery, monitoring environmental conditions and defining network metrics. The network discover package is used by a host to discover information about its neighboring nodes and to keep that information current at all times. Nodes periodically transmit small messages to neighboring nodes to retrieve information about the network. CONSUL [43] was developed and used in EgoSpaces as a general purpose sensor network registry to retrieve data about the environment for context description. The Source Initiated Context Construction (SICC) [93] was developed and used as the protocol to construct a subnet of an ad hoc network based on network properties.

The Common System for Middleware of Sensor Networks (COSMOS) [66] is a middleware solution to connect sensor deployed on various types of networks (CDMA, RFID, IP) to high level applications for ubiquitous sensor networks. The middleware is divided into three main layers connecting user applications to the sensor networks. A service layer acts as an interface to the middleware for user applications. This layer provides basic facilities related to security and sensor location. An intelligence layer receives information from the service layer and is used to perform tasks related to sensor data mining, event management and the integration of information from various sources. An abstraction layer is the direct interface to the sensor networks and connects directly to the intelligence layer. This layer is used extract data from sensors deployed across various networks and perform tasks related to sensor management.

DeftRFID [73] is a scalable middleware solution used to connect a sequence of hardware devices (i.e. sensors, RFID readers, etc.) to various applications. This architecture is divided into three layers connected through TCP sockets. A hardware abstraction layer provides direct connectivity to the sequence of RFID readers deployed in the middleware. This layer provides services for device management (adding or removing devices by USB, serial port or Ethernet), low level functions (i.e. device activation and deactivation, reading tag data, writing tag data) and the removal of duplicates. A data processing layer is connected to the

hardware abstraction layer and is used to provide a number of services like data aggregation, data transformation, data filtering, data dissemination, data storage, data querying and order transmission. These functions collectively interact with a rule management component that retrieves data from a database to enforce an arbitrary set of policies provided by an end user. An application interface layer is connected to the data processing layer and is used by applications to forward user requests and user defined rules to the rest of the middleware. Data retrieved through the middleware is presented to end users through this layer.

GridStat [42] is a middleware architecture designed to implement flexible and secure data communications for the operation of a power grid. Network management and data delivery are performed through separate domains described as the management plane and the data plane respectively.

The management plane allocates resources and adapts the network in reaction to changing power system configurations or communication network failures. The active components of the management plane are called QoS brokers and those of the data plane are called status routers. The QoS brokers are hierarchically arranged with policies set at higher levels in the hierarchy controlling more global aspects and allowing local concerns to be implemented using policies at lower levels. The hierarchical organization was designed to support decomposition of communication management and cyber-security policies along organizational and geographic boundaries.

2.3.7 Multimedia

Sensor Enhanced Video Annotation (SEVA) [72] is a digital recording system used to record the identities and locations of objects (as advertised by their sensors) along with visual images (as recorded by a camera). The process combines a series of correlation, interpolation, and extrapolation techniques and produces a tagged stream that later can be used to efficiently search for videos or frames containing particular objects or individuals.

A number of assumptions are made with respect to the implementation of the system used to perform video annotation. For the sake of simplicity it is assumed that a homogeneous sensor environment is used to collect data using a recorder with a single wireless radio. It is also assumed that the identify and location of every sensor is reported with every query.

Stationary objects can used hard coded locations. Mobile objects are presumed to contain sensors that will report an accurate in location in real-time using a positioning system like GPS. Passive sensors are presumed to store their current coordinates and transmit them upon request.

A prototype of SEVA was implemented using a Sony Motion Eye Web camera connected to a Vaio laptop. The location and identity querying, correlation, extrapolation and prediction, filtering and elimination, and database storage software runs on the laptop. SEVA currently uses two 3D locationing systems for the camera and objects: GPS and the Cricket Ultrasound location system [101]. The orientation of the camera is obtained through a digital compass attached to the laptop that provides the orientation (heading, pitch, and roll) of the lens of the camera.

AuViM [105] is a cluster-based middleware solution for audio and video sensor networks. This middleware solution permits the dissemination of audio and video data across a network while satisfying the QoS requirements of the user requesting data. AuViM is divided into four main components: a user interface, sink node, cluster head and a collection of audio and video sensors. The user interface permits individuals to communicate with the network of sensors using three interfaces: a query interface to make requests, a weight setting interface to specify QoS requirements and a data acquisition interface to receive data.

Cluster heads are directly connected to sink nodes and provide an interface to a collection of audio and video sensors deployed in the network. These components are divided into numerous modules performing the following tasks.

- The reception and interpretation of QoS requirements.
- Analysis and fusion of data received from audio and video nodes.
- Node scheduling and coordination.
- Generation of node configuration information (node selection, determining sampling frequency, determining image quality, etc.) based on user requests.
- Gathering information regarding node power consumption.

2.3.8 Security and Reliability

The framework presented in [37] discusses a generalized approach to provide information about the accuracy and integrity of data disseminated by sensors in a sensor network. A community of trust is permitted to developed for every sensor node in the network. Information regarding the reputation of each sensor node is maintained in other nodes deployed throughout the network. Reputation metrics are updated in other sensor nodes through continuous monitoring. The behavior of each node is rated as being cooperative (expected behavior of the nodes in the network) or noncooperative (unexpected behavior that is most likely the result of a system fault or node compromise). Then the node uses this reputation to evaluate the trustworthiness of other nodes and the data they provide.

The framework was implemented in software as a middleware service and supported by operating systems like SOS[47]. The Watchdog module maintains a library of outlier detection protocols and provides asynchronous interfaces so data can be exchanged with applications. The calling application specifies both its identity and the asynchronous interface on which it expects a reply from the Watchdog module. The calling application is also responsible for providing the requisite parameters for the corresponding outlier detection mechanism. The reputation systems provides asynchronous interfaces for reputation initialization, updates, and network parameter updates. Packets are periodically transmitted from the reputation system that contain the reputation information of nodes in the neighborhood. The period is equal to the integration period as specified by the applications. The reputation system also provides a synchronous function call that returns the trust metric of a given node.

2.3.9 Multicast

The middleware data dissemination architecture presented in [16] and [17]. This middleware acts as an intermediary between an application layer and a transportation layer. The application layer acts as an interface to the end user while the transportation layer is used to send and receive data through a network using IP multicast [23], single-source multicast [53], reliable multicast [48] or end-to-end multicast [18]. The middleware was designed for

the dissemination of documents across a network to multiple users while efficiently address issues related to data selection (the appropriate dissemination method for each document), scheduling (the frequency and order in which documents should be multicast), consistency (support for concurrency and consistency when documents are updated), cache replacement (management of client-side caches) and indexing (the use of indices to reduce waiting time for documents). Various methods for document selection are discussed in [111].

The health care alert system Real-time Outbreak and Disease Surveillance (RODS) ([70], [107]) was implemented based on this data dissemination middleware. In RODS, the server can disseminate data by choosing any combination of the following three schemes: multicast push, multicast pull, and unicast pull. In multicast push the server repeatedly sends information to the clients without explicit client requests. Multicast push is an ideal fit for asymmetric communication links, such as satellites and base station methods, where there is little or no bandwidth from the client to the server. For the same reason, multicast push is also ideal to achieve maximal scalability of Internet hot spots. In multicast pull, the clients make explicit requests for resources, and the server broadcasts the responses to all members of the multicast group. If multiple clients request the same resource at approximately the same time, the server may aggregate these requests, and only broadcast the resource once.

2.4 Conclusion

Data dissemination middleware describe solutions available to transmit a subset of available data from a collection of data producers to data consumers. Sensor networks represent one popular domain in which data dissemination solutions are applied to transmit data generated by sensor in real-time to entities with access to this data. Publish/subscribe architectures are sometimes used to facilitate the transmission of data between data consumers and producers by using a broker as an intermediary through which consumers create subscriptions for data and producers transmit data through notifications. The service oriented architecture permits services to discover each other through a registry and communicate directly with each other using SOAP and WSDL. Data dissemination solutions have been applied to numerous areas using a wide variety of techniques to address issues related to the transmission of data from

multiple sources to multiple destinations.

Data dissemination solutions designed for mobile clients are intended to transmit data to clients using wireless protocols. IMS based solutions are specifically intended to disseminate data to users through cellular networks while non-IMS based solutions may transmit data to clients connected to an IP network using WiFi, Bluetooth or another protocol with no scope beyond an IP network.

Several other middleware solutions have been described in the literature addressing concerns related to security, network control, and applications specific situations. These solutions use a wide variety of architectures and techniques including the publish/subscribe approach, service oriented architecture, mobile agents, and multicast data transmission for different types of data including multimedia. Furthermore the majority of these solutions are implemented in software and deployed on various devices. In many cases the software implementations cannot be deployed in embedded communications devices because the development frameworks cannot be used outside of a specific environment. In other cases, the software implementations would run inefficiently on an embedded device because they are not optimized for such operational environments. A hardware architecture for data dissemination would address these concerns.

While there is a significant overlap in the techniques and objectives described with data dissemination literature, the following collection of issues are not known to be addressed in existing middleware solutions:

- The dissemination of data from data producers to data consumers in mobile networks.
- The definition and transmission of heterogeneous data for dissemination between data producers and data consumers.
- The dissemination of information regarding the change in availability of data sources in real-time.
- Permitting mobile data consumers to define how data may be disseminated.
- The implementation of middleware functionality on mobile devices.

The remainder of this thesis describes an architecture to address these issues.

Chapter 3

High Level System Description

This section provides a high level description of the data dissemination architecture proposed in this thesis. The first section formally defines all the requirements of the architecture. Specifications are described in the following section to list relevant limitations and constraints. Architecture components are introduced to identify the necessary roles and interactions to fulfill the architecture requirements. Subsequent sections define the IP Multimedia Subsystem (IMS) and Session Initiation Protocol (SIP) as the communications framework that provide the basis for the implementation of the architecture.

3.1 Communications Architecture Definition

3.1.1 Requirements

The architecture provides the means for individuals to define heterogeneous data in real-time, register their data in the network and transmit their data upon request through an IMS network. The following high level requirements have been defined for the architecture:

- The architecture will permit end users to define heterogeneous data using arbitrary data formats.
- The architecture will permit the removal of previously defined heterogeneous data.

- The architecture will permit the registration of heterogeneous data sources into the network.
- The architecture will permit the deregistration of heterogeneous data from the network.
- The architecture will permit users to request a subset of heterogeneous data registered in the network.
- The architecture will permit users to automatically receive updates when heterogeneous data sources have been registered or deregistered from the network.
- The architecture will allow users to request the synchronous transmission of any subset of heterogeneous data registered in the network.
- The architecture will allow users to request the asynchronous transmission of any subset of heterogeneous data registered in the network.
- The architecture will permit end users to specify the rate and frequency of asynchronously disseminated data.

3.1.2 Specifications

The architecture described in this thesis is subject to the requirements listed in the previous section and the specifications below:

- The architecture must permit the dissemination of heterogeneous data through an IMS network using SIP.
- SIP functionality must be implemented using an embedded system to facilitate acceptable real-time performance of the architecture where mobile devices can be used for data consumption.
- High level data processing and interaction with end users may be performed using software.

3.1.3 Component Description

The following components have been defined for the architecture:

- Data Consumer: This component is required by end users to request information about available heterogeneous data. All data consumers contain the following features:
  - An embedded system implementation of SIP to exchange data through an IMS network.
  - The functionality to request heterogeneous data.
  - The functionality to subscribe to updates regarding the registration and deregistration of heterogeneous data available in the network.
  - The functionality to synchronously request any subset of available heterogeneous data.
  - The functionality to asynchronously request any subset of available heterogeneous data.
  - The functionality to control the rate and frequency of asynchronously transmitted heterogeneous data.
  - The functionality to terminate an existing stream of asynchronously transmitted heterogeneous data.
- Data Producer: This component is required by end users to define, register and deregister heterogeneous data and generate data upon request. All data producers contain the following features:
  - An embedded system implementation of SIP to exchange data through an IMS network.
  - The functionality to define heterogeneous data.
  - The functionality to register heterogeneous data in the network.
  - The functionality to deregister heterogeneous data from the network.

  - The functionality to generate and transmit heterogeneous data registered in the network.

- Broker: This component acts as an intermediary between a collection of data consumers and data producers. The broker will contain the following features:
  - An embedded system implementation of SIP to exchange data through an IMS network.
  - The functionality to save all information regarding available heterogeneous data from data producers.
  - The functionality to fulfill the request of any data consumer for information regarding heterogeneous data registered in the network.
  - The functionality to create and maintain subscriptions for updates regarding registered heterogeneous data.
  - The functionality to fulfill a synchronous request for registered heterogeneous data from any data consumer.
  - The functionality to fulfill an asynchronous request for registered heterogeneous data from any data consumer.
  - The functionality to terminate a stream of asynchronously transmitted heterogeneous data.

- Reconfigurable Multimedia Collaborative System (RMCS): This component is used to process data formatted in the Session Initiation Protocol (SIP). A collection of these interfaces can be used to exchange data while simultaneously utilizing the functionality available in an IMS network. The RMCS contains the following features:
  - The functionality to parse SIP messages received from a transportation network.
  - The functionality to automatically generate and transmit SIP based responses as required for all messages received through a transportation network.
  - The functionality to initiate SIP requests based on input from an external client.

- The functionality to terminate an existing SIP request based on input from an external client.
- The functionality to save, update and terminate SIP sessions based on SIP data received through a transportation network.
- The functionality to automatically transmit information about the status of SIP sessions to an external client.
- The functionality to automatically transmit information received from end users through SIP to an external client.

3.1.4 Component Interaction

The components are connected in a manner that combines the Observer [36] design pattern and the Share the Load [92] design pattern. The Observer design pattern defines a one-to-many dependency between various objects so the changes in the state of an object can be transmitted to all other objects. The Share the Load pattern is used in communications networks to increase the processing power available in a network by ensuring that requests do not exceed the total processing capacity. This objective is achieved by shifting processing functionality into different processors.

Fig. 3.1 illustrates the application of the components described in the previous section to the aforementioned design pattern to describe the communications architecture.

An IMS core (or a collection of connected IMS cores) serves as the lowest level transportation medium through which data is exchanged. An RMCS is required for every entity connected to IMS because SIP is the primary communications protocol of IMS. The collection of RMCS and IMS cores form a SIP transportation network through which data can be exchanged between data consumers, data producers and a broker.

Data consumers, data producers and the broker use a protocol implemented with SIP to perform the high level tasks described earlier in this section. Information received from the RMCS about the status of SIP sessions and the arrival of new data is formatted in this data protocol and processed by all high level entities. The protocol is used primarily by data consumers to initiate data requests and interpret their responses. Data generated by a data

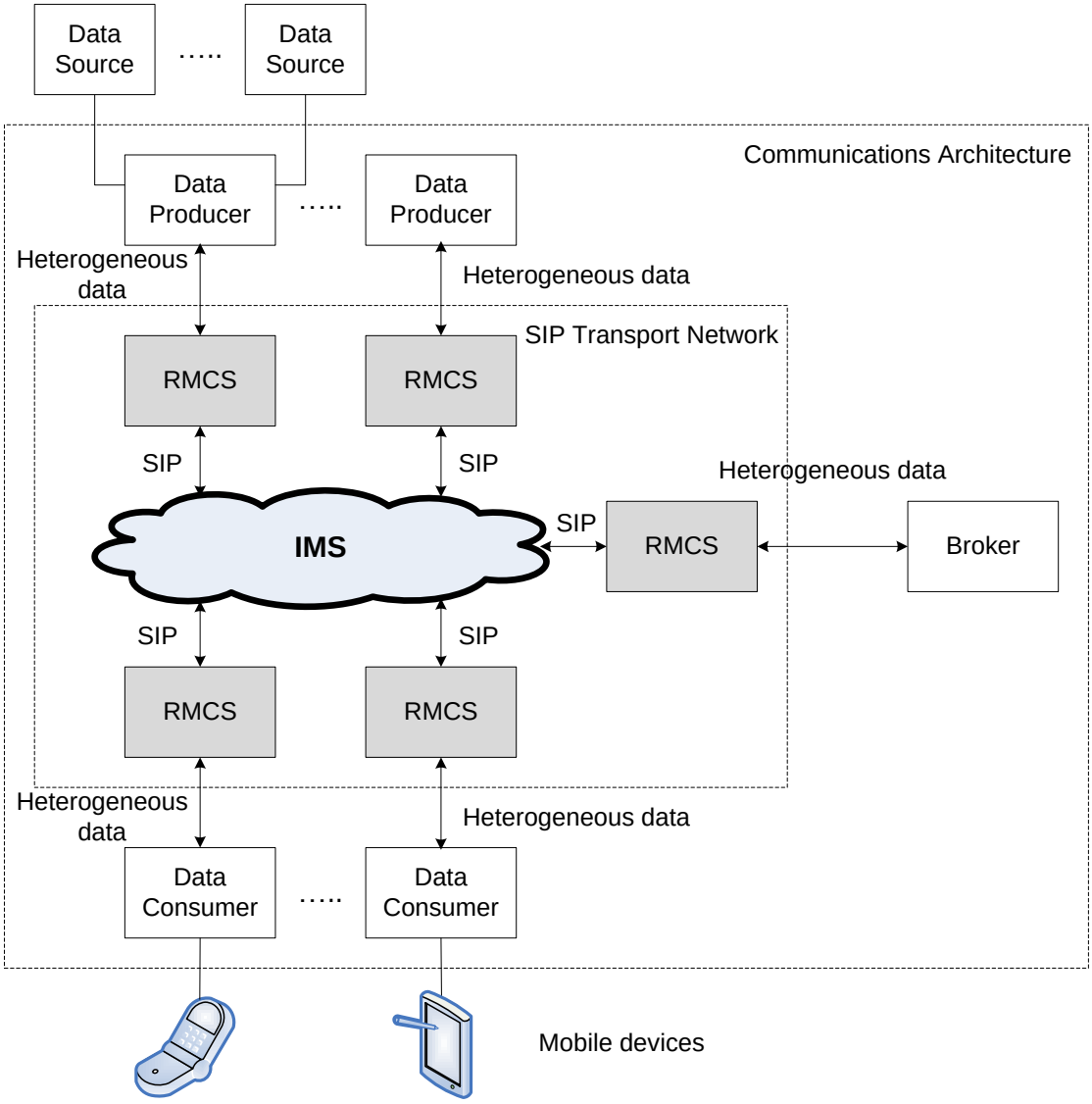

Figure 3.1: Communications Architecture Component Interaction.

producer is formatted in this protocol and transmitted through the RMCS to the network. Tasks fulfilled by the broker to initiate requests and manage subscriptions on behalf of data consumers and manage registered data sources are all initiated by requests received through the data protocol.

3.2 IP Multimedia Subsystem

The first subsection defines IMS and explains its importance in establishing connections between end users across divergent networks. The second subsection describes the core of IMS in significant detail. Each component of the IMS core is described to illustrate its purpose in providing end-to-end connectivity between various users and network services. The final subsection describes features provided to users through IMS. While numerous services are available this subsection will only describe the services most important to exchange heterogeneous data through IMS.

3.2.1 Definition

As stated in section 1.2, the IP multimedia subsystem (IMS) is an architecture that converges networks, services and devices into a unified communications platform. A more formal definition of IMS is provided below from [90]:

IMS is a global, access-independent and standard-based IP connectivity and service control architecture that enables various types of multimedia services to end-users using common Internet-based protocols.

Fig. 3.2 provides an illustration of IMS to illustrate how the convergence of networks and services is performed. End users communicate with each other with the facilities available through their divergent network. For example, individuals using desktop computers or laptops are connected to each other through Internet Service Providers (ISPs). Telephone users are connected through Public Switched Telephone Networks (PSTN). Numerous wireless networks provided connectivity to cell phone and mobile device users through a multitude of technologies like the Global System for Mobile Communications (GSM) [57] and Enhanced Data Rates for GSM Evolution (EDGE)[106]. Communication would ordinarily be limited to individuals who are connected on the same network. The IMS core provides the functionality for users on divergent networks to establish communications sessions with each other using IP connectivity.

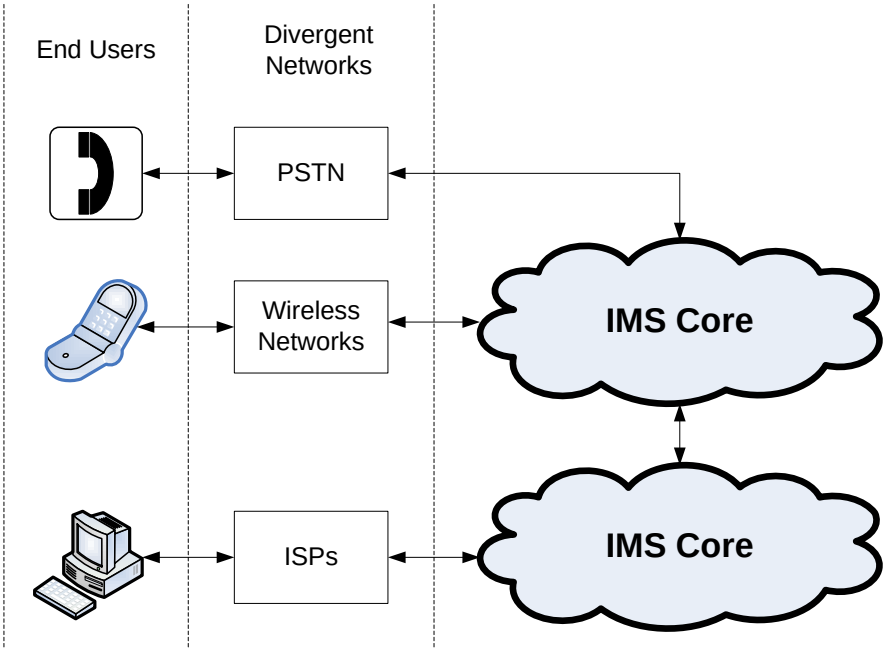

Figure 3.2: Description of IMS.

3.2.2 Core Description

The IMS core is primarily composed of Call Session Control Functions (CSCFs) and a database described as the Home Subscriber Server (HSS). CSCFs are servers deployed in the IMS core that perform specific tasks. The HSS is used to store subscriber and service related information for retrieval by CSCFs and application servers. Fig. 3.3 illustrates how the different components of the IMS core interact with divergent networks and application servers. The subsections below provide additional information about the functionality of all CSCFs and the HSS.

Proxy Call Session Control Functions

The Proxy Call Session Control Function (P-CSCF) acts like an interface to the IMS core for end users through their divergent networks. All data transferred to the IMS core from an end user must pass through the P-CSCF. All data intended for an end user from an application server or another end user must be delivered through a P-CSCF. Tasks assigned to the P-CSCF include data compression, network security and policy enforcement as required.

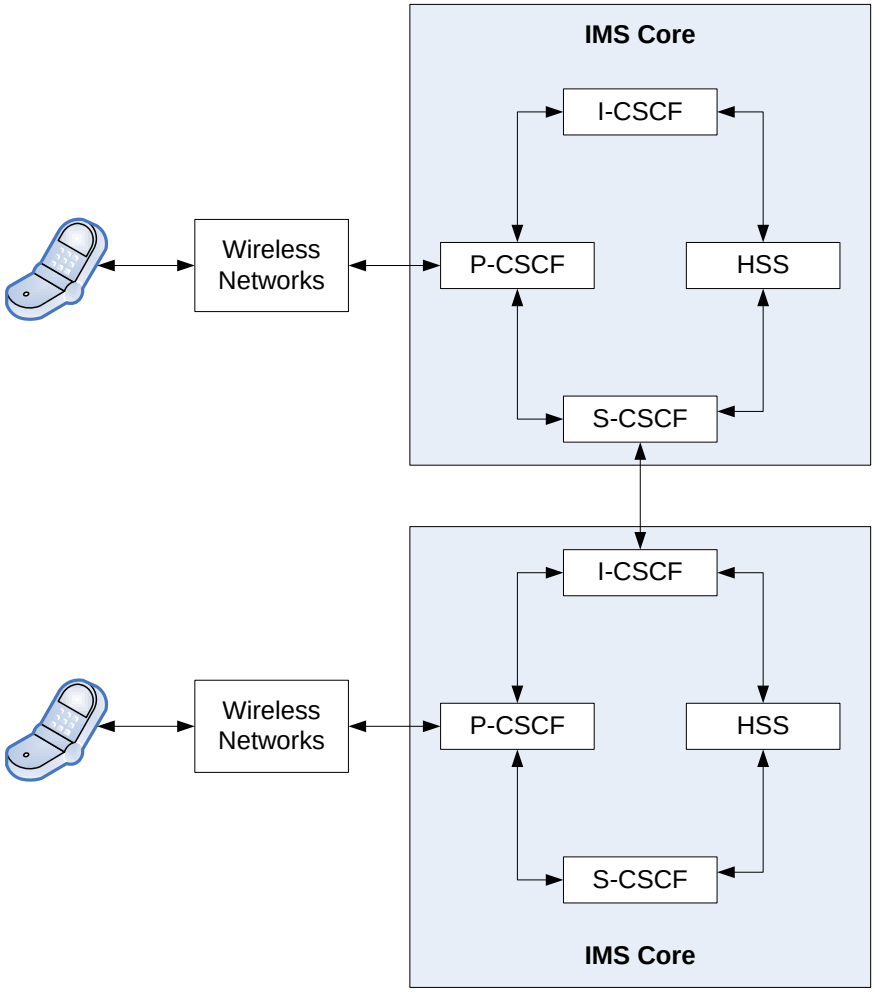

Figure 3.3: IMS Core.

Security associations (SAs) are negotiated between the end user and P-CSCF before data is transmitted through the IMS core.

Interrogating Call Session Control Functions

IMS networks may be divided into several administrative domains with each domain requiring its own proxy. Interrogating Call Session Control Functions (I-CSCF) are located at the edge of an administrative domain to act as an interface for other IMS domains. I-CSCF addresses are located in Domain Name System (DNS) [75] servers so a DNS lookup must be performed at the P-CSCF to retrieve the address of the I-CSCF where data will be forwarded. The next destination for data is determined by the I-CSCF by querying the HSS. A serving call

session control function (see the section below) is assigned based on the results of the HSS query. Data may be routed to a serving call session control function or application server from an I-CSCF.

Serving Call Session Control Functions

Serving Call Session Control Functions (S-CSCF) are used to maintain a relationship between the current location of a user and their universal identifiers. S-CSCF nodes also provide SIP routing functions to find a desired user and enforces network administrator policies. The majority of critical IMS core functionality is implemented into S-CSCFs as it is the only CSCF that does not act as an interface for end users or administrative domains. Registration functions, routing decisions, and updating information in the HSS are all performed from the S-CSCF. S-CSCF data is exchanged with the P-CSCF (i.e. registration requests from end users), the HSS (modifying user profiles) and application servers (forwarding requests and retrieving responses).

Home Subscriber Server

The Home Subscriber Server (HSS) is a database used to store information about users registered in the IMS network and all supported application servers. User information is stored in user profiles which contain private information (assigned by the home network operator for tasks like authorization and registration) and public information required for requesting communication with end users. User requirements are also stored for each S-CSCF and they are used by I-CSCFs to determine the appropriate S-CSCF upon request. Information for all application services are stored and queried through S-CSCFs. The HSS uses the Diameter [11] protocol.

A Subscription Locator Function (SLF) is another database supported by IMS. The SLF enables a SIP application server, I-CSCF and S-CSCF to find the address of a specific HSS when many HSS have been deployed.

3.2.3 Features

This section summarizes the features available in IMS necessary to successfully disseminate heterogeneous data in real-time. The mechanisms to identify users are called user identities and they are summarized in the User Identification subsection. Procedures adding user identities to an IMS network are described in the registration subsection. The session initiation feature of IMS is described in the final subsection.

User Identification

Several identifiers are used to isolate specific information about a user before data can be exchanged. The identify of a user is generally referred to as a public user identity. Public identities can be published using web pages, business cards or any mechanism used to distribute contact information for an individual. They must be formatted as Session Initiation Protocol (SIP) Uniform Resource Identifiers (URI) or telephone Uniform Resource Locators (URL). Public identities must be registered before they are used and are not authenticated during registration.

Private user identities are unique, global and defined by the operator of the home network. Private identities are described as Network Access Identifiers (NAI) [22], must be stored in the HSS and must be contained in every registration request. The primary purpose of the private identity is user authentication, which only occurs during registration tasks.

Registration

Fig. 3.4 illustrates the procedure used to store user identities in an IMS network through registration. The procedure may require two separate phases requiring the same sequence of steps. A registration request is transmitted from an end user to the P-CSCF and forwarded to the I-CSCF. An S-CSCF is assigned to the user if necessary and authentication is performed. If authentication is not successful a response is returned to the end user indicating that the registration has failed and no user identity has been added to the IMS network. Upon successful registration a public profile is added to the HSS.

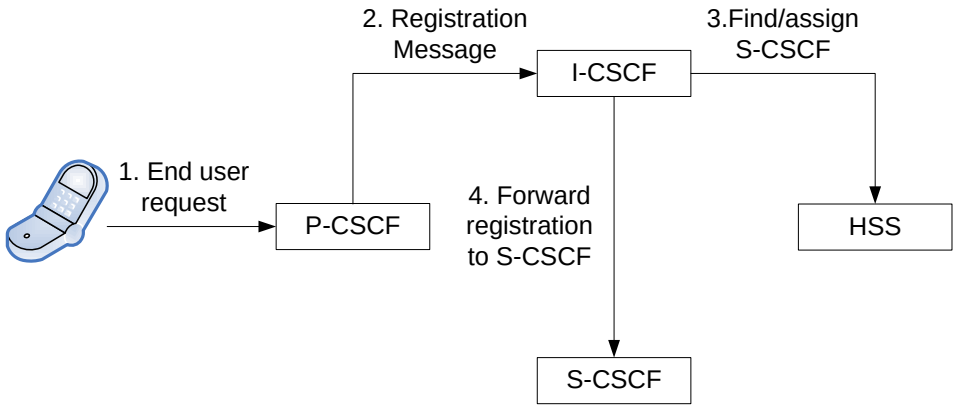

Figure 3.4: IMS Registration.

Session Initiation

The procedure used by an end user to establish a communications session with another end user is described in Fig. 3.5. A session initiation requestion is transmitted from the end user to the S-CSCF of its home network through the P-CSCF. If the end user is connected to another IMS network then the request is forwarded through other I-CSCFs until it reaches the correct home network. Each IMS network queries the user identities stored in the HSS to determine if the desired user is connected. Once the user identity of the destination end user has been identified the request is transmitted from I-CSCF to the desired party. The response generated from the destination end user is transmitted from its IMS network to the original IMS network using the reverse path.

3.2.4 Group Management

IMS provides a service that allow users to efficiently exchange data with multiple end users by organizing collections of users into groups. Group management refers to the facilities available in IMS permitting end users to store data on an intermediate server for transmission to others who request the data. By using an intermediate device to store data in the network, a single piece of data requested by multiple end users is efficiently disseminated through an IMS core when requests are made to the server for a group of end users. The group of users who are potential sources of data for transmission are collectively called a resource

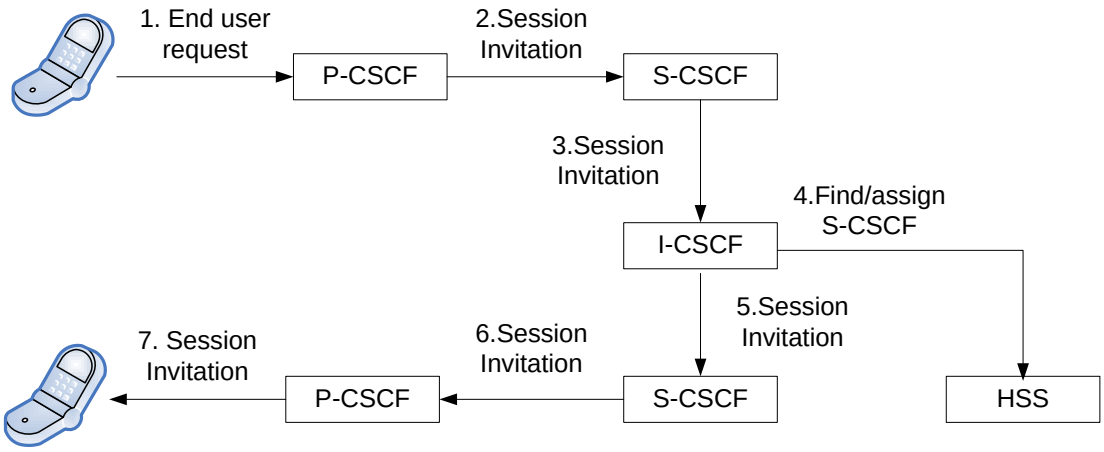

Figure 3.5: IMS Session Initiation.

list. The server used as an intermediary between a user requesting data and the resource list is described as a Resource List Server (RLS). Information is uploaded to an RLS using the XML Configuration Access Protocol (XCAP) [97]. Using XCAP, end users can add, edit and delete data describing the resource list and other information required for the RLS to fulfill user requests. XCAP is transmitted over HTTP using HTTP PUT, GET and DELETE operations.

The exchange of data between an end user and a resource list is illustrated in 3.6. After the resource list has been successfully transmitted to the RLS a subscription is created by an end user requesting data from the entire group. This request is processed by the RLS and accepted through the creation of individual sessions to each member of the resource list. Once the subscription has been created, data generated by any member of the resource list is transmitted to the RLS and immediately forwarded to the end user.

The group management service of IMS is a suitable mechanism for the dissemination of data between groups of users under specific sets of circumstances. Data may be easily disseminated using a push mechanism where data that does not change frequently is transmitted from multiple sources to multiple destinations. There are no known mechanisms for end users to specify how data may be transmitted using a pull approach should data be updated very frequently and end users wish to control the rate of dissemination. Furthermore, end users must support XCAP and HTTP to define the group of users from whom data

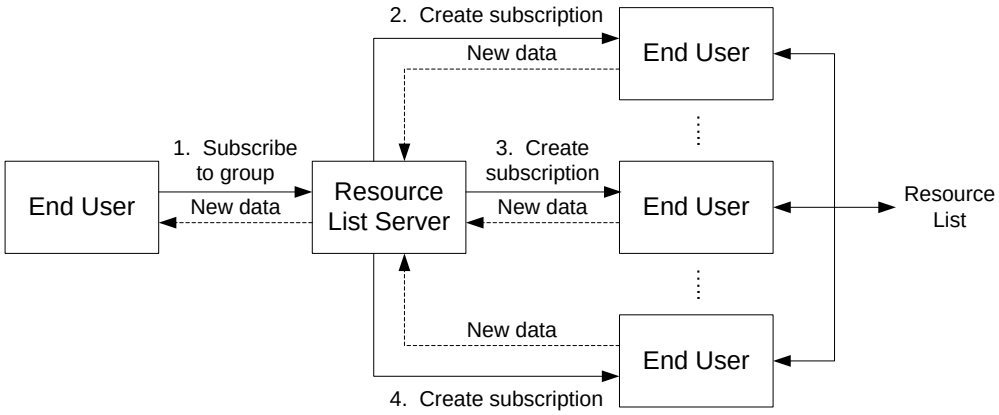

Figure 3.6: IMS Group Management.

should be transmitted. There are no known mechanisms to identify a subset of desired data for transmission from an end user. The resources in a resource list refer exclusively to end users. There are no facilities available to identify data sources.

3.3 Session Initiation Protocol

The following subsections provide a detailed overview of the Session Initiation Protocol (SIP). The first few subsections are used to formally define SIP, describe the end points used to exchange messages and describe the user identification mechanism. All the servers used with SIP are well defined before summarizing relevant SIP functionality including session management, subscriptions, notifications, instant messaging and mobility.

3.3.1 Definition

The Session Initiation Protocol (SIP) [98] is an application level protocol used to invite users to participate in multimedia sessions. It is similar to HTTP in its request-response model and its ASCII-based packets. Each line in the packet header contains a label and its corresponding information separated by a colon and space. Every line is terminated by a carriage return and line feed. The packet header is terminated by an additional carriage return and line feed before the payload begins. Examples of SIP packets are described in some of the following subsections.

3.3.2 End Points

A network end point enabled with SIP is called a User Agent (UA). UAs are used to establish, modify and terminate sessions with other agents. Users may be humans or machines implementing other protocols or functionality. SIP UAs contain state information necessary to distinguish among various sessions and two applications. The User Agent Client (UAC) describes all the functionality used to initiate a request by a SIP agent. The User Agent Server (UAS) is used by a SIP agent to generate a response. Multiple responses may be issued to a UAC in a client/server or peer-to-peer fashion.

3.3.3 User Identification

Uniform Resource Identifiers (URI) are used to identify two categories of entities in SIP: users and devices. A user URI is referred to as an Address of Record (AOR). AOR URIs are used in database queries, services and features. A device URI refers to a specific end user device (PC, cell phone, etc.) where an individual may initiate, modify or terminate a SIP session. Device URIs do not require database queries as they are bound to a user URI for a shorter period of time. User and device URIs are written in separate fields of a SIP header. Translating a SIP URI into an IP address is done through DNS.

3.3.4 Servers

SIP servers are logical network entities that provide services and features to user agents. They accept SIP requests and generate responses based on available functionality that is provided. Implementations may contain numerous functions or change functions under different conditions. Therefore numerous transport protocols must be supported like TCP, TLS [24] and UDP. The subsections below discuss commonly used SIP servers.

Proxy Server

A SIP proxy server receives requests from users agents or other SIP proxy servers and forwards them to the desired end user. SIP proxy servers do not initiate requests and have no multimedia functions. Only SIP header information is processed to determine its course

of action. A location service or database may be required to determine the IP address of the end user. If it is not possible to find the end user the proxy may send the request to another proxy if its address is known.

Redirect Server

Redirect servers are used to retrieve the location of a SIP UA without forwarding a message to the UA. This server is used to reduce the burden placed on SIP proxy servers by allowing SIP UAs to send requests directly. Only SIP headers are examined and no other functions are provided.

Registration Server

UAs must update their location when it changes to various services have access to the correct location at all times. Registration servers provide a service allowing UAS to register their current location so it is accessible to other SIP entities. Authentication is usually required to ensure that SIP sessions are not hijacked by unauthorized users.

3.3.5 Session Management

Fig. 3.7 illustrates the steps required to establish a session using SIP. The UAC transmits an INVITE packet to a prospective session participant that is either a UAC or a UAS. Session information is described in the INVITE packet payload using the Session Description Protocol (SDP) [49]. If the request is accepted then an OK message is transmitted back to the UAC. The UAC responds with an ACK (acknowledgment) message to confirm establishment of the session. Either entity may terminate the session by sending a BYE packet which must be acknowledged with an OK packet.

Fig. 3.8 provides an example of an INVITE packet with an SDP payload. Sender and receiver information are written in the 'To' and 'From' fields. The 'Via' field describes the lower level protocol used to transmit data (UDP or TCP) and the network path used to transmit the packet. Its contents are modified by every server processing the SIP packet in the network. The maximum number of times the packet can be forwarded is described with

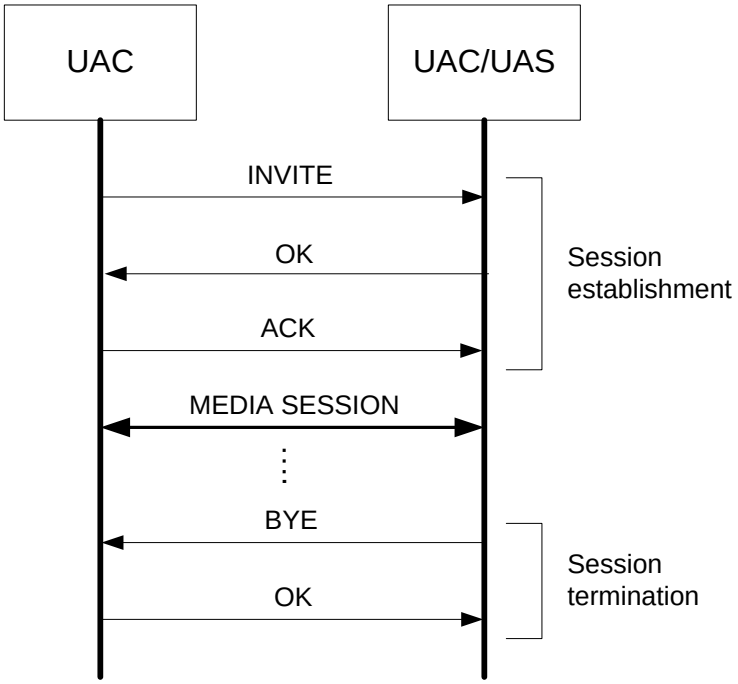

Figure 3.7: SIP Used to Establish and Terminate a Multimedia Session.

the 'Max Forwards' field. A unique identifier for the call is provided through the 'Call-ID' field. Separate INVITE requests are differentiated through the command sequence (Cseq) field. The length and type of the payload are described with the Content-Type and Content-Length fields respectively. Alternative user contacts can be specified through the 'Contact' field. An OK response to an INVITE contains the same SDP payload. The ACK, BYE and OK packets do not contain payloads.

3.3.6 Event Subscription and Notification

The functionality for a UAC to automatically receive information when events occur is provided through SUBSCRIBE and NOTIFY packets in SIP. Fig. 3.9 illustrates how subscriptions and notifications are performed using SIP. The example in 3.10 demonstrates the exchange of presence information as described in [94]. A SUBSCRIBE packet is transmitted by a subscribing UAC to receive notifications from another UAC. If the SUBSCRIBE request is accepted an OK message is transmitted back to the user. Every time the presence is updated a NOTIFY packet is transmitted to all UACs who successfully established

```
INVITE sip:fadi@uottawa.ca SIP/2.0
Via: SIP/2.0/UDP 100.101.102.103:5060;branch=z9hG4bKuotwa
Max-Forwards: 70
To: <sip:fadi@uottawa.ca>
From: <sip:ray@uottawa.ca>;tag=42
Call-ID: 195@100.110.120.130
CSeq: 1 INVITE
Contact: <sip:ray@uottawa.ca>
Content-Type: application/sdp
Content-Length: 159

v=0
o=schroed5244 2890844526 2890844526 IN IP4 100.101.102.103
s=Phone Call
t=0 0
c=IN IP4 100.101.102.103
m=audio 49170 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

Figure 3.8: An Example SIP INVITE Packet.

subscriptions. All NOTIFY packets must be acknowledged with an OK message.

Fig. 3.10 provides an example of a SUBSCRIBE packet used to initiate session notification. Many of its fields are similar to those with with the INVITE message from Fig. 3.8. The 'To', 'From', 'Via', 'Call-ID', 'Cseq', 'Max-Forwards', 'Contact' and 'Content-Length' fields have the same meaning and interpretation. The 'Allow', 'Allow-Events' and 'Event' fields are used to permit the transmission of presence information through subscriptions and indicate the appropriate packet types. No payloads are transmitted with SUBSCRIBE packets, however NOTIFY packets contains payloads describing presence information. Presence is described with Presence Information Data Format [104].

Multiple data formats are supported in the subscription framework of SIP describing different types of information. Subscriptions may be created to receiving notifications regarding call control for SIP conferences [61], the consent-related state of resources being added to a resource list [12], the state of a SIP dialog [99] and the monitoring of key presses using the Key Press Markup Language [10]. Additional types of data supported by SIP subscriptions include voicemail[74], user registrations [95], third party referencing [103] and

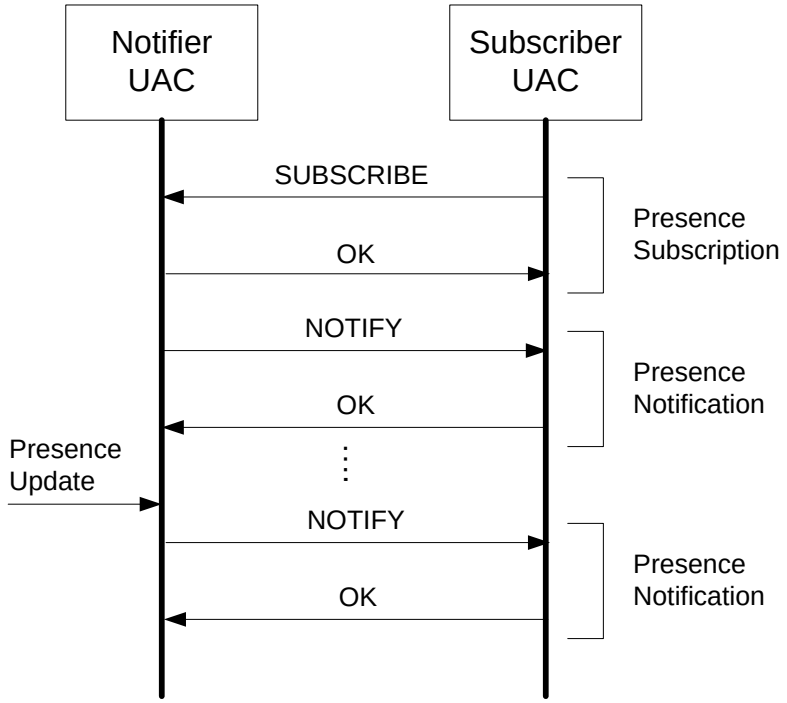

Figure 3.9: SIP Used to Perform Subscriptions and Notifications.

the set of users subscribed to a particular resource for a specific event package, and the state of their subscriptions (collectively referred to as watcher information) [96]. Other types of data will continue to be supported with SIP subscriptions. Data formats generally describe dissimilar information and are implemented using XML. The ability of end users to create subscriptions is entirely dependent on their ability to support the protocol describing the information, especially since payloads cannot be transmitted using SIP SUBSCRIBE packets. The 'event' field is used to list the type of information requested in the subscription with a predetermined data format. One method to allow the transmission of arbitrary information through SIP subscription is to permit the transmission of payloads with the subscription request so end users may describe the contents of their subscription in significant detail.

3.3.7 Instant Messaging

Synchronous data exchange across a network is commonly performed through instant messaging. The ability to exchange instant messages (IM) is available with SIP through an extension [13]. Fig. 3.11 demonstrates how instant messages are exchanged with SIP using

```
SUBSCRIBE sip:fadi@uottawa.ca SIP/2.0
Via: SIP/2.0/TCP domain.uottawa.ca:5162;
branch=z9hG4bKivlwn23
Max-Forwards: 70
To: <sip:fadi@uottawa.ca>
From: <sip:ray@uottawa.ca>;tag=21171
Call-ID: fnaw4nli3n203n29n42
CSeq: 3412 SUBSCRIBE
Allow-Events: presence
Allow: ACK, INVITE, CANCEL, BYE, NOTIFY, SUBSCRIBE,
MESSAGE
Contact: <sip:ray@uottawa.ca>
Event: presence
Content-Length: 0
```

Figure 3.10: An example SIP SUBSCRIBE packet.

MESSAGE packets. Its contents are a subset of the fields used in the INVITE packet described in Fig. 3.8. The 'Content-Type' field indicates that the payload is unformatted text which is interpreted as an instant message. Every MESSAGE packet must be acknowledged with an OK message. Participants may exchange IMs at any time with MESSAGE commands in the absence of a dialog. An example of a MESSAGE packet is shown in Fig. 3.12. No payload is included with an OK response.

3.3.8 Mobility

The SIP REGISTER packet is the mechanism required by end users to provide their current contact URI to the SIP network. This mechanism provides mobility to end users as an end user may register multiple times to reflect a change in their location. The contact URI acts as an IP address in the sense that is provides a specific location to an individual that may change at any time. An end user may utilize numerous end points (i.e. e-mail addresses, phone numbers, etc.) to exchange data. To identify the desired contact point for communication, an end user transmits a REGISTER packet with a contact URI to a SIP proxy server which forwards the packet to the Registrar server. The contact URI is saved in the registrar server and an acknowledgment is transmitted back to the end user. An example

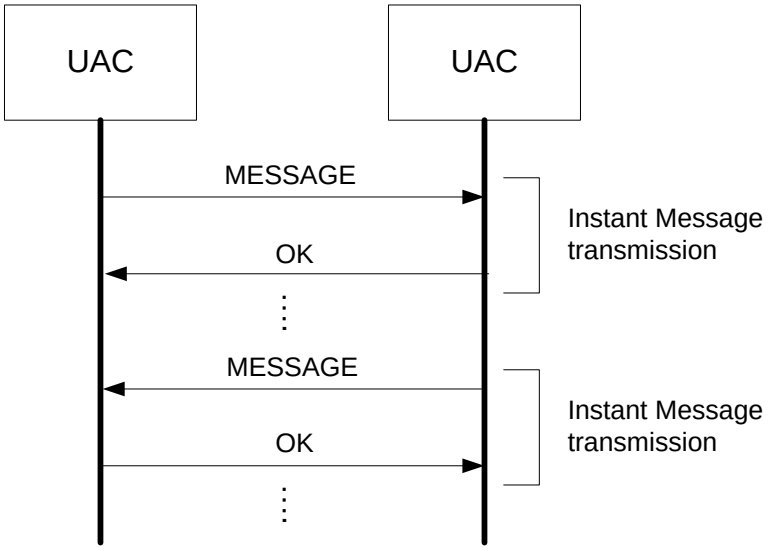

Figure 3.11: SIP Used to Exchange Instant Messages.

of a SIP REGISTER message is shown in Fig. 3.13.

3.4 Conclusion

This chapter provided a high level system description of the architecture presented in this thesis. The architecture provides the means for end users to define heterogeneous data and register their data to a network. The registration of heterogeneous data in the network facilitates end user requests for any subset of available data. Heterogeneous data may be transmitted through the architecture using synchronous requests through instant messaging or asynchronously transmitted data streams performed through subscriptions. The rate of asynchronous data dissemination may be specified by the data consumer upon request. Asynchronous heterogeneous data streams can be terminated by an end user any time. Data consumers, data producers and a broker are the collection of end users defined by the architecture to perform all tasks for requesting and generating heterogeneous data. All data must be exchanged using SIP so data may be successfully transmitted through an IMS network. SIP functionality is provided to all end users through an RMCS. The RMCS is implemented using an embedded system for optimal real-time performance and to facilitate

```
MESSAGE sip:fadi@uottawa.ca SIP/2.0
Via: SIP/2.0/TCP domain2.uottawa.ca;
branch=z9hG4bK71a2g5reh
Max-Forwards: 70
From: <sip:ray@uottawa.ca>;tag=49583
To: <sip:fadi@uottawa.ca>
Call-ID: 3l28f290v@110.120.130.140
CSeq: 1 MESSAGE
Content-Type: text/plain
Content-Length: 19

Hello, how are you?
```

Figure 3.12: An example SIP MESSAGE packet.

```
REGISTER sip:ray@uottawa.ca SIP/2.0
Via: SIP/2.0/UDP domain2.uottawa.ca;
branch=z9hG4bK71a2g5reh
Max-Forwards: 70
From: <sip:ray@uottawa.ca>;tag=49583
To: <sip:server1@uottawa.ca>
Call-ID: 3l28f290v@110.120.130.140
CSeq: 1 REGISTER
Contact: ray@mobile.uottawa.ca
Content-Length: 0
```

Figure 3.13: An example SIP REGISTER packet.

its integration into mobile devices.

Chapter 4

Data Dissemination Information

This chapter describes information relevant to the data dissemination performed through the communications architecture presented in this thesis. The first section describes the session management mechanism and illustrates how sessions are created and terminated to facilitate data dissemination. Subsequent sections define all the use cases required to fulfill the requirements introduced in the previous chapter. Each use case includes the necessary preconditions, necessary steps, post conditions, related use cases and any other information considered appropriate for a given scenario.

4.1 Session Management

Fig. 4.1 illustrates how session management is performed using the communications architecture. Abstractly speaking a session is a dialogue established between two entities to exchange data in a specific format. A session is established by one entity transmitting some initial message. Depending on the semantics of the protocol the session may be terminated immediately or prompt one or more responses before it is terminated. All data transmitted through the communications architecture is done in the context of a session.

The RMCS manages every session required to transmit data based on the request initiated by its corresponding application. All data transmitted from the high level protocol of the broker, data consumer and data producer is sent through a high level session. Therefore every high level session initiated by a communicating entity has a corresponding SIP session

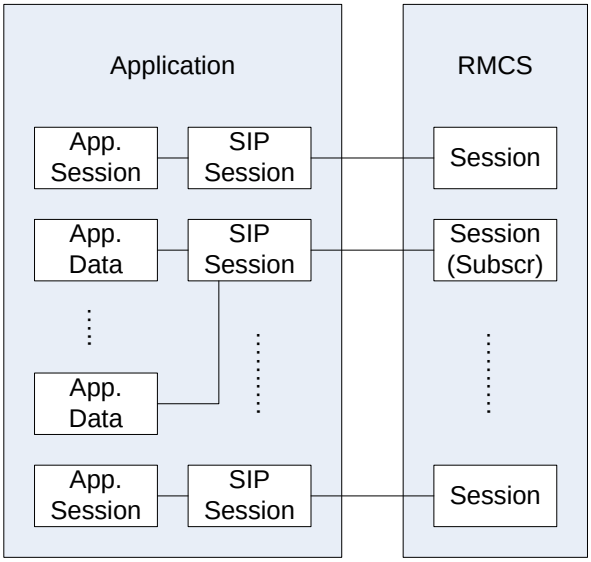

Figure 4.1: Session Management in the Communications Architecture.

required to transmit that data through the IMS network. The vast majority of high level communications sessions are terminated along with their SIP sessions at the same time. Subscriptions created by a communications entity are represented by a persistent session to transmit many pieces of data over an extended period of time.

4.2 Use Cases

This section describes all the scenarios supported by the architecture to fulfill the requirements described in the previous chapter subject to system specifications.

4.2.1 Successful Creation of a Data Source Description

This use case describes the scenario where an end user describes a data source and successfully adds the data source description to the data producer component. Data source descriptions must be present in a data producer before they can be registered in the broker.

Goals To add the description of a previously unknown data source to a data producer so it can be registered with a broker and ultimately used to transmit data through queries and data subscriptions.

Preconditions

- The data source in question must be available with an interface to read its data in real-time.

Related Use Cases

- Unsuccessful Creation of a Data Source Description.

Steps

1. The end user creates a description of a data source.
2. The end user transmits the data source description to the data producer.
3. The data producer alerts the end user that the data source description has been successfully received through an acknowledgment.

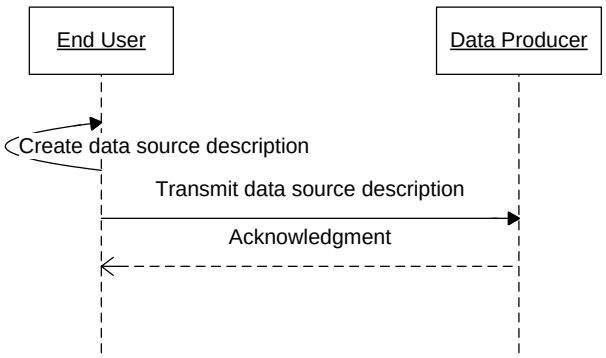

Figure 4.2: Message Exchange for the Successful Creation of a Data Source Description.

Postconditions

- The description of a previously unknown data source has been added to a data producer.

4.2.2 Unsuccessful Creation of a Data Source Description

This use case describes the scenario where an end user describes a data source and is unsuccessful in adding the data source description to the data producer. This scenario exist when

a user attempts to add a new data source description using a data source identifier that has already been used to create a data source.

Goals

To prevent an end user from adding multiple data sources to a data producer using the same identifier for more than one data source.

Preconditions

- At least one data source description must have already been successfully added to a data producer.

Related Use Cases

- Successful Creation of a Data Source Description.

Steps

1. The end user creates a description of a data source.
2. The end user transmits the data source description to the data producer.
3. The data producer alerts the end user that the data source description has not been successfully received through a negative acknowledgment.

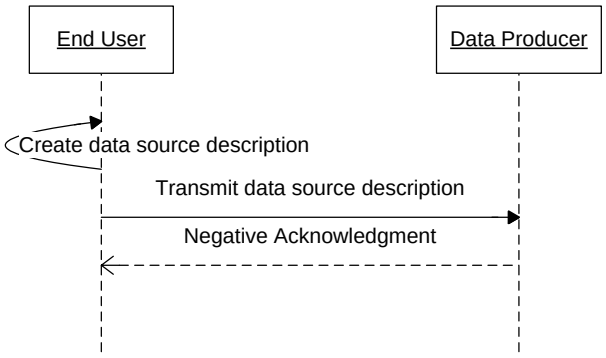

Figure 4.3: Message Exchange for the Unsuccessful Creation of a Data Source Description.

Postconditions

- No additional data source descriptions have been added to a data producer.

4.2.3 Registering Data Descriptions

This use case describes the scenario where the end user selects an arbitrary sequence of data sources that have been added to a data source and registers those data sources with the broker. All data consumers that have created subscriptions for information about data descriptions receive updates from the broker if any data descriptions are successfully added.

Goals

To permit the addition of new data sources to a broker by a data producer while ensuring that data consumers are informed of all data sources registered in the communications architecture.

Preconditions

- At least one data source description must have already been successfully added to a data producer.

Related Use Cases

- Deregistering Data Descriptions.

Steps

1. The end user selects the desired set of data descriptions for registration from the data producer.

2. The end user selects the option to register the selected set of data descriptions with the broker.

3. The request to register data descriptions is serialized and transmitted to the RMCS from its corresponding data producer.

4. The serialized data description registration request is converted into SIP and transmitted to the IMS core. Information about the SIP session established to transmit the information to the IMS core is transmitted to the data producer.

5. The serialized data description registration request is forwarded through the IMS core to the RMCS of the broker.

6. The data description request is transmitted to the broker from its RMCS where it is deserialized and interpreted for processing. All data descriptions that can be added to the broker are successfully saved and written to a response message. Any data descriptions that cannot be saved for any reason (i.e. a data description of the same name from the same data producer already exists) are written to a response message. An acknowledgment for the SIP session is generated and transmitted from the RMCS of the broker and forwarded through the IMS core until it is received and processed by the data producer.

7. The data description response formed in the previous step is serialized and transmitted to the RMCS of the broker.

8. The serialized data description response is converted into SIP and transmitted to the IMS core. Information about the SIP session established to transmit the information to the IMS core is transmitted to the broker.

9. The serialized data description response is forwarded through the IMS core to the RMCS of the data producer.

10. The data description response is transmitted to the data producer from its RMCS where it is deserialized and processed.

11. Information about data source descriptions registered or not registered with the broker is displayed to the end user.

12. Subscriptions created for updates about data descriptions are verified in the broker. Data description updates are created for every data consumer that has created a subscription for updates about data source descriptions.

13. The data description update is serialized and transmitted to the RMCS of the broker.

14. The serialized data description update is converted into SIP and transmitted to the IMS core. Information about the SIP session established to transmit the information to the IMS core is transmitted to the broker.

15. The serialized data description update is forwarded through the IMS core to the RMCS of the data consumer.

16. The data description update is transmitted to the data consumer from its RMCS where it is deserialized and interpreted for processing. The list of available registered data sources in the network is updated.

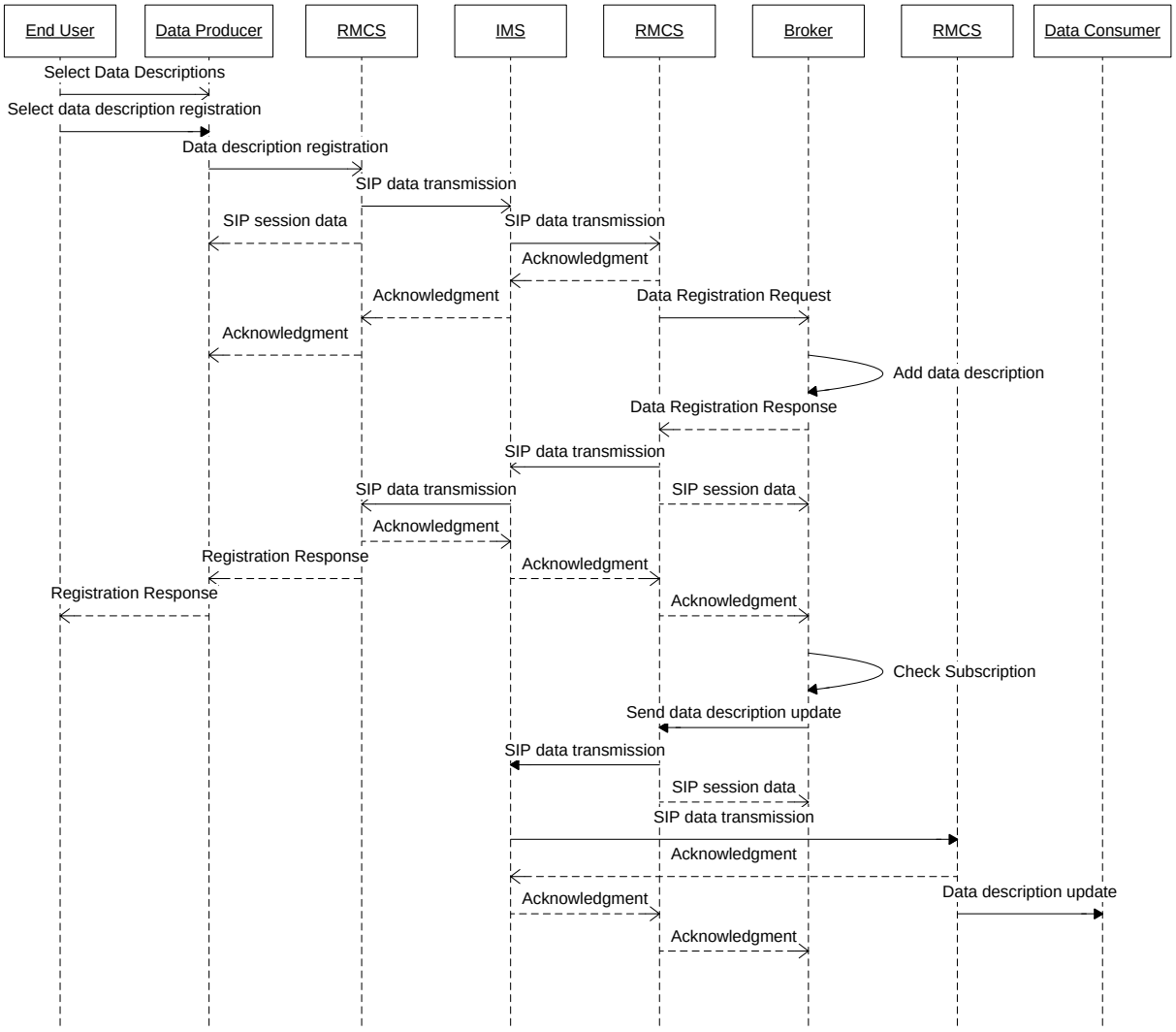

Figure 4.4: Message Exchange for the Registration of Data Source Descriptions.

Postconditions

- The descriptions of previously unregistered data sources have been added to the broker from a data producer. All data consumers who have created subscriptions to receive

updates about data sources are automatically informed of the updates.

4.2.4 Deregistering Data Descriptions

This use case describes the scenario where the end user selects an arbitrary sequence of data sources that have registered with the broker and performs a deregistration. All data consumers that have created subscriptions for information about data descriptions receive updates from the broker if any data descriptions are successfully removed.

Goals

To permit the removal of data sources from a broker by a data producer while ensuring that data consumers are informed of all data sources deregistered in the communications architecture.

Preconditions

- At least one data source description must have already been successfully registered in the broker.

Related Use Cases

- Registering Data Descriptions.

Steps

1. The end user selects the desired set of data descriptions for deregistration from the data producer.

2. The end user selects the option to deregister the selected set of data descriptions from the broker.

3. The request to deregister data descriptions is serialized and transmitted to the RMCS from its corresponding data producer.

4. The serialized data description deregistration request is converted into SIP and transmitted to the IMS core. Information about the SIP session established to transmit the information to the IMS core is transmitted to the data producer.

5. The serialized data description deregistration request is forwarded through the IMS core to the RMCS of the broker.

6. The data description request is transmitted to the broker from its RMCS where it is deserialized and interpreted for processing. All data descriptions that can be removed from the broker are successfully removed and written to a response message. Any data descriptions that cannot be removed for any reason are written to a response message. Existing data subscriptions using the removed data descriptions are updated so they are no longer in subsequent data transmissions. An acknowledgment for the SIP session is generated and transmitted from the RMCS of the broker and forwarded through the IMS core until it is received and processed by the data producer.

7. The data description response formed in the previous step is serialized and transmitted to the RMCS of the broker.

8. The serialized data description response is converted into SIP and transmitted to the IMS core. Information about the SIP session established to transmit the information to the IMS core is transmitted to the broker.

9. The serialized data description response is forwarded through the IMS core to the RMCS of the data producer.

10. The data description response is transmitted to the data producer from its RMCS where it is deserialized and processed.

11. Information about data source descriptions registered or not registered with the broker is displayed to the end user.

12. Subscriptions created for updates about data descriptions are verified in the broker. Data description updates are created for every data consumer that has created a subscription for updates about data source descriptions.

13. The data description update is serialized and transmitted to the RMCS of the broker.

14. The serialized data description update is converted into SIP and transmitted to the IMS core. Information about the SIP session established to transmit the information to the IMS core is transmitted to the broker.

15. The serialized data description update is forwarded through the IMS core to the RMCS of the data consumer.

16. The data description update is transmitted to the data consumer from its RMCS where it is deserialized and interpreted for processing. The list of available registered data sources in the network is updated.

Postconditions

- The descriptions of previously registered data sources have removed from the broker. All data consumers who have created subscriptions to receive updates about data sources are automatically informed of the updates. Existing data value subscriptions using these data descriptions are updated so there is no further attempt to transmit data from the now removed data source descriptions.

4.2.5 Removing Data Descriptions

This use case describes the scenario where the end user selects an arbitrary sequence of data source descriptions that have added to a data producer and removes their description. Any data sources registered with the broker are automatically deregistered. All data consumers that have created subscriptions for information about data descriptions receive updates from the broker if any data descriptions are successfully removed.

Goals

To permit the removal of data sources descriptions from a data producer while automatically performing a deregistration and updating data consumers as required.

Preconditions

- At least one data source description must have already been successfully added to a data producer.

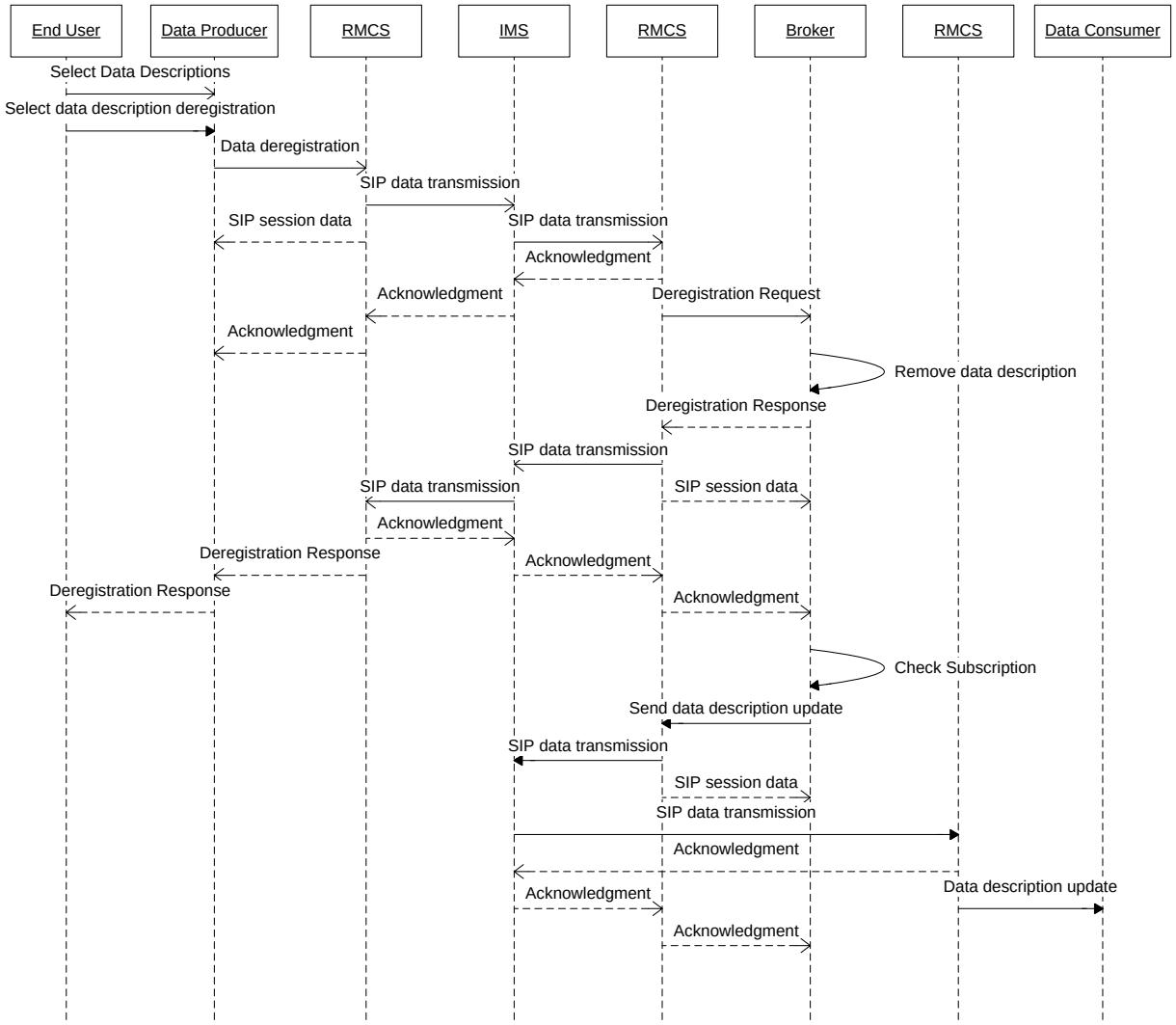

Figure 4.5: Message Exchange for the Deregistration of Data Source Descriptions.

Related Use Cases

- Deregistering Data Descriptions.

Steps

1. The end user selects the desired set of data descriptions for removal from the data producer.

2. The end user selects the option to remove the selected set of data descriptions from the broker.

3. Should any selected data description represent a data source that has been registered, a request to deregister data descriptions is serialized and transmitted to the RMCS from its corresponding data producer.

4. The serialized data description deregistration request is converted into SIP and transmitted to the IMS core. Information about the SIP session established to transmit the information to the IMS core is transmitted to the data producer.

5. The serialized data description deregistration request is forwarded through the IMS core to the RMCS of the broker.

6. The data description request is transmitted to the broker from its RMCS where it is deserialized and interpreted for processing. All data descriptions that can be removed from the broker are successfully removed and written to a response message. Any data descriptions that cannot be removed for any reason are written to a response message. Existing data subscriptions using the removed data descriptions are updated so they are no longer in subsequent data transmissions. An acknowledgment for the SIP session is generated and transmitted from the RMCS of the broker and forwarded through the IMS core until it is received and processed by the data producer.

7. The data description response formed in the previous step is serialized and transmitted to the RMCS of the broker.

8. The serialized data description response is converted into SIP and transmitted to the IMS core. Information about the SIP session established to transmit the information to the IMS core is transmitted to the broker.

9. The serialized data description response is forwarded through the IMS core to the RMCS of the data producer.

10. The data description response is transmitted to the data producer from its RMCS where it is deserialized and processed.

11. Information about data source descriptions removed from the data producer are displayed to the end user.

12. Subscriptions created for updates about data descriptions are verified in the broker. Data description updates are created for every data consumer that has created a subscription for updates about data source descriptions.

13. The data description update is serialized and transmitted to the RMCS of the broker.

14. The serialized data description update is converted into SIP and transmitted to the IMS core. Information about the SIP session established to transmit the information to the IMS core is transmitted to the broker.

15. The serialized data description update is forwarded through the IMS core to the RMCS of the data consumer.

16. The data description update is transmitted to the data consumer from its RMCS where it is deserialized and interpreted for processing. The list of available registered data sources in the network is updated.

Postconditions

- The descriptions of previously added data sources have removed from the data producer. Data source descriptions registered with the broker are automatically deregistered in the process.

4.2.6 Subscribing to Data Description Updates

This use case describes the scenario where the end user requests automatic updates about the status of data source descriptions in the network through a subscription. The requesting user would immediately receive a list of all data descriptions registered in the network and would receive an updated list every time a new data description was added or removed.

Goals To permit end users to receive information about all the data source descriptions registered in the network without polling the broker through continuous queries.

Preconditions

- None.

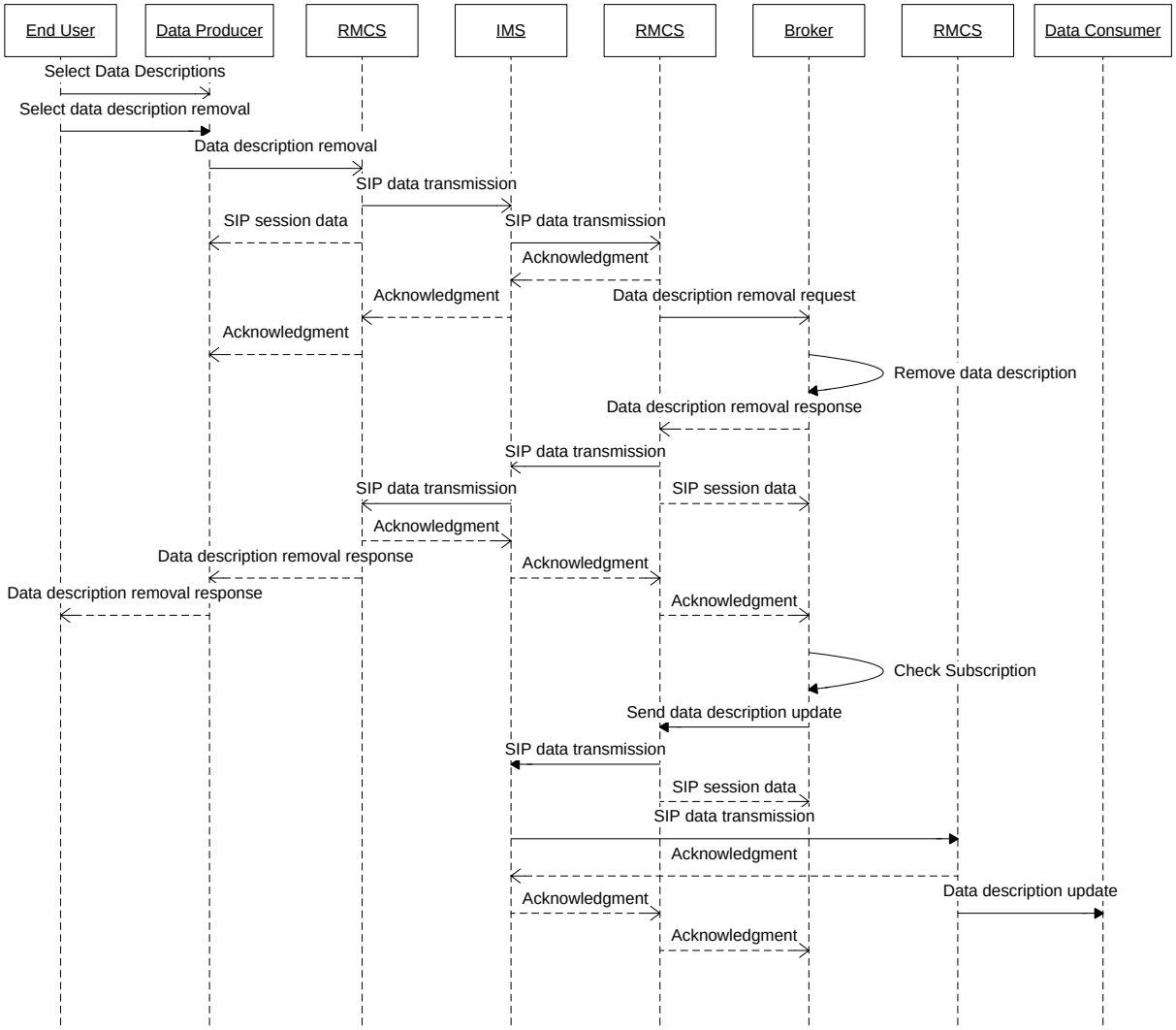

Figure 4.6: Message Exchange for the Removal of Data Source Descriptions.

Related Use Cases

- Unsubscribing from Data Description Updates.

Steps

1. The end user selects the option to create a subscription for data description updates.

2. A request to subscribe to data source descriptions is serialized and transmitted to the RMCS from its corresponding data producer.

3. The serialized data description subscription request is converted into SIP and transmitted to the IMS core. Information about the SIP session established to transmit the

information to the IMS core is transmitted to the data consumer.

4. The serialized data description subscription request is forwarded through the IMS core to the RMCS of the broker.

5. The data description subscription request is transmitted to the broker from its RMCS where it is deserialized and interpreted for processing. All data descriptions that exist in the broker are written to a response message.

6. The data description response formed in the previous step is serialized and transmitted to the RMCS of the broker.

7. The serialized data description response is converted into SIP and transmitted to the IMS core. Information about the SIP session established to transmit the information to the IMS core is transmitted to the broker.

8. The serialized data description response is forwarded through the IMS core to the RMCS of the data consumer.

9. The data description response is transmitted to the data producer from its RMCS where it is deserialized and processed.

10. Information about all data source descriptions is displayed to the end user.

11. An update to data source descriptions is transmitted from a data producer to a broker. This steps represents the sequence of steps performed in the use case 'Registering Data Descriptions' or 'Deregistering Data Descriptions'. All data descriptions that exist in the broker are written to a response message.

12. The data description response formed in the previous step is serialized and transmitted to the RMCS of the broker.

13. The serialized data description response is converted into SIP and transmitted to the IMS core. Information about the SIP session established to transmit the information to the IMS core is transmitted to the broker.

14. The serialized data description response is forwarded through the IMS core to the RMCS of the data consumer.

15. The data description response is transmitted to the data producer from its RMCS where it is deserialized and processed.

16. Information about all data source descriptions is displayed to the end user.

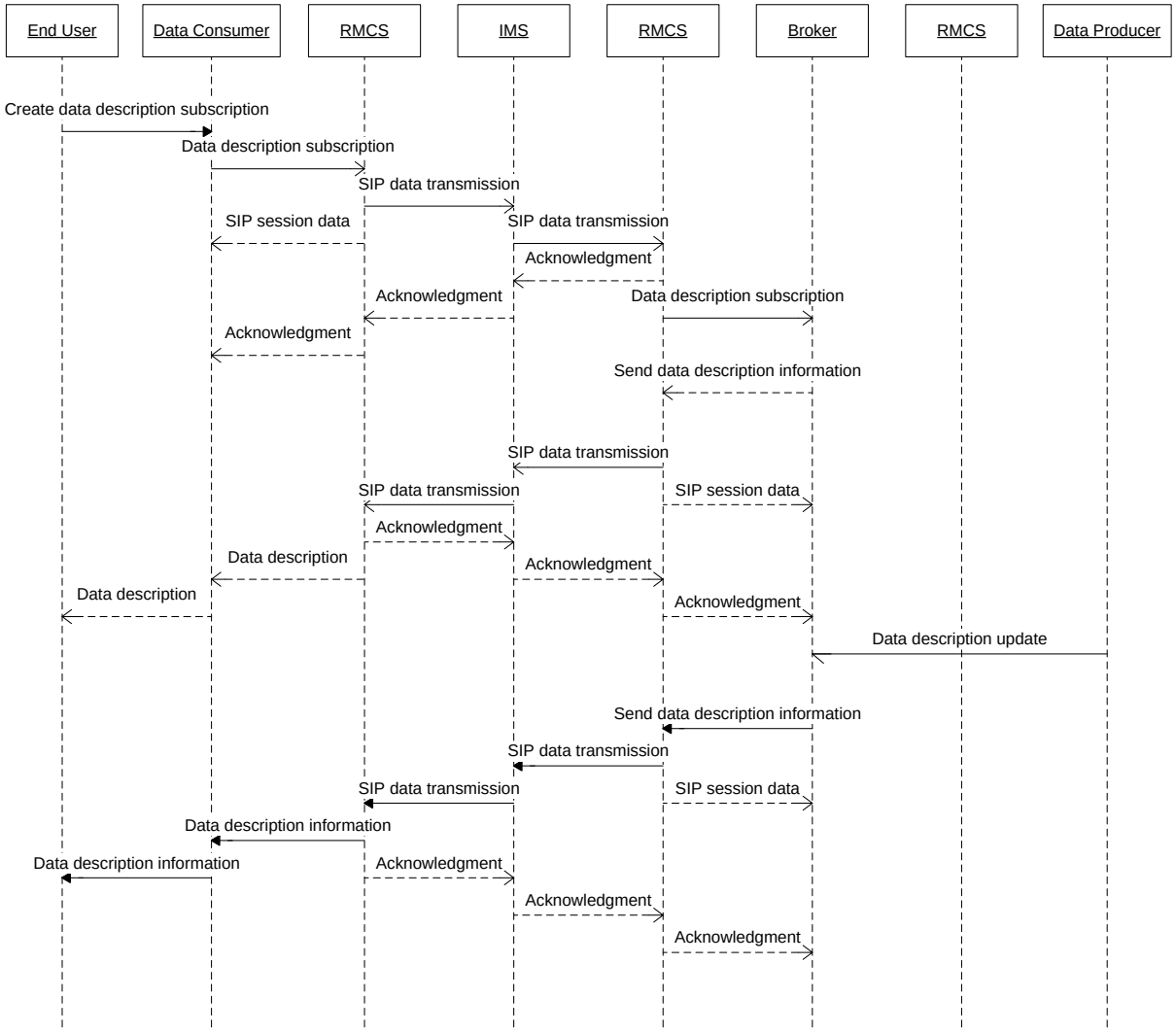

Figure 4.7: Message Exchange to Create a Subscription for Data Source Descriptions.

Postconditions

- A data consumer will automatically receive information about all the data sources

that have been registered in the network. Updates to the set of existing data source descriptions are automatically transmitted to all subscribe data consumers.

4.2.7 Unsubscribing from Data Description Updates

This use case describes the scenario where the end user requests terminates an existing subscription to receive automatic updates about the status of data source descriptions in the network.

Goals

To permit end users to terminate an ongoing request for updates about data source descriptions so data consumers are not processing unwanted information and the broker does not generate redundant data.

Preconditions

- A subscription for data source descriptions must have been successfully established between a data consumer and the broker.

Related Use Cases

- Subscribing to Data Description Updates.

Steps

1. The end user selects the option to terminate a subscription for data description updates.
2. A request to unsubscribe from data source descriptions is serialized and transmitted to the RMCS from its corresponding data consumer.
3. The serialized data description unsubscription request is converted into SIP and transmitted to the IMS core. Information about the SIP session established to transmit the information to the IMS core is transmitted to the data consumer.
4. The serialized data description unsubscription request is forwarded through the IMS core to the RMCS of the broker.

5. The data description unsubscription request is transmitted to the broker from its RMCS where it is deserialized and interpreted for processing. All data descriptions that exist in the broker are written to a response message.

6. The data description response formed in the previous step is serialized and transmitted to the RMCS of the broker.

7. The serialized data description response is converted into SIP and transmitted to the IMS core. Information about the SIP session established to transmit the information to the IMS core is transmitted to the broker.

8. The serialized data description response is forwarded through the IMS core to the RMCS of the data consumer.

9. The data description response is transmitted to the data producer from its RMCS where it is deserialized and processed.

10. Information about all data source descriptions is displayed to the end user.

Postconditions

- A data consumer will no longer receive information about all the data sources that have been registered in the network. Upon successful termination of the subscription the data consumer receives one last update about regarding data sources descriptions that have been registered in the network.

4.2.8 Querying Data Values

This use case describes the scenario where the end user makes a one-time (snapshot) request for data from an arbitrary set of data sources distributed across numerous data producers.

Goals

To permit end users to request data from multiple data sources using a single query without any repetition.

Preconditions

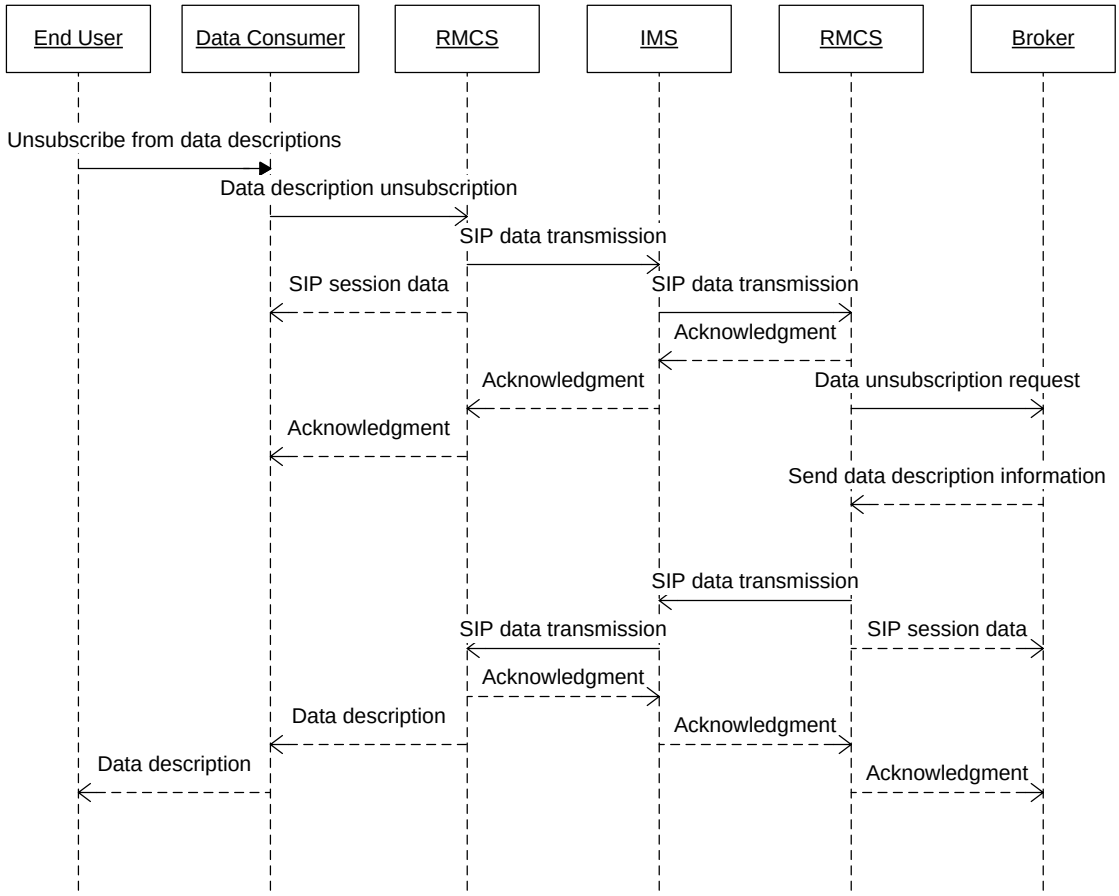

Figure 4.8: Message Exchange to Terminate a Subscription for Data Source Descriptions.

- The data source descriptions of interest must have been registered with the broker.

Related Use Cases

- Subscribing to Data Values.

Steps

1. The end user selects the desired set of data descriptions for a data value request.
2. The end user selects the option to query the selected set of data descriptions from the broker.
3. The request to query data values is serialized and transmitted to the RMCS from its corresponding data producer.

4. The serialized data query request is converted into SIP and transmitted to the IMS core. Information about the SIP session established to transmit the information to the IMS core is transmitted to the data consumer.

5. The serialized data value request is forwarded through the IMS core to the RMCS of the broker.

6. The data query request is transmitted to the broker from its RMCS where it is deserialized and interpreted for processing. Information regarding all the data descriptions and their corresponding data brokers is parsed from the query request. Data query requests are formed for every data producer listed in the data query from the data consumer.

7. The data value query (for every data producer) formed in the previous step is serialized and transmitted to the RMCS of the broker.

8. The serialized data query request is converted into SIP and transmitted to the IMS core. Information about the SIP session established to transmit the information to the IMS core is transmitted to the broker.

9. The serialized data value request is forwarded through the IMS core to the RMCS of the data producer.

10. The data query request is transmitted to the data producer from its RMCS where it is deserialized and interpreted for processing. Information regarding all the data descriptions and their corresponding data brokers is parsed from the query request. The requests data values are generated and written to a data query response.

11. The data value response formed in the previous step is serialized and transmitted to the RMCS of the data producer.

12. The serialized data query response is converted into SIP and transmitted to the IMS core. Information about the SIP session established to transmit the information to the IMS core is transmitted to the data producer.

13. The serialized data value response is forwarded through the IMS core to the RMCS of the broker.

14. The data query response is transmitted to the broker from its RMCS where it is deserialized and interpreted for processing. A data response for the original data query (from the data consumer) is updated with information from the data query request from the data producer.

15. When all the requested data from the data consumer has been received a data value response is formed, serialized and transmitted to the RMCS of the broker.

16. The serialized data query response is converted into SIP and transmitted to the IMS core. Information about the SIP session established to transmit the information to the IMS core is transmitted to the data consumer.

17. The serialized data query response is forwarded through the IMS core to the RMCS of the data consumer.

18. The data query response is transmitted to the data consumer from its RMCS where it is deserialized and interpreted for processing.

19. All data received through the data query response is displayed to the end user.

Postconditions

- Data has been generated by potentially numerous data sources and transmitted to a data consumer upon request.

4.2.9 Subscribing to Data Values

This use case describes the scenario where the end user makes a continuous request for data from an arbitrary set of data sources distributed across numerous data producers.

Goals

To permit end users to request continuous data from multiple data sources using a single request.

Preconditions

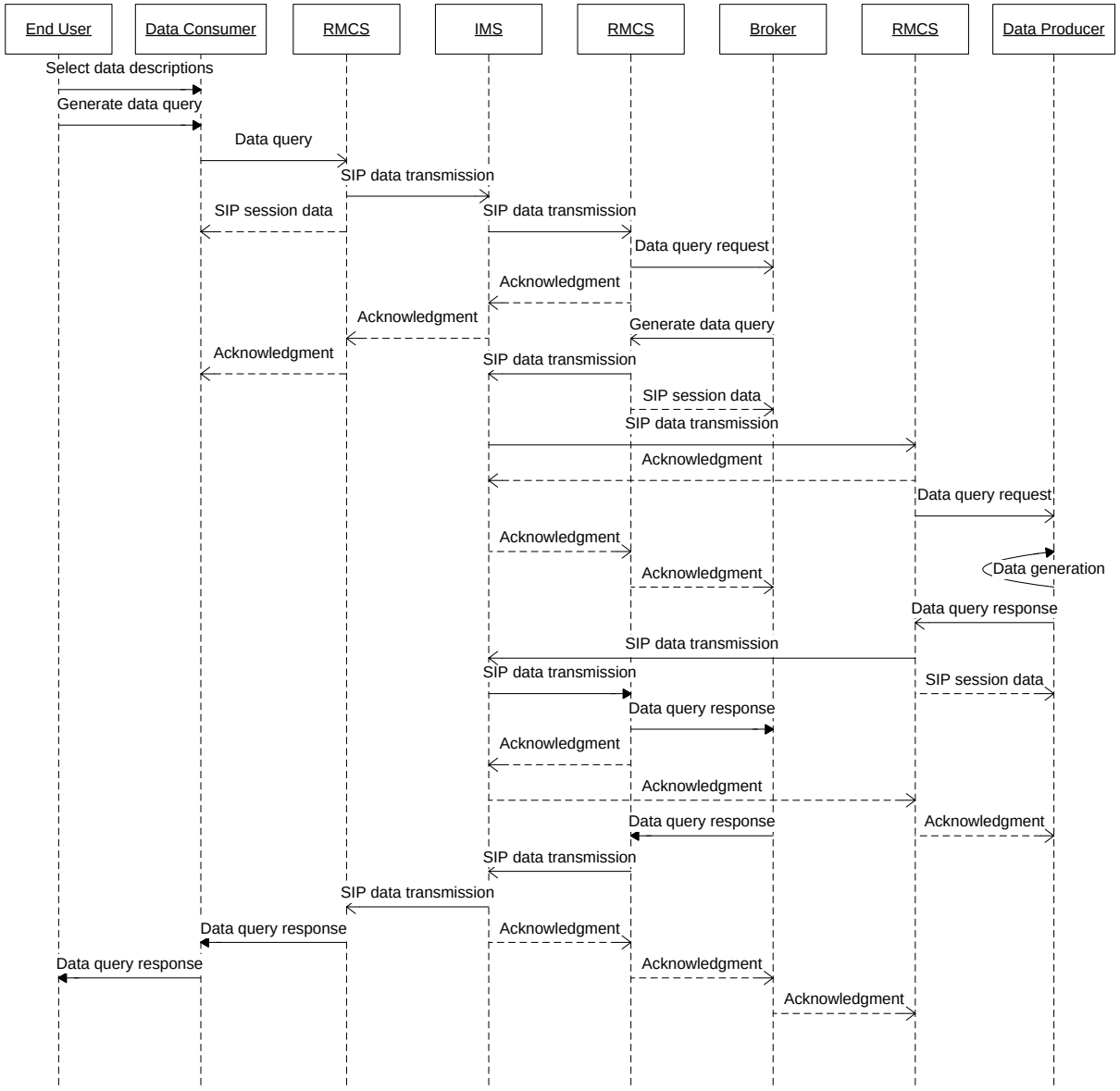

Figure 4.9: Message Exchange to Query for Data Values from Data Sources.

- The data source descriptions of interest must have been registered with the broker.

Related Use Cases

- Querying to Data Values.

Steps

1. The end user selects the desired set of data descriptions for a data value request.

2. The end user selects the option to query the selected set of data descriptions from the broker.

3. The request to query data values is serialized and transmitted to the RMCS from its corresponding data producer.

4. The serialized data query request is converted into SIP and transmitted to the IMS core. Information about the SIP session established to transmit the information to the IMS core is transmitted to the data consumer.

5. The serialized data value request is forwarded through the IMS core to the RMCS of the broker.

6. The data query request is transmitted to the broker from its RMCS where it is deserialized and interpreted for processing. All the information requested in the data query is immediately retrieved and sent to the data consumer. Additional requests are performed based on timeouts generated using the parameters specified in the continuous data query. When a timeout occurs the process to retrieve all the information requested in the original query is repeated.

7. The data value query (for every data producer) is serialized and transmitted to the RMCS of the broker.

8. The serialized data query request is converted into SIP and transmitted to the IMS core. Information about the SIP session established to transmit the information to the IMS core is transmitted to the broker.

9. The serialized data value request is forwarded through the IMS core to the RMCS of the data producer.

10. The data query request is transmitted to the data producer from its RMCS where it is deserialized and interpreted for processing. Information regarding all the data descriptions and their corresponding data brokers is parsed from the query request. The requests data values are generated and written to a data query response.

11. The data value response formed in the previous step is serialized and transmitted to the RMCS of the data producer.

12. The serialized data query response is converted into SIP and transmitted to the IMS core. Information about the SIP session established to transmit the information to the IMS core is transmitted to the data producer.

13. The serialized data value response is forwarded through the IMS core to the RMCS of the broker.

14. The data query response is transmitted to the broker from its RMCS where it is deserialized and interpreted for processing. A data response for the original data query (from the data consumer) is updated with information from the data query request from the data producer.

15. When all the requested data from the data consumer has been received a data value response is formed, serialized and transmitted to the RMCS of the broker.

16. The serialized data query response is converted into SIP and transmitted to the IMS core. Information about the SIP session established to transmit the information to the IMS core is transmitted to the data consumer.

17. The serialized data query response is forwarded through the IMS core to the RMCS of the data consumer.

18. The data query response is transmitted to the data consumer from its RMCS where it is deserialized and interpreted for processing.

19. All data received through the data query response is displayed to the end user.

Postconditions

- Data has been continuously generated by potentially numerous data sources and transmitted to a data consumer.

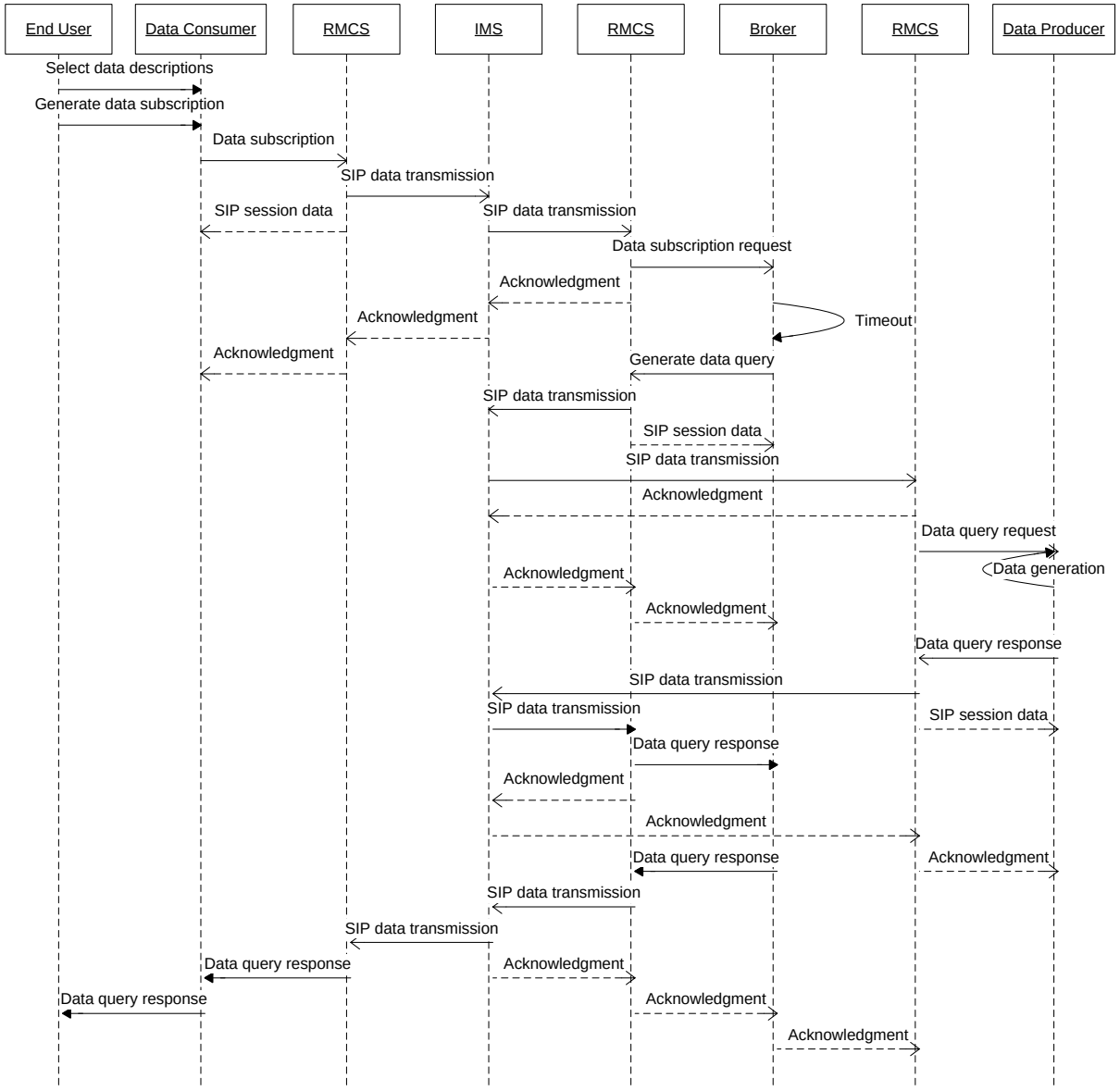

Figure 4.10: Message Exchange to Subscribe to Data Values from Data Sources.

4.2.10 Unsubscribing from Data Values

This use case describes the scenario where the end user requests terminates an existing subscription to receive data values through a previously established subscription.

Goals

To permit end users to terminate an ongoing request for data so data consumers are not processing unwanted information, data producers are not generating redundant data and no

unnecessary processing is occurring in the broker.

Preconditions

- A subscription for data source values must have been successfully established between a data consumer and the broker.

Related Use Cases

- Subscribing to Data Values.

Steps

1. The end user selects the option to terminate a subscription for data values.
2. A request to unsubscribe from data values is serialized and transmitted to the RMCS from its corresponding data consumer.
3. The serialized data value unsubscription request is converted into SIP and transmitted to the IMS core. Information about the SIP session established to transmit the information to the IMS core is transmitted to the data consumer.
4. The serialized data value unsubscription request is forwarded through the IMS core to the RMCS of the broker.
5. The data value unsubscription request is transmitted to the broker from its RMCS where it is deserialized and interpreted for processing. The corresponding subscription is terminated so no further timeouts are generated.
6. A data value response is formed, serialized and transmitted to the RMCS of the broker.
7. The serialized data value response is converted into SIP and transmitted to the IMS core. Information about the SIP session established to transmit the information to the IMS core is transmitted to the broker.
8. The serialized data value response is forwarded through the IMS core to the RMCS of the data consumer.

9. The data value response is transmitted to the data producer from its RMCS where it is deserialized and processed.

10. Information about all data values is displayed to the end user.

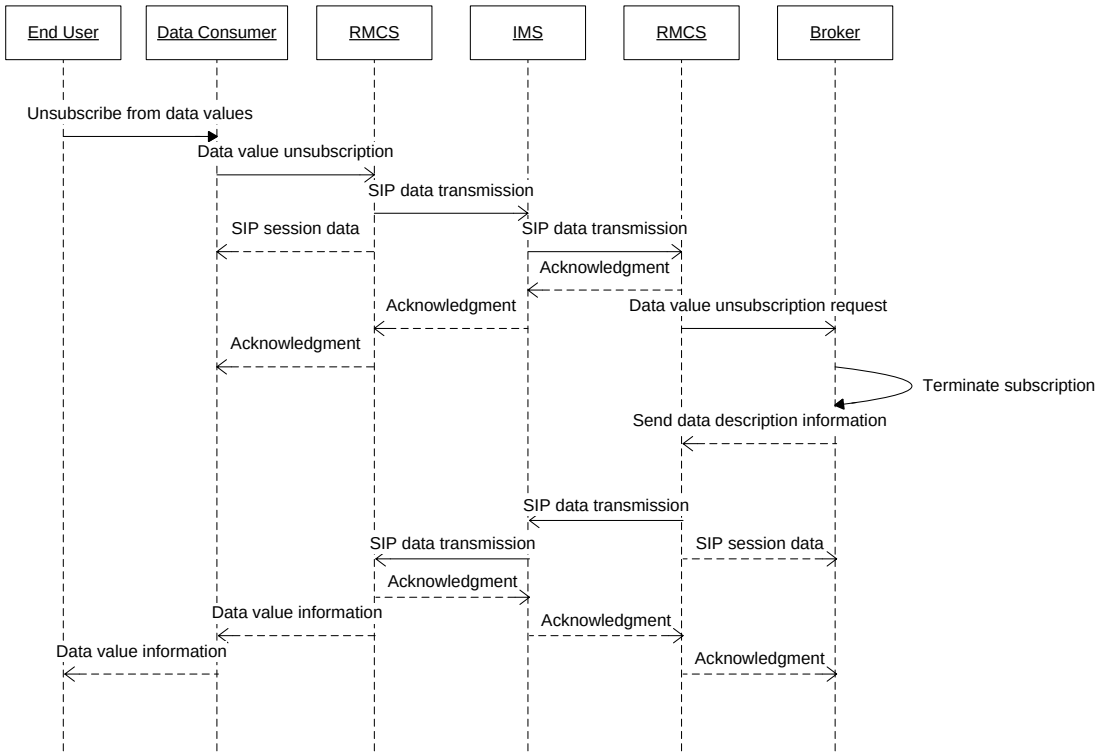

Figure 4.11: Message Exchange to Terminate a Subscription for Data Values.

Postconditions

- A data consumer will no longer receive data values on a period basis through a continuous query that have been registered in the network. Upon successful termination of the subscription the data consumer receives one last update about regarding data sources descriptions that have been registered in the network.

4.3 Conclusion

This chapter described relevant information regarding data dissemination by specifying the session management features and uses cases required for the dissemination of heterogeneous

data. Session management is performed by all components of the architecture in a manner where sessions defined at the application layer are mapped to sessions managed through the RMCS. Numerous use cases were defined to fulfill the requirements of the communications architecture. The definition and removal of descriptions for data sources, querying data and termination of communications sessions are all possible with the architecture to facilitate the efficient dissemination of heterogeneous data in real-time between entities connected through IMS.

Chapter 5

Components and Algorithms

This chapter provides a detailed description of the components used to exchange data through communications architecture defined in this thesis. The protocol defines a broker, data consumer and data producer used to disseminate heterogeneous data. These components are described in separate sections in this chapter. Subcomponents and algorithms are defined as required to identify important functions and how they are performed. Finite state machines are used to describe the relevant algorithms. The formats of the messages used for data dissemination are presented in Appendix A for additional information.

5.1 Broker

Fig. 5.1 illustrates the high level components of the broker. A central algorithm is used to process data received through all other components, generate the appropriate responses and coordinate all other activities accordingly.

The RMCS is used to receive data from the IMS network and write any responses to the network that are generated from the broker. The sessions component is used to manage information regarding all sessions of interest including sessions managed in the RMCS and any sessions created using the high level protocol implemented to exchange data between the broker, data consumers and data producers. The subscriptions component is used to manage all information related to subscriptions created by data consumers to receive automatic updates regarding data descriptions and perform continuous, periodic queries. The

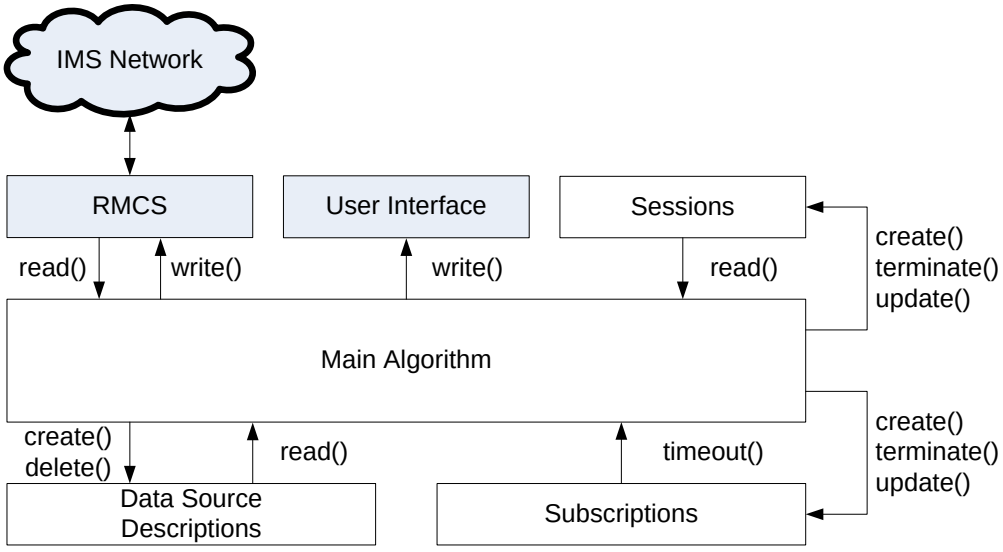

Figure 5.1: The Architecture of the System Broker.

data source descriptions component contains all the information registered and deregistered from data producers regarding their data sources. All feedback generated by the broker is presumed to be written to a user interface.

5.1.1 Main Algorithm

Fig. 5.2 describes a finite state machine illustrating the main algorithm of the broker. The broker is initialized in an idle state and proceeds to read the RMCS for any data that has been received from the IMS network. Any data received from the RMCS is immediately processed before subscriptions are checked for any timeouts that may have occurred. Any timeouts prompt the generation of a query to the appropriate set of data producers. The entire process is repeated after all data sources have been checked and all data has been processed. The subsections below provide additional details on the procedures required to process data from the RMCS and generate queries based on timeouts.

Fig. 5.3 describes a finite state machine illustrating the algorithm performed to process data from the RMCS in the broker. This algorithm is performed when the 'Process SIP Interface Data' state has been reached in the main algorithm of the broker. When data is read from the RMCS is it parsed to determine the type of the message. Five message types

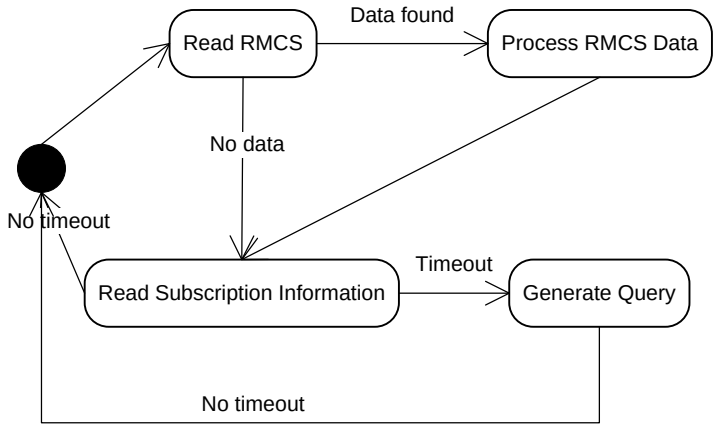

Figure 5.2: The Main Algorithm of the Broker.

generated by the RMCS are directly supported by the broker. Unsupported types result in the broker transmitting feedback to the user interface presenting the message that has been received and terminating the algorithm.

When a message has been received from the RMCS indicating that a new SIP session has been created, all information regarding this message must be saved in the broker for later reference. This message is ordinarily received by the broker when a new message has been transmitted from the broker through the RMCS. A new session is created in the RMCS to permit the transmission of this data through the IMS network. Information regarding this session is transmitted to the broker when it has been successfully established so future updates received about this session can be processed properly.

A request for a new subscription may be received from the RMCS and must be processed by the broker. The subscription request is parsed to determine whether a subscription must be created to generate notifications for updates to data source descriptions or data values. If a subscription request has been created for data values most of the information regarding the subscription request is contained in a payload that must be processed according to the specifications of the high level protocol. Data source description requests do not contain any payloads and therefore can be created immediately. The process to create a data source description subscription involves creating a data description alert (with information for all data source descriptions registered with the broker) so it can be immediately transmitted to the requesting data consumer. An outgoing session is created to transmit the data description

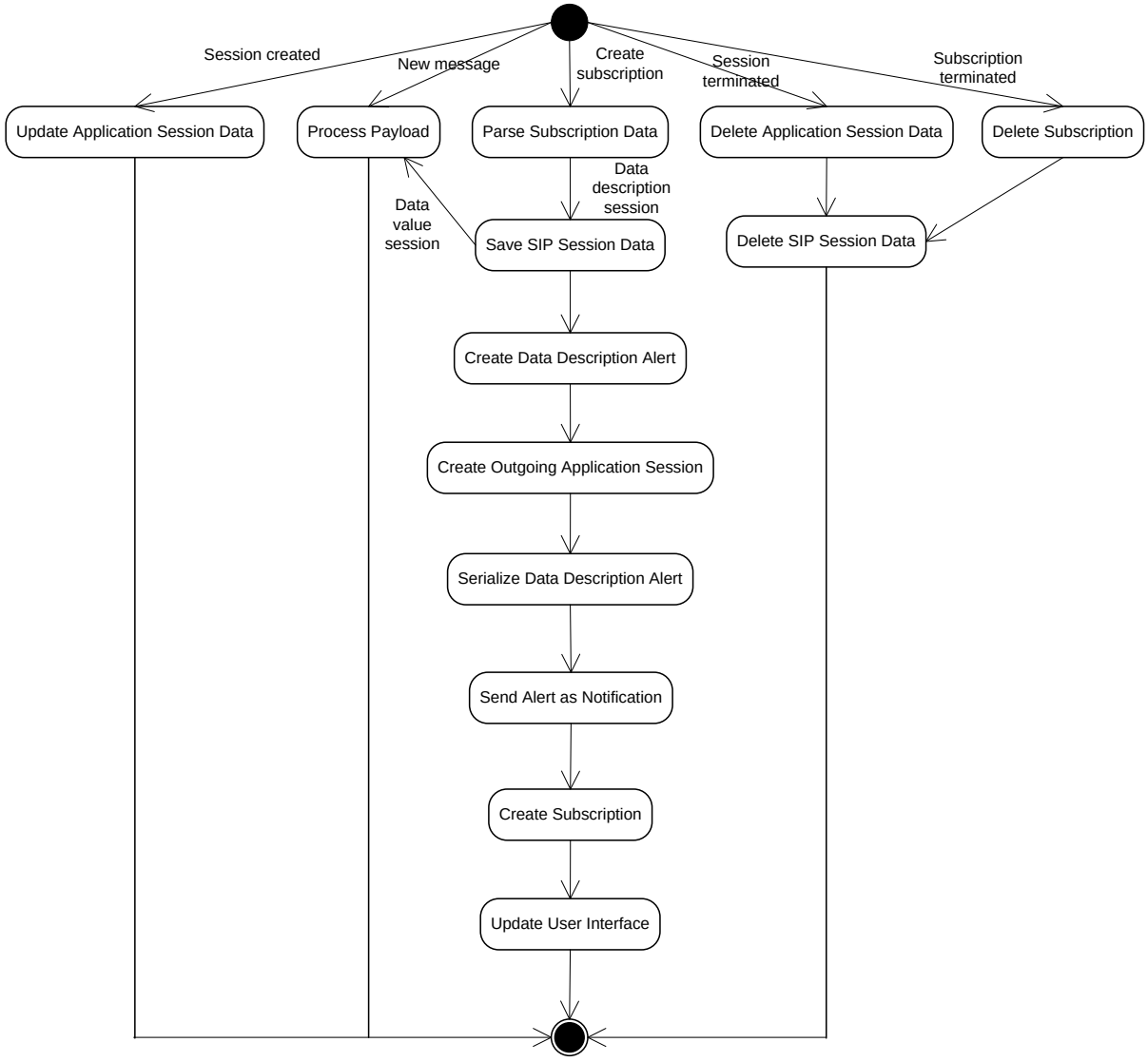

Figure 5.3: The Algorithm to Process SIP Data in the Broker.

alert before the alert is subsequently serialized and send through a notification using the subscription through the RMCS.

Messages to terminate subscriptions are also received by the broker from the RMCS anytime a SIP session is terminated. This message may be received when a SIP message is acknowledged and no other messages are required or when a high level command from a data consumer or data producer results in the termination of a session created in the RMCS. SIP sessions have an associated high level session since all high level data must be transported through the IMS network using a SIP session. Therefore the termination of a SIP session

requires the termination of its corresponding high level session in the broker. If the session does not exist then an error message is generated and transmitted to the user interface.

The termination of a subscription requires similar actions to the termination of an ordinary SIP session. Subscriptions are managed by the subscriptions component of the broker used to generate timeouts to indicate when notifications must be generated for data consumers. These subscriptions must be removed from the subscriptions component to prevent the generation of future notifications. Furthermore information for the session created in the RMCS for the subscription must be removed when subscriptions no longer exist.

When a new instant message is received from the RMCS the payload of the message must be processed by the broker according to the specifications of the high level protocol used to exchange data among data producers and data consumers. A number of algorithms may be performed depending on the data that has been received. The algorithm performed to add a new data source description to the broker is described in Fig. 5.4.

When a request to add a new data source description has been recognized an outgoing session is created to transmit the response that is formed with the list of all registered data sources. A response message is created for the data producer that has submitted the data source for registration and a data description alert is created for all the data consumers with registrations for data description updates. All unique data descriptions that have not been registered in the broker are added to the broker and noted as accepted data descriptions in the response message. Any data description that has already been registered is noted as a rejected data description. When all data descriptions from the initial request have been processed the response message is serialized and transmitted to the data producer as an instant message. Every registered data source description is added to the data description alert and transmitted to all subscribed data consumers.

The algorithm required to remove a data source description from the broker is illustrated in Fig. 5.5. This algorithm is similar to the process described in Fig. 5.4 with the notable exception that data descriptions are removed instead of added and that data value descriptions must be updated with the removal of any data descriptions. An outgoing data description, response and data description alert are also created and every data description received in the request is processed to update the list of registered data descriptions. A response with

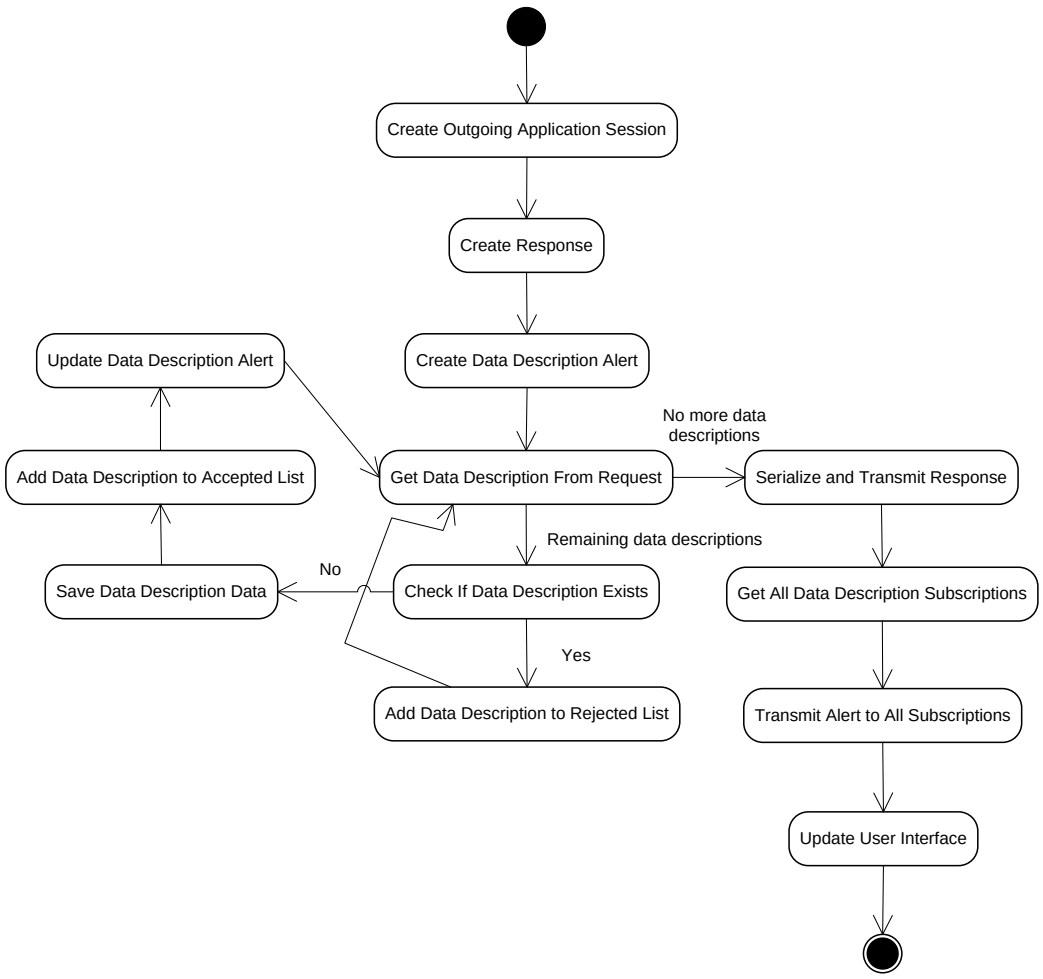

Figure 5.4: The Algorithm to Add a Data Source Description to the Broker.

the accepted and rejected data source descriptions is transmitted to the data producer when the entire request has been processed. Before transmitting updates about data descriptions, every data value subscription is updated to remove references for deleted data sources.

The algorithm performed when processing a data request in the broker is described in Fig. 5.5. During this procedure a sequence of data requests may be transmitted to various data producers on behalf of a single data consumer to request a multitude of data from data sources distributed throughout the IMS network. Information for the incoming communications session is immediately saved upon receiving a new request before creating a response that must be populated as data is received through various requests. When a timeout is generated from a data value subscription there is no need to create a new incoming session since it

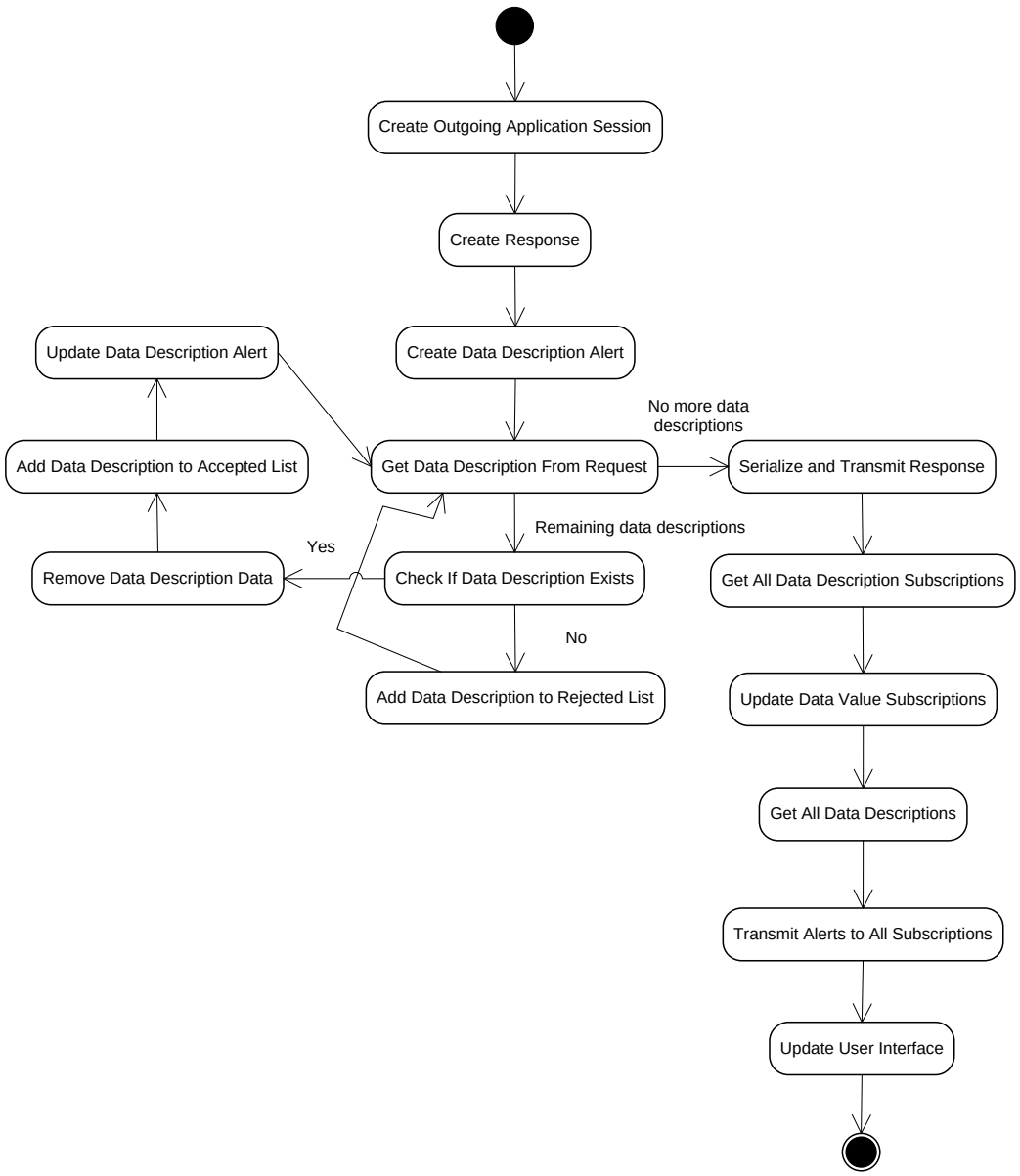

Figure 5.5: The Algorithm to Remove a Data Source Description from the Broker.

was created when the original subscription request was received. The list of data identifiers from the data request is parsed to isolate the list of data sources that are registered in the broker. Unregistered data sources are added to the list of rejected data identifiers. If no acceptable data sources are identified then a response is immediately transmitted to the data consumer to indicate that the request could not be fulfilled. Acceptable data identifiers are subsequently sorted into groups based on the data producers so multiple data requests can

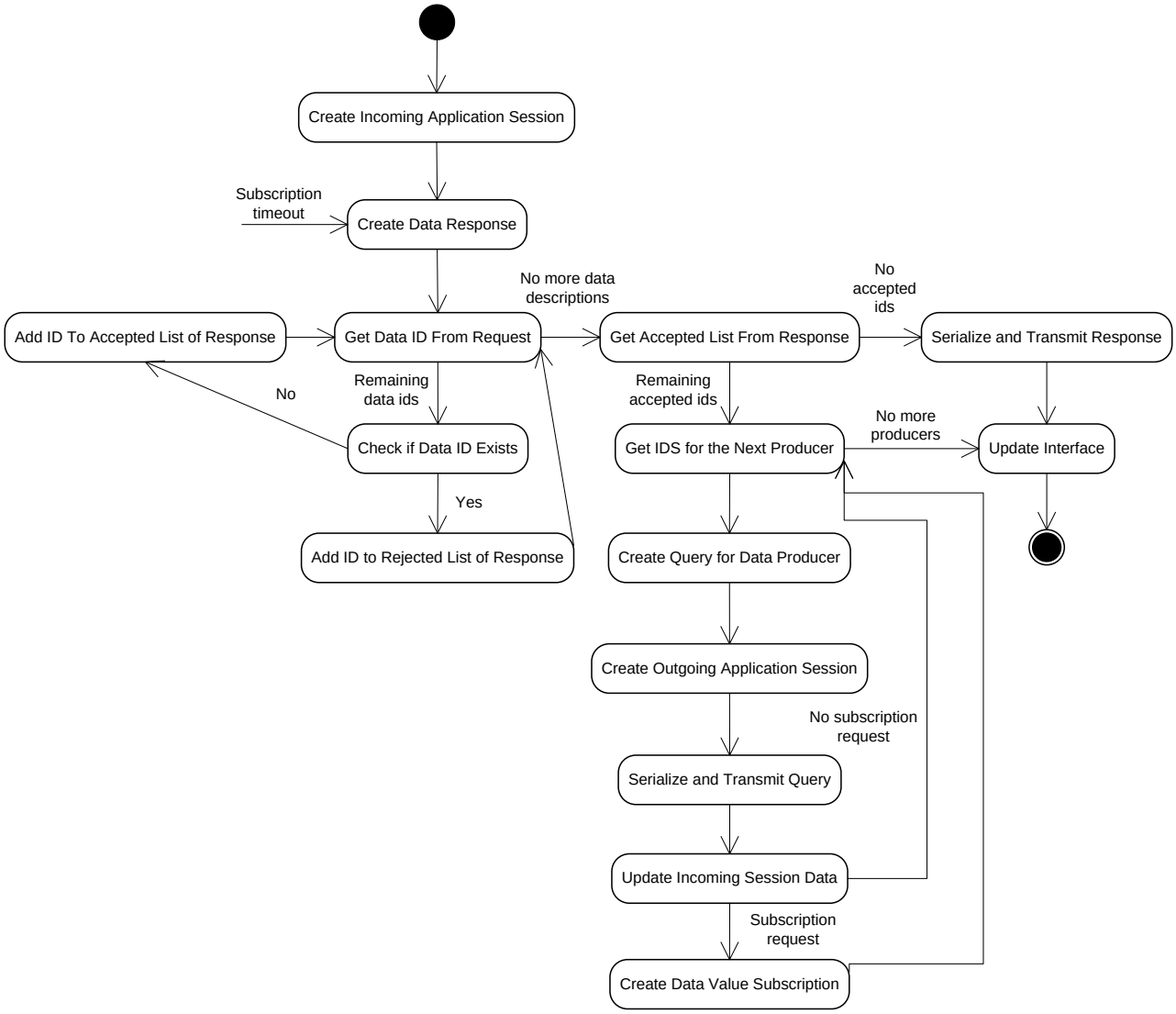

Figure 5.6: The Algorithm to Process a Data Request in the Broker.

be transmitted. One data request is transmitted per data producer (instead of transmitting one request per data source) so data can be transmitted as efficiently as possible through the network. With the creation of each data query the incoming data request is updated with information regarding the session used to transmit the request so responses can be appropriately mapped to the response that must be transmitted to the data consumer. When the data request is also a subscription request, a new data value subscription is created so additional data requests can be submitted in the future using timeout according to specifications provided by the data consumer.

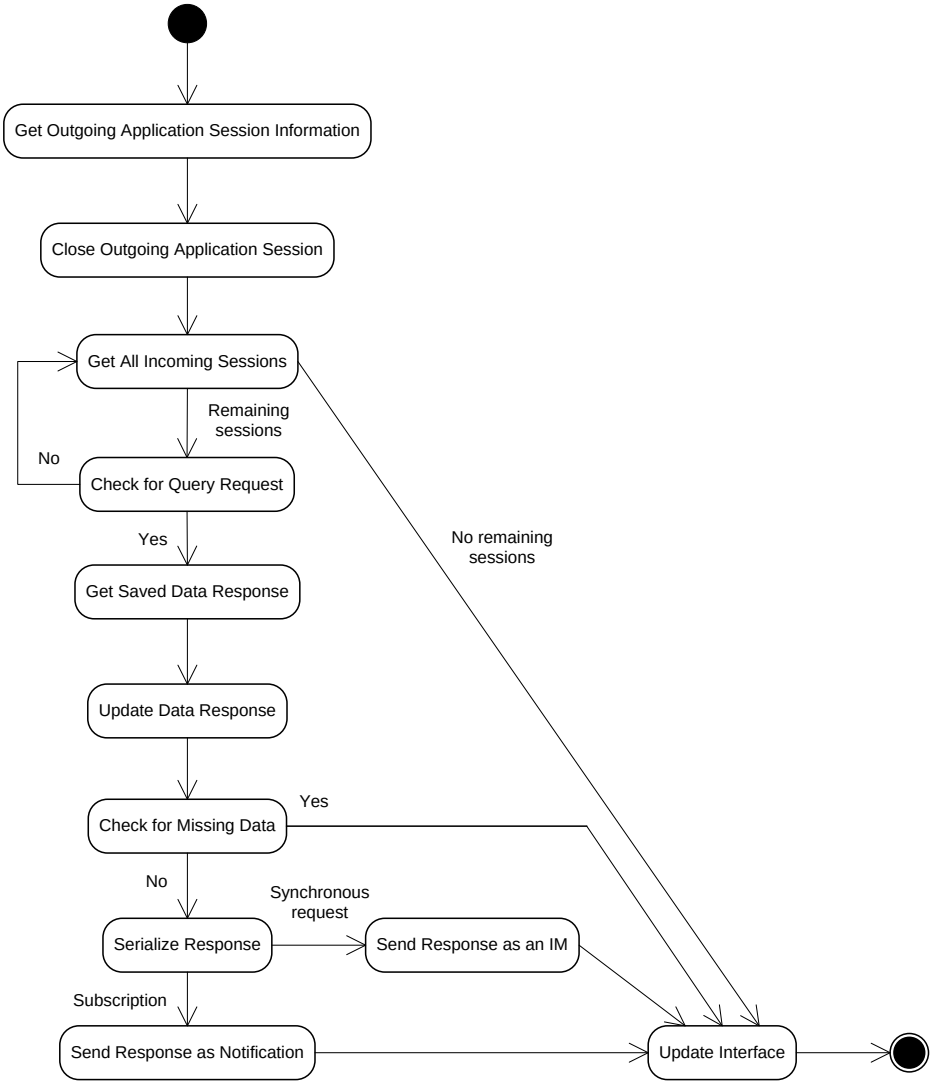

Figure 5.7: The Algorithm to Process a Data Response in the Broker.

The procedure required to parse a data response is illustrated in Fig. 5.7. The outgoing session required to transmit the request to the data producer is closed before information about the incoming session from the data consumer is retrieved. The data response for the data consumer is updated with the data received from the data producer. If additional information is required from more data producers then the algorithm is terminated. If all the information requested by the data consumer has now been received then the response is serialized and transmitted to the data consumer as an instant message or notification.

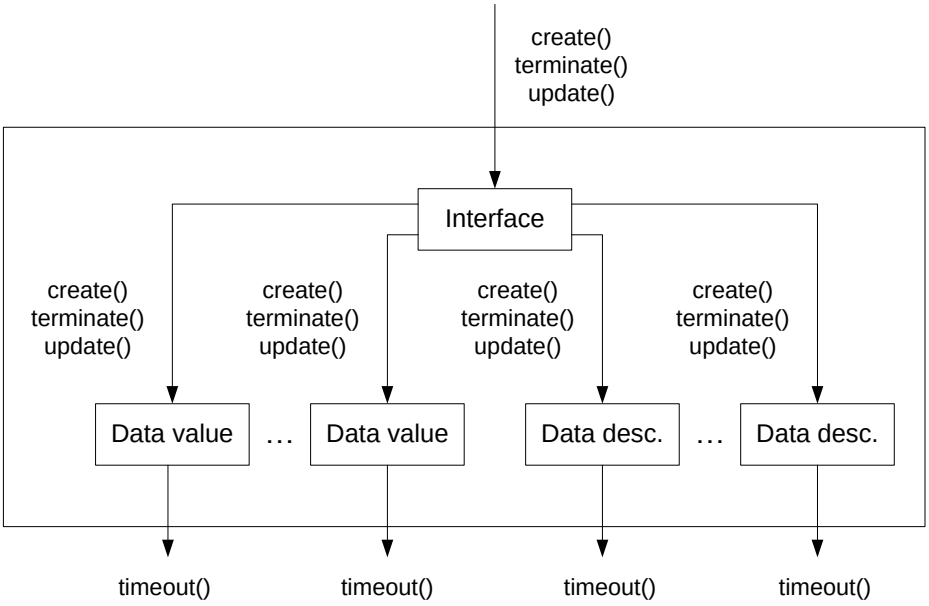

Figure 5.8: The Subscriptions Management Component of the Broker.

5.1.2 Subscription Management

All subscriptions executed through the broker are managed through the subscriptions component introduced and described in Fig. 5.1. The subscriptions management mechanism is described in Fig. 5.8.

Subscriptions are created to manage ongoing requests for data source descriptions and data values. When a subscription is created a new process is instantiated in the broker that runs independently from the main process used to interact with the IMS network and write data to a user interface. Information about each process is returned to the main application for future reference. Each subscription contains a timer that generates a timeout when new information must be sent to the user that transmitted the subscription request. Data source descriptions generate a single timeout when the requesting user specified the subscription should end. Subscriptions for data values generate a timeout every time a new data request must be transmitted on behalf of the requesting data consumer. These subscriptions can be updated to change the information requested with each time out should any data sources be deregistered during the lifetime of the subscription. A subscription terminates itself when the last possible timeout is generated based on the requirements of the requesting entity. A

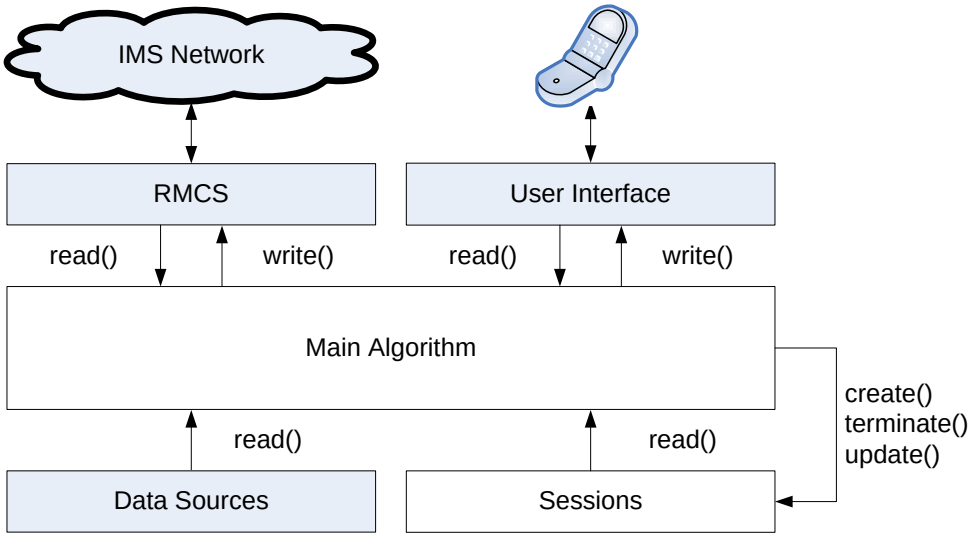

Figure 5.9: The Architecture of the Data Producer.

subscription may also be arbitrarily terminated on request to prevent future timeouts from being generated.

5.2 Data Producer

Fig. 5.9 illustrates the high level components of the data consumer. A central algorithm is used to process data and coordinate the activities of other components like the algorithm for the broker and data producer.

The RMCS, data source descriptions and sessions components are used to perform the same functions described for the broker. The user interface is the means through which data is transmitted and received from the end user. The collection of data sources can be read at any time to generate data. The interfaces through which data is ready is dependent on the data source. The mechanisms required to integrate the data source interfaces into the data producer are beyond the scope of this thesis and are not directly discussed.

External commands are required to initiate certain commands in the data producer. These commands are related to the creation, deletion, registration and deregistration of data source descriptions. Responses to data query requests are automatically performed as data is processed from the RMCS. The creation of a data source description requires

an arbitrary string for a name and a description provided to data consumers for optional information about the data. No two data sources can be created with the same name by a data producer. The deletion, registration and deregistration of data source descriptions require selecting an arbitrary set of data sources before the operation can be performed. The selection of an incompatible set of data sources for an operation prompt an response to be transmitted through the user interface to the end user. No operation is performed if no appropriate data sources have been selected. In this manner, the total amount of data transmitted through the IMS network is minimized and the probability of error is reduced.

The data producer can process RMCS messages regarding a new instant message session acknowledgment, session creation, and session termination. Instant messages are received with payloads that must be interpreted according to the high level protocol for exchanging data between the data producer and the broker. All other messages are related to session or subscription management and result in updating the sequence of sessions maintained in the data producer to keep track of data and associate the reception of data with a specific request.

The payloads of instant messages may be interpreted as a data request, the addition of a data description or the removal of a data description. When a data request is received, all the data sources named in the request are read to generate data. This data is compiled into a data response that is created as soon as the data request has been successfully parsed. Any non-existent data sources are added to the rejected list of the data response. An outgoing session is created and the data response is serialized and transmitted to the broker. When a response is received for the addition of data source descriptions all successfully added data sources are internally updated to indicate they have been successfully added to the broker. This information is used to prevent future operations from being performed incorrectly. A response for the removal of a data source description causes all information for the appropriate data sources to be removed entirely.

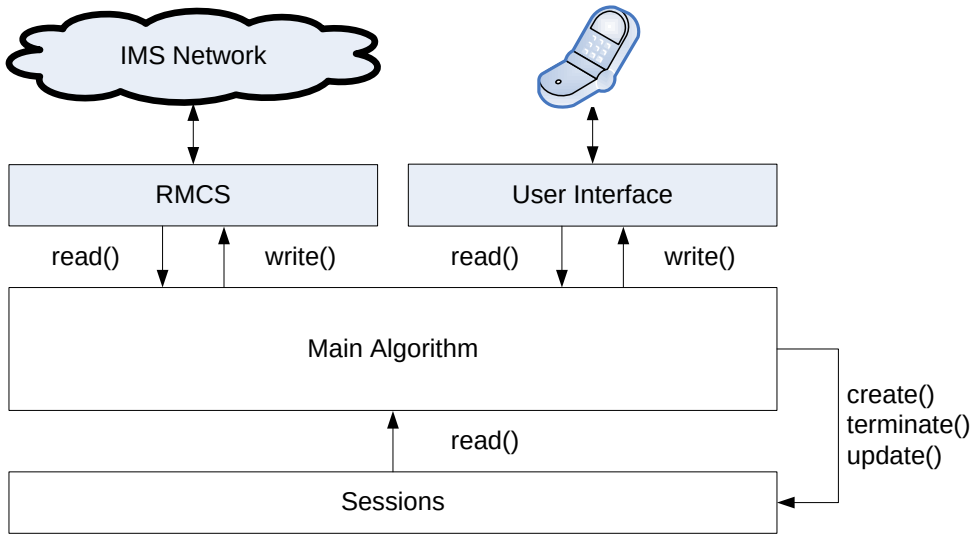

Figure 5.10: The Architecture of the Data Consumer.

5.3 Data Consumer

Fig. 5.10 illustrates the high level components of the data consumer. A central algorithm is used to process data and coordinate the activities of other components like the algorithm for the broker and data consumer.

The RMCS and sessions components are used to perform the same functions described for the broker and data producer. The user interface is the means through which data is transmitted and received from the end user.

External commands required to initiate requests are sent by an end user through the user interface. The data consumer can process commands to subscribe and unsubscribe for data source description updates. No additional information is required to create a subscription while a session identifier must be specified when indicating a subscription must be terminated. Data values may also be requested through synchronous queries or continuous subscriptions. To request any data, the sequence of data values must first be selected and specified in the request. The two parameters required to perform a query are a subscription length and rate at which the query must be repeated by the broker. A synchronous data request is performed when either the subscription length or repetition rate are specified as zero or if either value falls outside an acceptable range of values. Should the information

required for the subscription be correct then a subscription request is made to query data.

The data consumer can process RMCS messages regarding a new instant message, new notification, session acknowledgment, session creation, session termination and subscription termination. Instant message and notification messages are received with payloads that must be interpreted according to the high level protocol for exchanging data between the data consumer and the broker. All other messages are related to session or subscription management and result in updating the sequence of sessions maintained in the data consumer to keep track of data and associate the reception of data with a specific request.

Instant messages and notifications must be received as data responses or data description alerts respectively. Should any other message be received outside of these circumstances an error is generated and reported to the user interface. When a data response is received, the data is parsed and displayed in the user interface. Sessions are terminated when the data was received as a result of a synchronous query. Data description alerts prompt the internal list of data source descriptions to be updated so future data value requests can be performed with accurate information.

5.4 Conclusion

This chapter described the components and algorithms of the communications architecture defined in this thesis. A broker, data consumer and data producer are the data dissemination components of the architecture. The broker contains components for session management, subscription management and data source descriptions and is coordinated through algorithms for SIP message processing, the addition and removal of data source descriptions, fulfilling data requests and generating data responses. The data producer accepts commands from an end user to create, delete, register and deregister data sources while responding to data query requests from the broker. The data consumer accepts user requests to manage subscriptions for data source descriptions and data values.

Chapter 6

Communications Framework

This chapter provides a detailed description of the communications framework required to exchange data between data consumers, data producers and the broker. This framework is based on SIP because it is the primary communications protocol of IMS. The first section describes the SIP packets required to exchange data between the architecture components. A sequence of requests, responses and alerts defined for the RMCS are described in significant detail. All use cases for the RMCS to interact with architecture components and each other through the IMS network are fully described to illustrate how messages are successfully mapped between SIP packets.

6.1 SIP Packets Used to Exchange Data

Data must be formatted according to the specifications of SIP for it to be forwarded through an IMS network between end users. The arbitrary exchange of data between two entities can be performed using the instant messaging framework of SIP described in Chapter 2. Application level data can be transmitted from one end user to another by appending the data to a SIP MESSAGE packet as an instant message and describing the data as plain text. Fig. 6.1 illustrates the use of a SIP MESSAGE packet used to transmit data through an IMS network between end users.

The subscription framework of SIP can be used to create subscriptions and exchange

```
MESSAGE sip:fadi@uottawa.ca SIP/2.0
Via: SIP/2.0/TCP domain2.uottawa.ca;
branch=z9hG4bK71a2g5reh
Max-Forwards: 10
From: <sip:ray@uottawa.ca>;tag=49583
To: <sip:fadi@uottawa.ca>
Call-ID: 1234@110.120.130.140
CSeq: 4 MESSAGE
Content-Type: text/plain
Content-Length: 40

Data exchanged between two architecture components.
```

Figure 6.1: A SIP MESSAGE packet used to transmit data between architecture components.

data between architecture components. Limitations exist in the subscriptions mechanisms of SIP that prevent data from being exchanged as efficiently as possible according to the requirements listed in Chapter 4. One limitation is that users can only subscribe to a limited set of data transmitted in predefined formats. No mechanisms exist for an end user to define a schema for heterogeneous data in real-time and transmit their schema through a subscription request. Another limitation is the absence of a feature for end users to request a data subset through a subscription. The instant messaging framework of SIP could be used to permit a user to continuously poll data among multiple data producers. However this approach would result in significant amounts of data transmitted through the network that could be eliminated through a single request. The subscription framework of SIP was designed to prevent such requests from being necessary but does not permit end users to specify the conditions under which new data would be transmitted.

Modifications to the subscription framework of SIP are proposed in this thesis to permit end users to request an asynchronous and continuous stream of heterogeneous data periodically from multiple sources using a single subscription. Two types of subscriptions are required for data consumers to automatically receive data: a subscription for data source descriptions and a subscription for data values. No data needs to be transmitted to request a subscription for data source descriptions. Therefore support for a new subscription data type is one proposed modification for the SIP subscription framework. However, a subscrip-

SUBSCRIBE sip:fadi@uottawa.ca SIP/2.0
Via: SIP/2.0/TCP domain.uottawa.ca:5162;
branch=z9hG4bKivlwn23
Max-Forwards: 70
To: <sip:fadi@uottawa.ca>
From: <sip:ray@uottawa.ca>;tag=21171
Call-ID: fnaw4nli3n203n29n42
CSeq: 3412 SUBSCRIBE
Allow-Events: **data-description**
Expires: 3600
Contact: <sip:ray@uottawa.ca>
Event: **data-description**

Figure 6.2: A SIP SUBSCRIBE packet used to request a subscription for data source descriptions.

tion for data values requires data to be transmitted from the data consumer to the broker like the rate of repetition and the desired data values. Therefore the addition of a payload to the SUBSCRIBE packet along with a new subscription data type is a second proposed modification to the SIP subscription framework.

The modification required to request a subscription for data source descriptions is illustrated in the SUBSCRIBE packet of Fig. 6.2. Modifications to the SUBSCRIBE packet are shown in bold face. The data type 'data-description' is used with the SUBSCRIBE packet to indicate that data source descriptions should be transmitted with notifications if the session is accepted. Support for this data type must implemented into any RMCS used to exchange data between end users.

The modification required to request a subscription for data values is illustrated in the SUBSCRIBE packet of Fig. 6.3. Modifications to the SUBSCRIBE packet are shown in bold face. The data type 'data-value is used with the SUBSCRIBE packet to indicate that data source descriptions should be transmitted with notifications if the session is accepted. All the specifications of the data value request are attached in a payload. Support for this data type must implemented into any RMCS used to exchange data between end users.

SUBSCRIBE sip:fadi@uottawa.ca SIP/2.0
Via: SIP/2.0/TCP domain.uottawa.ca:5162;
branch=z9hG4bKivlwn23
Max-Forwards: 70
To: <sip:fadi@uottawa.ca>
From: <sip:ray@uottawa.ca>;tag=21171
Call-ID: fnaw4nli3n203n29n42
CSeq: 3412 SUBSCRIBE
Allow-Events: **data-value**
Expires: 3600
Contact: <sip:ray@uottawa.ca>
Event: **data-value**
Content-Type: text/plain
Content-Length: 40

Data exchanged between two architecture components.

Figure 6.3: A SIP SUBSCRIBE packet used to request a subscription for data values.

6.2 RMCS Messages

This section defines all the messages that can be generated and parsed by the RMCS to exchange data with an architecture component. Requests are transmitted from a component to the RMCS to execute a command that typically requires the generation and transmission of a SIP packet. Responses are generated by a RMCS and transmitted to components to acknowledge a request with specific information. Alerts are used for the asynchronous transmission of data from the RMCS to a component.

6.2.1 Requests

Create Instant Message

This request contains the information required to generate and transmit a SIP MESSAGE packet from the RMCS.

Components

- To: An index indicating the individual transmitting the packet.
- From: An index indicating the individual receiving the packet.

- Application Session Id: An index providing an identifier of the application level session used to transmit the instant message.
- Message Buffer: A sequence of characters describing the instant message to be transmitted with the SIP MESSAGE packet.

Create Notification

This request contains the information required to generate and transmit a SIP NOTIFY packet from the RMCS.

Components

- Application Session Id: An index providing an identifier of the application level session used to transmit the notification.
- SIP Interface Session Id: An index providing an identifier of the SIP subscription used to transmit the notification.
- State: An index describing the state of the notification.
- Notification Buffer: A sequence of characters describing the notification to be transmitted with the SIP NOTIFY packet.

Create Subscription

This request contains the information required to generate and transmit a SIP SUBSCRIBE packet from the RMCS.

Components

- To: An index indicating the individual transmitting the packet.
- From: An index indicating the individual receiving the packet.
- Application Session Id: An index providing an identifier of the application level session used to transmit the subscription request.
- Expires: An integer describing the duration of the subscription in seconds.

- Subscription Buffer: A sequence of characters describing the notification to be transmitted with the SIP SUBSCRIBE packet.

Terminate Subscription

This request contains the information required to terminate an existing SIP subscription.

Components

- Application Session Id: An index providing an identifier of the application level session used to create a SIP session.
- SIP Interface Session Id: An index providing an identifier of the SIP subscription to terminate.

6.2.2 Responses

Session Created

This response contains information about a SIP session that has been created.

Components

- Application Session Id: An index providing an identifier of the application level session used to create a SIP session.
- SIP Interface Session Id: An index providing an identifier of the SIP subscription that was successfully created.

6.2.3 Alerts

New Instant Message

This alert is used to transmit application data from the RMCS to a component received through a SIP MESSAGE packet.

Components

- To: An index indicating the individual that transmitted the packet.

- From: An index indicating the individual that received the packet.
- Message Buffer: A sequence of characters describing the data transmitted through a SIP MESSAGE packet.

New Notification

This alert is used to transmit application data from the RMCS to a component received through a SIP NOTIFY packet.

Components

- SIP Interface Session Id: An index providing an identifier of the SIP subscription used to transmit the notification.
- Notification Buffer: A sequence of characters describing the data transmitted through a SIP NOTIFY packet.

New Subscription

This alert is used to transmit application data from the RMCS to a component received through a SIP SUBSCRIBE packet.

Components

- Expires: An integer describing the duration of the subscription in seconds.
- SIP Interface Session Id: An index providing an identifier of the SIP subscription used to transmit the subscription.
- Notification Buffer: A sequence of characters describing the data transmitted through a SIP SUBSCRIBE packet.

Session Terminated

This alert contains information about a SIP session that has been terminated.

Components

- SIP Interface Session Id: An index providing an identifier of the SIP session that was successfully terminated.

Subscription Terminated

This alert contains information about a SIP subscription that has been terminated.

Components

- SIP Interface Session Id: An index providing an identifier of the SIP subscription that was successfully terminated.

6.3 Use Cases

This section illustrates the use cases describing how the RMCS is used to interact with its corresponding component and exchange data with other RMCS through an IMS network. Use cases describe how the instant messages, notifications and subscription are used in all applicable context to transmit data, deliver data, manage sessions and manage subscriptions.

6.3.1 Transmit Instant Message

This use case describes the scenario where a component uses the RMCS to transmit data through the IMS network through using an instant message.

Goals

To permit the transmission data through an IMS network using SIP MESSAGE packets.

Preconditions

- All communicating components and their corresponding RMCS must be functional.

Related Use Cases

- Transmit Notification.
- Transmit Subscription.

Steps

1. A command is transmitted by the component requiring data to be transmitted to another component across the IMS network using an instant message. This command may be sent by an end user to a data consumer or data producer or generated in a broker after processing data from the IMS network.

2. The necessary data is formed and transmitted to the RMCS through a 'Create Instant Message' request.

3. A SIP MESSAGE packet is created in the RMCS based on the request received from the component. The creation of a SIP packet prompts the creation of a SIP session.

4. The generated SIP MESSAGE packet is transmitted through the SIP network.

5. A 'Session Created' response is transmitted from the RMCS to the component.

6. Information about the SIP session is updated in the component and displayed to the end user.

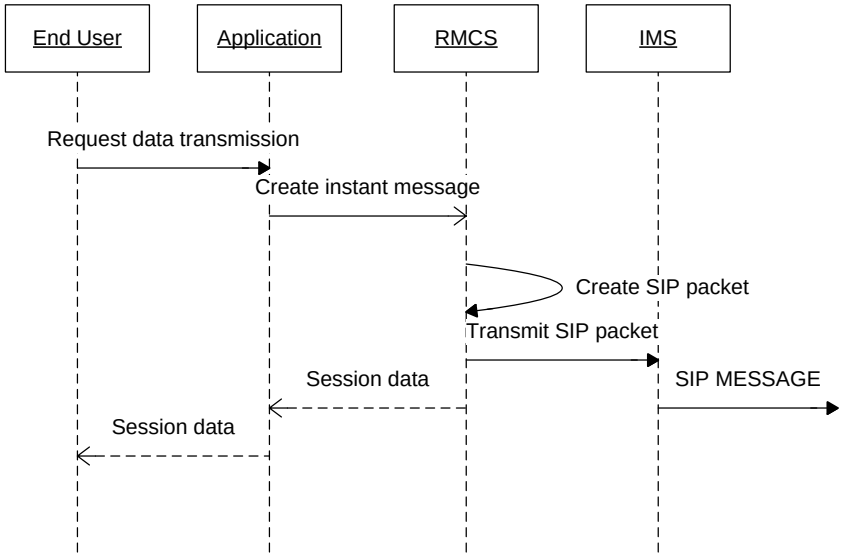

Figure 6.4: The Message Exchange for the Transmission of Data Using a SIP MESSAGE Packet.

Postconditions

- Data has been successfully transmitted through the IMS network between components using a SIP MESSAGE packet.

6.3.2 Transmit Notification

This use case describes the scenario where the broker transmit a notification to a data consumer using the RMCS to transmit data through the IMS network through using a notification.

Goals

To permit a broker to transmission notification data through an IMS network using SIP NOTIFY packets.

Preconditions

- All communicating components and their corresponding RMCS must be functional.
- A subscription must have been successfully established between a data consumer and the broker.

Related Use Cases

- Transmit Instant Message.
- Transmit Subscription.

Steps

1. A determination is made in the broker that data must be transmitted to a data consumer through a subscription. The necessary data is written to a 'Create Notification' request and transmitted to the RMCS.
2. A SIP NOTIFY packet is created in the RMCS based on the request received from the broker. The creation of a SIP packet prompts the creation of a SIP session.
3. The generated SIP NOTIFY packet is transmitted through the SIP network.
4. A 'Session Created' response is transmitted from the RMCS to the component.

Postconditions

- Notification data has been successfully transmitted through the IMS network between a broker and a data consumer.

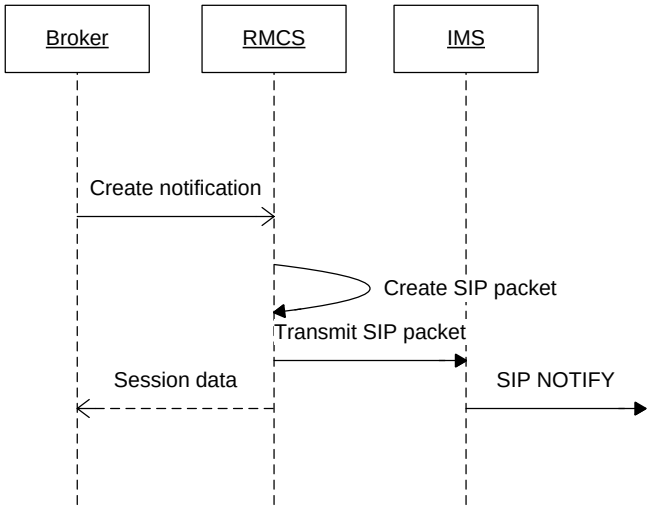

Figure 6.5: The Message Exchange for the Transmission of Data Using a SIP NOTIFY Packet.

6.3.3 Transmit Subscription

This use case describes the scenario where a data consumer uses the RMCS to transmit data through the IMS network through using a subscription.

Goals

To permit the transmission data through an IMS network using SIP SUBSCRIBE packets.

Preconditions

- All communicating components and their corresponding RMCS must be functional.

Related Use Cases

- Transmit Instant Message.
- Transmit Notification.

Steps

1. A command is transmitted by the data consumer requiring data to be transmitted to the broker across the IMS network using a subscription.
2. The necessary data is formed and transmitted to the RMCS through a 'Create Subscription' request.

3. A SIP SUBSCRIBE packet is created in the RMCS based on the request received from the data consumer. The creation of a SIP packet prompts the creation of a SIP session.

4. The generated SIP SUBSCRIBE packet is transmitted through the SIP network.

5. A 'Session Created' response is transmitted from the RMCS to the data consumer.

6. Information about the SIP session is updated in the data consumer and displayed to the end user.

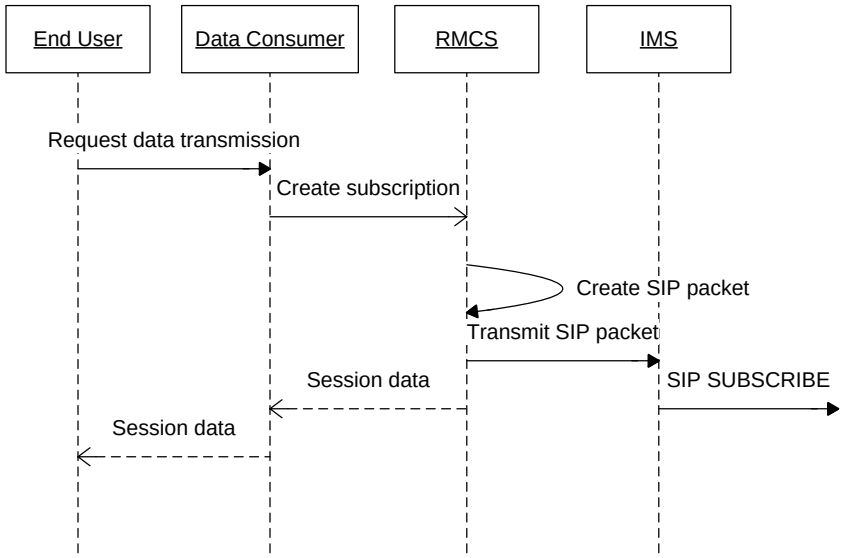

Figure 6.6: The Message Exchange for the Transmission of Data Using a SIP SUBSCRIBE Packet.

Postconditions

- Data has been successfully transmitted through the IMS network between a data consumer and the broker using a SIP SUBSCRIBE packet.

6.3.4 Instant Message Delivery

This use case describes the scenario where application level data has been received through the IMS network using an instant message.

Goals

To deliver the data received from a SIP MESSAGE packet to a component so it can be processed.

Preconditions

- All communicating components and their corresponding RMCS must be functional.

Related Use Cases

- Notification Delivery.
- Subscription Delivery.

Steps

1. A SIP MESSAGE packet is received from the IMS network and delivered to the RMCS.
2. The SIP packet is processed and the application data transmitted through the packet is extracted.
3. A 'New Instant Message' alert is generated and delivered to the component.
4. The alert is processed, all appropriate responses are generated and feedback is provided to the end user where appropriate and possible.

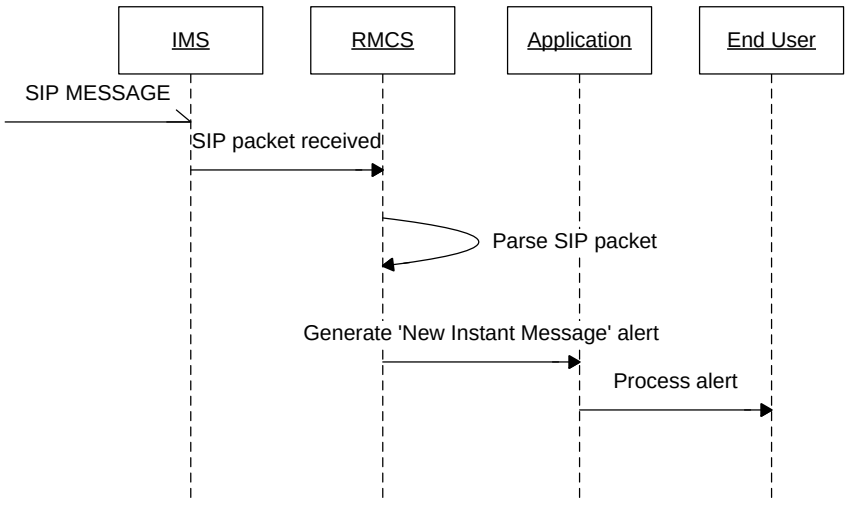

Figure 6.7: The Message Exchange for the Delivery of Data Using a SIP MESSAGE Packet.

Postconditions

- Application data has been successfully received by the RMCS through a SIP MESSAGE packet.

6.3.5 Notification Delivery

This use case describes the scenario where application level data has been received through the IMS network using a notification.

Goals

To deliver the data received from a SIP NOTIFY packet to a component so it can be processed.

Preconditions

- All communicating components and their corresponding RMCS must be functional.
- A subscription must have been successfully established between a data consumer and the broker.

Related Use Cases

- Instant Message Delivery.
- Subscription Delivery.

Steps

1. A SIP NOTIFY packet is received from the IMS network and delivered to the RMCS.
2. The SIP packet is processed and the application data transmitted through the packet is extracted.
3. A 'New Notification' alert is generated and delivered to the component.
4. The alert is processed, all appropriate responses are generated and feedback is provided to the end user where appropriate and possible.

Postconditions

- Application data has been successfully received by the RMCS through a SIP NOTIFY packet.

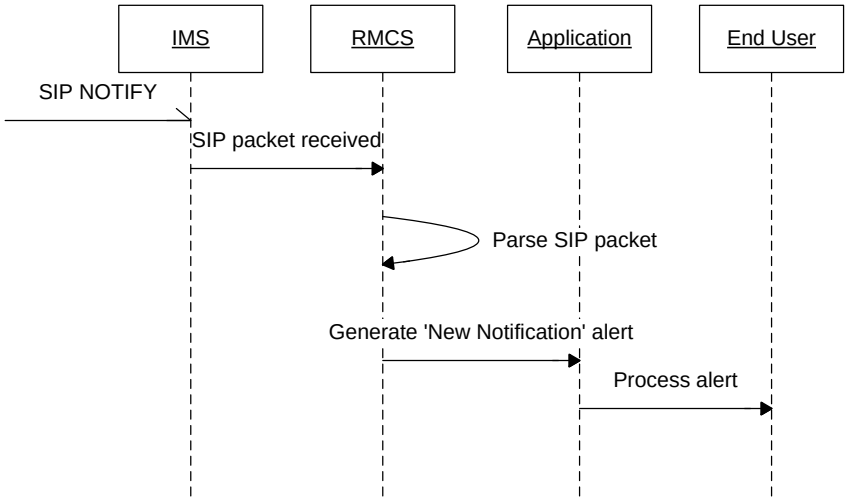

Figure 6.8: The Message Exchange for the Delivery of Data Using a SIP NOTIFY Packet.

6.3.6 Subscription Delivery

This use case describes the scenario where application level data has been received through the IMS network using a subscription.

Goals

To deliver the data received from a SIP SUBSCRIBE packet to a component so it can be processed.

Preconditions

- All communicating components and their corresponding RMCS must be functional.

Related Use Cases

- Instant Message Delivery.
- Notification Delivery.

Steps

1. A SIP SUBSCRIBE packet is received from the IMS network and delivered to the RMCS.
2. The SIP packet is processed and the application data transmitted through the packet is extracted.

3. A 'New Subscription' alert is generated and delivered to the component.

4. The alert is processed, all appropriate responses are generated and feedback is provided to the end user where appropriate and possible.

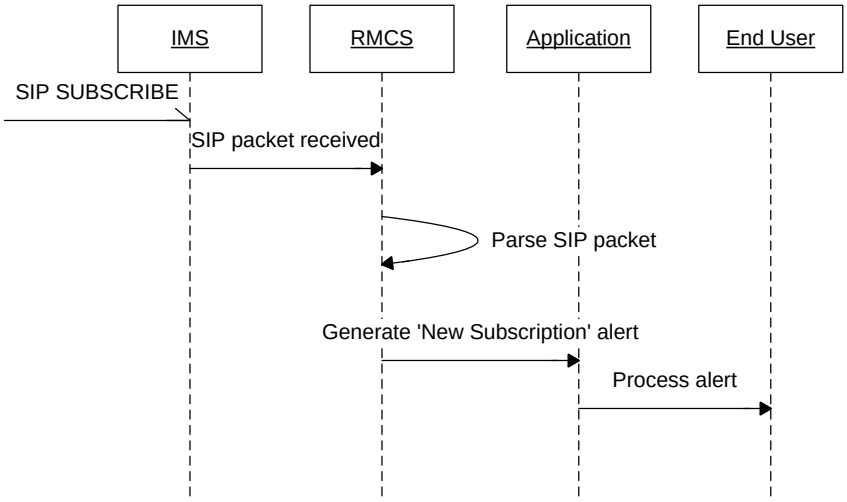

Figure 6.9: The Message Exchange for the Delivery of Data Using a SIP SUBSCRIBE Packet.

Postconditions

- Application data has been successfully received by the RMCS through a SIP SUBSCRIBE packet.

6.3.7 Session Termination from the Network

This use case describes the scenario where a SIP session is terminated in the RMCS. This scenario applies to circumstances where SIP messages have been acknowledged through a SIP 200 OK packet and the session used to transmit the original message is terminated.

Goals

To terminate a SIP session when a SIP 200 OK message has been received from the network and send information about the terminated session to the component.

Preconditions

- All communicating components and their corresponding RMCS must be functional.

- A SIP session must have been successfully established between two RMCS.

Related Use Cases

- Session Termination from a Component.

Steps

1. A SIP 200 OK packet is received from the IMS network and delivered to the RMCS.
2. The SIP 200 OK packet is processed and the session specified by the packet is terminated.
3. A 'Session Terminated' alert is generated and transmitted to the component.
4. Information about the terminated SIP session is displayed to the end user.

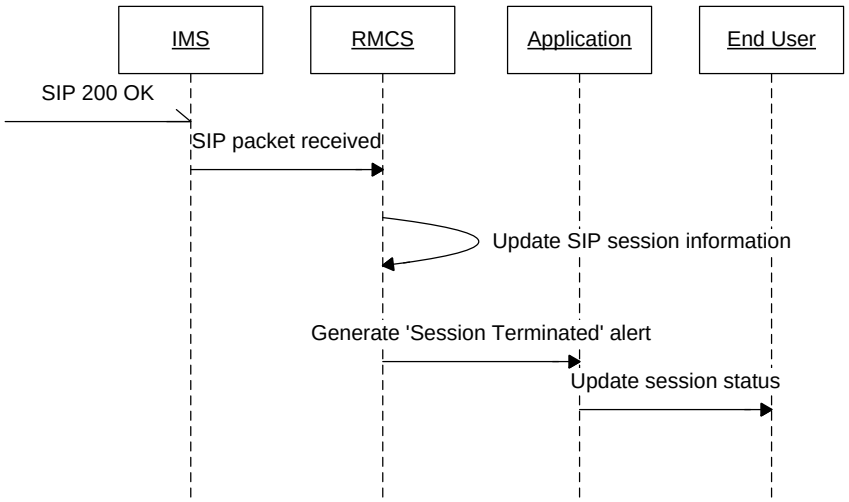

Figure 6.10: Message Exchange for the Termination of a SIP Session based on a SIP packet.

Postconditions

- A SIP session has been successfully terminated in a RMCS based on the reception of a SIP 200 OK packet.
- The end user is automatically informed when the SIP session is terminated.

6.3.8 Session Termination from the Component

This use case describes the scenario where a SIP session is terminated in the RMCS due to a command transmitted from the component.

Goals

To permit the termination of a SIP session by an end user or through a component after receiving a specific sequence of commands.

Preconditions

- All communicating components and their corresponding RMCS must be functional.
- A session must have been successfully established between two RMCS.

Related Use Cases

- Session Termination from the Network.

Steps

1. A command is transmitted to the component that requests the termination of a SIP session that was established to exchange data across the IMS network. SIP sessions that can be terminated asynchronously are typically subscriptions established to exchange multiple pieces of application data over an extended period of time. This command may be sent by an end user to a data consumer or data producer or generated in a broker based on a timeout.
2. A 'Terminate Subscription' request is generated and transmitted from the component to the RMCS.
3. The specified SIP session is terminated. A SIP NOTIFY or SUBSCRIBE packet is generated so it can transmitted through the IMS network for session termination at the other RMCS. SIP SUBSCRIBE packets are transmitted by components that sent the original subscription request while SIP NOTIFY packets are transmitted by components where the subscription request was accepted.

4. A 'Session Created' response is generated (for the transmission of the SIP NOTIFY or SUBSCRIBE packet) and transmitted to the component.

5. Information about the SIP session is updated in the component and displayed to the end user.

6. A SIP 200 OK message is received for the SIP NOTIFY or SUBSCRIBE message that was transmitted to terminate the subscription.

7. A 'Subscription Terminated' alert is generated and transmitted to the component.

8. Information is transmitted to the end user indicating that the requested subscription has been terminated.

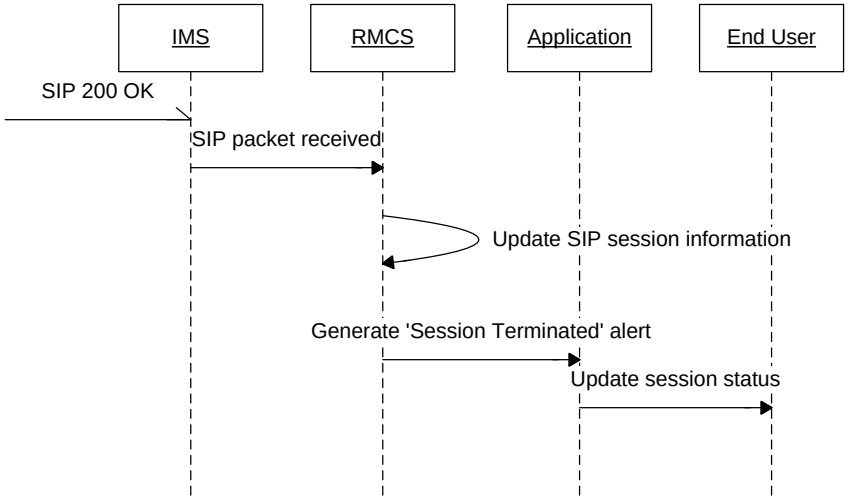

Figure 6.11: Message Exchange for the Termination of a SIP Session based on a Component Command.

Postconditions

- A SIP session has been successfully terminated in a RMCS based on a request submitted through a component.

- The end user is automatically informed when the SIP session is terminated.

6.3.9 Subscription Termination

This use case describes the scenario where a SIP subscription is terminated in the RMCS.

Goals

To terminate a SIP session when a SIP NOTIFY or SUBSCRIBE message has been received from the network.

Preconditions

- All communicating components and their corresponding RMCS must be functional.
- A SIP session must have been successfully established between two RMCS.

Related Use Cases

- Session Termination from an Component.

Steps

1. A SIP NOTIFY or SUBSCRIBE packet is received from the IMS network and delivered to the RMCS.
2. The SIP packet is processed and the subscription specified by the packet is terminated.
3. A 'Subscription Terminated' alert is generated and transmitted to the component.
4. Information about the terminated SIP subscription is displayed to the end user.

Postconditions

- A SIP subscription has been successfully terminated in a RMCS.
- The end user is automatically informed when the SIP subscription is terminated.

6.4 Conclusion

The communications framework required to exchange heterogeneous data through the communications architecture of this thesis was described in this chapter. The SIP packets required are SIP MESSAGE, NOTIFY and SUBSCRIBE. The SIP MESSAGE packet is used

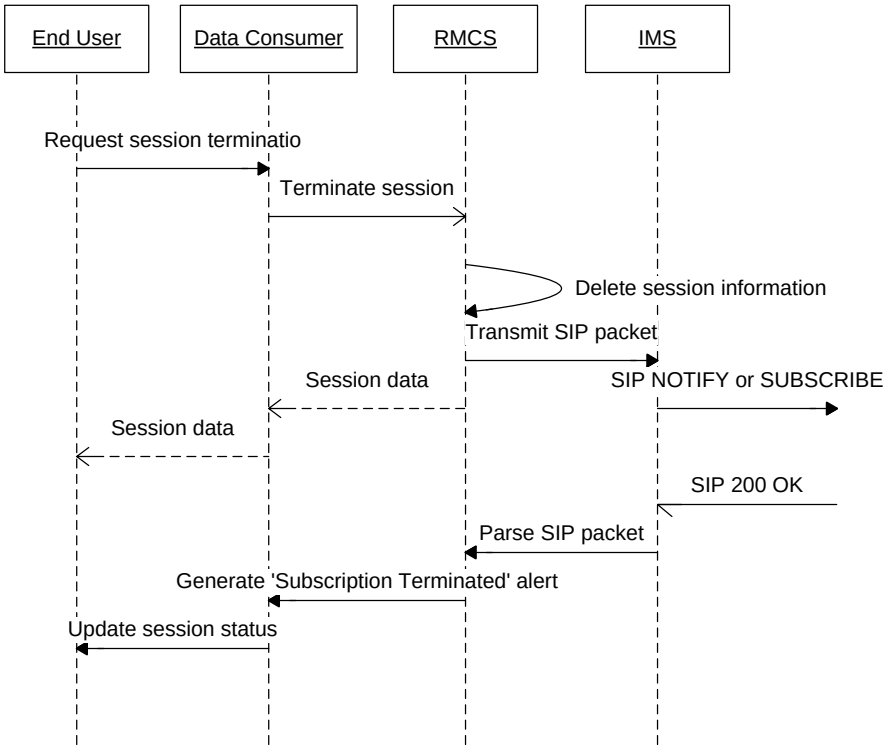

Figure 6.12: Message Exchange for the Termination of a SIP Subscription.

to transmit data synchronously through the instant messaging framework of SIP and IMS. The SIP NOTIFY and SUBSCRIBE packets are used to create sessions and transmit data asynchronously through notifications. The subscription framework of SIP does not have any mechanisms to request heterogeneous data or transmit data through a payload. Therefore, modifications are proposed to implement these features so the requirements of the communications architecture can be fulfilled. Numerous requests, responses and alerts are defined so components can exchange data with a RMCS to execute various use cases related to the transmission of data, management of sessions and management of subscriptions.

Chapter 7

Reconfigurable Multimedia Collaborative System

This chapter describes the Reconfigurable Multimedia Collaborative System (RMCS), an embedded system developed for the communications architecture of this thesis. The RMCS is used as an intermediary between architecture components and the IMS network to process SIP data and application level responses. Embedded systems performing similar functions are surveyed before discussing the architecture of the RMCS. Subsequent sections discuss all details of the RMCS including its architecture, commands, session management, packet processing facilities and performance.

7.1 SIP Hardware Implementations

Various SIP implementations exist in the literature and commercial products today. However, almost all commercial implementations of SIP are entirely software based, the consequence of which is a significant cost with respect to real-time performance. Open source SIP implementations like oSIP [28] are also entirely implemented in software. Lower level protocols can take in the order of several milliseconds to correctly process information. Significantly more time is required to process SIP packets because SIP is an ASCII-based protocol. That kind of delay can lead to substantial performance degradation and lost data in high speed networks. Embedded system implementations have been researched to minimize the

delay required to process SIP packets.

An embedded software implementation for VoIP is presented in [55]. Signaling control functions of SIP are integrated in the proposed embedded system platform. Additional features like voice conferences, call waiting and call on-hold can be built into the system based through SIP but no facilities exist to implement any functionality in hardware.

A microcontroller based phone is described in [64] with SIP based signaling. The microcontroller includes a real-time operating system to optimize the execution of software but provided no facilities to implement any functionality directly in hardware. Similar work was performed in [85] with the development of a VoIP gateway name the Enhanced Multi Media Adaptor (EMMA). EMMA was implemented as a system on a programmable chip (SOPC) with an Altera FPGA. With this implementation, numerous hardware blocks were used to directly implement accelerated voice packetization. However with both projects session establishment was provided through a multi threaded, real-time operating system implementation of SIP.

The work in [81] describes a reconfigurable hardware implementation of SIP. It describes a hardware processor that parses SIP headers and includes checksums for IP and TCP packets. However the processor lacks the functionality to interpret SIP content. No description of an interface is presented to allow for the processor's integration in a software environment. Therefore, the processor cannot be used directly to implement SIP for robust multimedia applications.

The absence of a robust embedded system implementation of SIP in the literature and commercial products illustrates the novelty introduced by the RMCS to the domain of computer communications and embedded device development. The RMCS is described in [58] through a patent application and provides the foundation for the work described in [27], [86], [87] and [88].

7.2 High Level System Description

The Reconfigurable Multimedia Collaborative System (RMCS) is an embedded system used to generate SIP packets, parse SIP packets and manage SIP sessions according to the re-

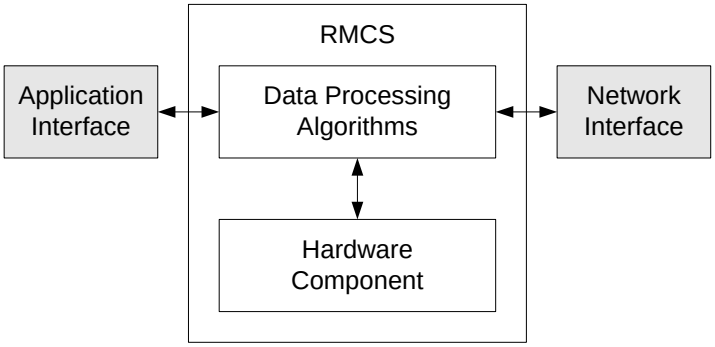

Figure 7.1: The High Level Architecture of the RMCS.

quirements described earlier in this thesis. The following subsections describe the details of the RMCS in significant detail to illustrate how these actions are performed. Please refer to Appendix B for a complete list of RMCS commands.

7.2.1 High Level Architecture

Fig. 7.1 illustrates the architecture of the RMCS. Data processing algorithms are implemented to process application data and network data. Sockets are used by the RMCS to establish connections with remote devices and exchange data through the Internet. Messages are received from end users or other clients through UDP and TCP sockets. Multimedia information is received by UDP from an end user. This information is processed to determine its destination and is forwarded in an unaltered state. RMCS commands and text based messages may be sent through TCP connections. Responses are sent back using TCP.

7.2.2 Session Management

Table 7.1 describes how sessions are maintained in the SIP Session Data component of the RMCS. Each session is represented by a row in the table comprised of an incoming SIP packet and an outgoing SIP packet. All combinations of incoming and outgoing packets supported by the RMCS are illustrated in Table 7.1. The reception of a 200 OK packet prompts the termination of a session in all cases except where a subscription has been created. Subscriptions are represented by a persistent SUBSCRIBE egress packet and 200 OK incoming packet that are not removed unless the RMCS receives a command to terminate

a subscription. All notifications transmitted through a subscription retrieve the necessary data from the SUBSCRIBE egress packet to generate the required NOTIFY packet.

Table 7.1: A Description of SIP Session Data Maintained in the RMCS

Incoming (ingress) packet	Outgoing (egress) packet	Session Description
200 OK	MESSAGE	Transmitting an instant message
200 OK	NOTIFY	Transmitting a notification
200 OK	NOTIFY	Terminating a subscription initiated through another RMCS
200 OK	SUBSCRIBE	Initiating a subscription with another RMCS
200 OK	SUBSCRIBE	Terminating a subscription initiated with this RMCS
MESSAGE	200 OK	Receiving an instant message
NOTIFY	200 OK	Receiving a notification
NOTIFY	200 OK	Receiving a session termination request
SUBSCRIBE	200 OK	Receiving a subscription request
SUBSCRIBE	200 OK	Receiving a session termination request

7.3 Data Processing Algorithms

Fig. 7.2 illustrates the algorithm required to process data received through the application interface. When a message is first received its message type is first verified before further actions are taken. When a request has been received to create an instant message it is generated using all the information received from the application. When this message is completed it is transmitted through the IMS network. After the instant message has been transmitted to the network a response indicating the creation of a session is created and transmitted to the application. When a subscription request has been submitted to the RMCS, a SIP SUBSCRIBE packet is generated and transmitted to the IMS network before a response is sent to the application about the creation of a new session. The same procedure is performed when a notification request is received from an application.

A subscription termination request requires that information about the existing subscription be read before the appropriate SIP packet is transmitted to the network. When a subscription termination request is processed at the SIP interface where the subscription was created a SIP SUBSCRIBE packet must be generated. Otherwise a SIP NOTIFY message

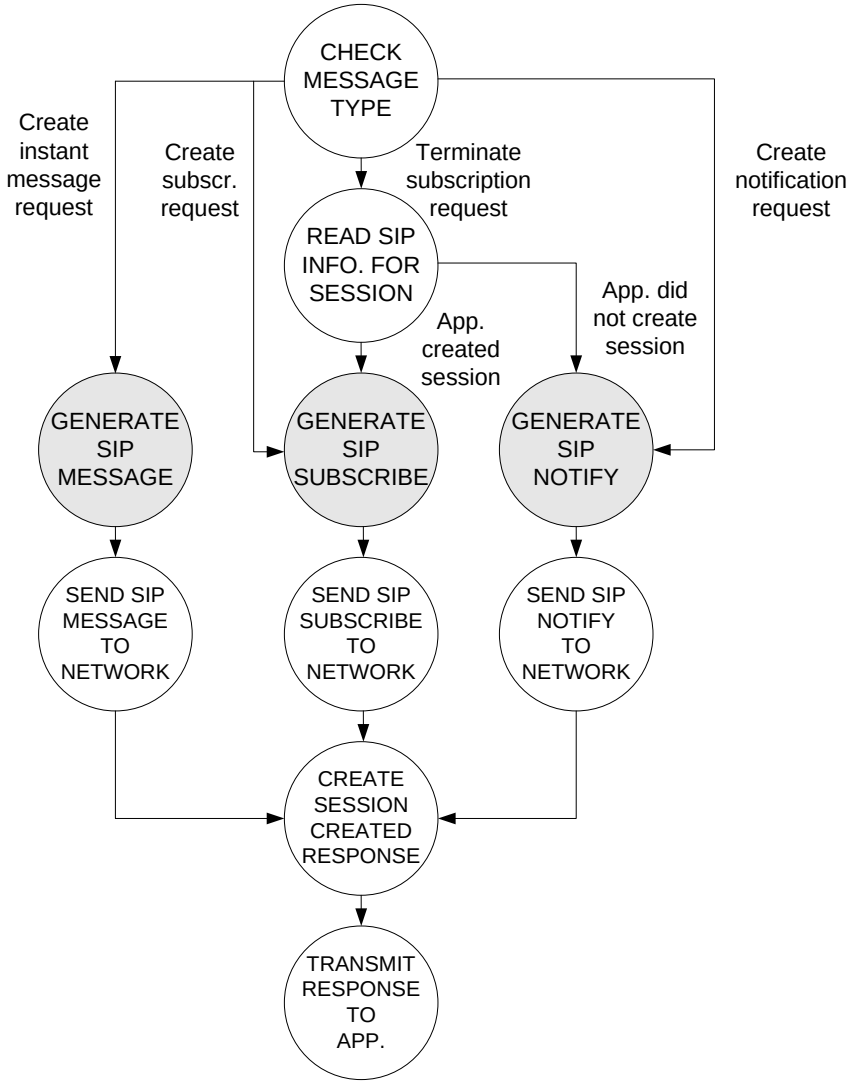

Figure 7.2: The Algorithm of the SIP Interface to Process Application Messages.

must be generated. When the correct SIP packet has been generated it is transmitted to the IMS network before a response is transmitted to the application. Information about SIP subscriptions are contained in the RMCS so this information must be read by the data processing algorithm before further action is taken. All SIP packets and their responses must be generated and read by data processing algorithms before they can be transmitted to the IMS network. The grey coloured states demonstrate operations that are be performed using the hardware component of the RMCS.

Fig. 7.3 illustrates the algorithm that is performed when data is received from the IMS

network. Every packet of data received from the network is presumed to be in the SIP format and is parsed accordingly. Any syntactical errors discovered while parsing the packet automatically cause termination of the algorithm. If the packet is successfully parsed the type of SIP is checked to determine the appropriate course of action.

When a 200 OK packet has been successfully parsed from the IMS network the original request for the 200 OK response must be ascertained before performing the correct sequence of actions. When the 200 OK packet has been received for a SIP SUBSCRIBE packet the original expires value is checked to determine if the original message was sent to terminate or create a SIP subscription. If the expires value was zero, then it is presumed that the original SUBSCRIBE request was transmitted to terminate a SIP subscription. Therefore the requested session information is removed before a session termination response is transmitted to the application. Information for the session used to transmit the 200 OK packet is also removed and transmitted to the application before the algorithm terminates.

A 200 OK packet for a SIP NOTIFY messages prompt the verification of the subscription state value to determine the next course of action. If the subscription state is 'terminated' then all the actions required to terminate the requested session and incoming session are performed as they are when a subscription is terminated through a SIP SUBSCRIBE message transmission. Otherwise, only the incoming session is terminated before a response is transmitted to the application. Similar actions are performed when a 200 OK packet is received for a SIP MESSAGE packet.

When a SIP SUBSCRIBE packet is received the 'Expires' field value is immediately checked to determine if the value is zero. If the 'Expires' field value is zero then the actions necessary to terminate the desired subscription and incoming session are performed. Otherwise the information retrieved from the packet are used to create a new subscription alert for delivery to the application. After this alert has been transmitted to the application the 200 OK response for the incoming packet is read and transmitted to the IMS network.

The subscription state value of a SIP NOTIFY packet is checked as soon as it has been received by the data processing algorithm. A 'terminated' state prompts the termination of the subscription and incoming session. An 'active' state prompts the generation of a new notification alert and its delivery to the application before a 200 OK response is sent to the

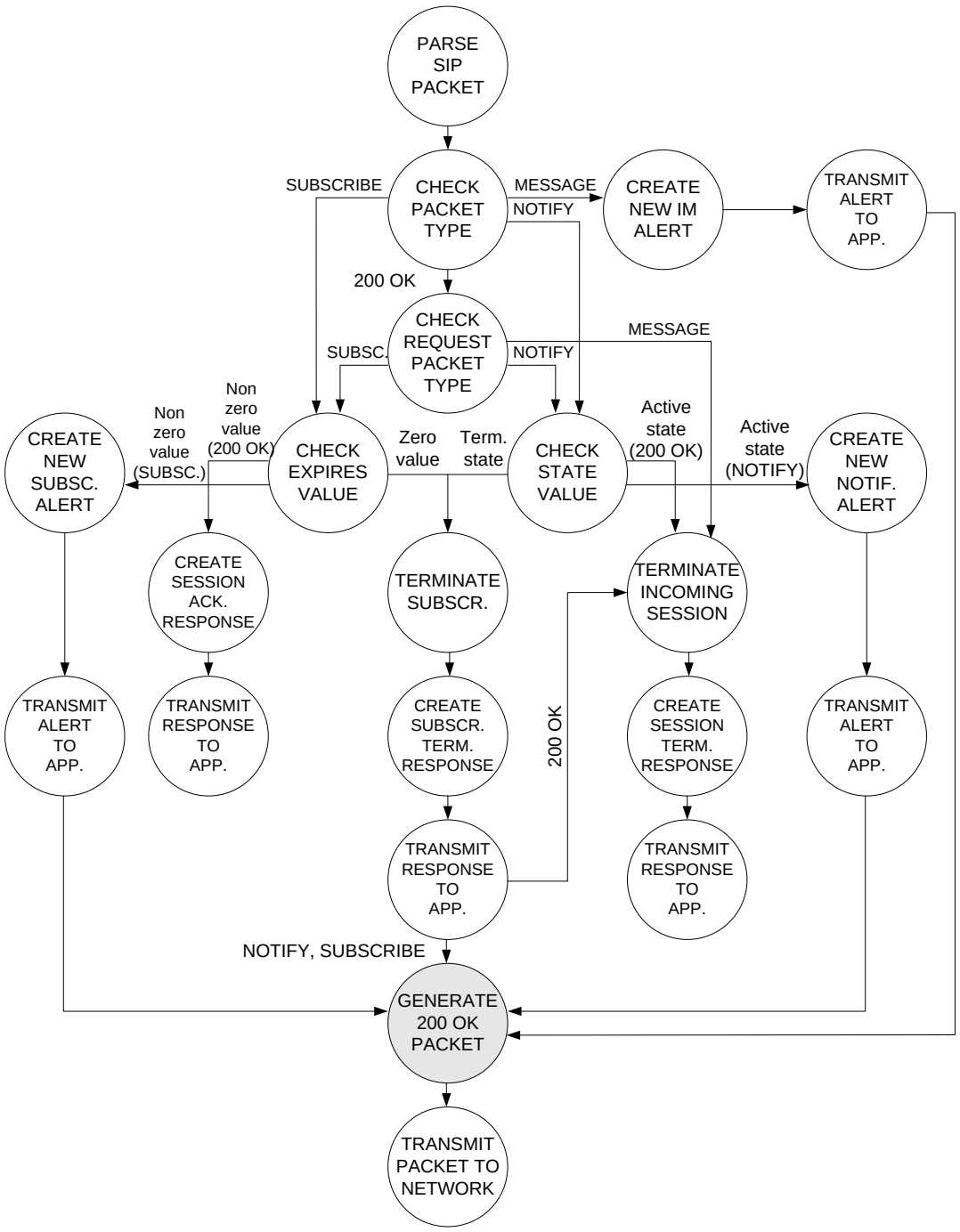

Figure 7.3: The Algorithm of the SIP Interface to Process Network Messages.

IMS network. The actions performed when a SIP MESSAGE is always identical. A new instant message alert is created and transmitted to the application before a 200 OK response is transmitted to the IMS network. The grey coloured states indicate actions that must be

performed using the RMCS.

7.4 Hardware Component

The hardware component of the RMCS is used for SIP packet generation. It is divided into a control unit and data path responsible for high level logic and holding data respectively. The control unit is a collection of finite state machines containing all the logic for the generation and retrieval of SIP packets. The data path of the hardware component is shown in Fig. 7.4.

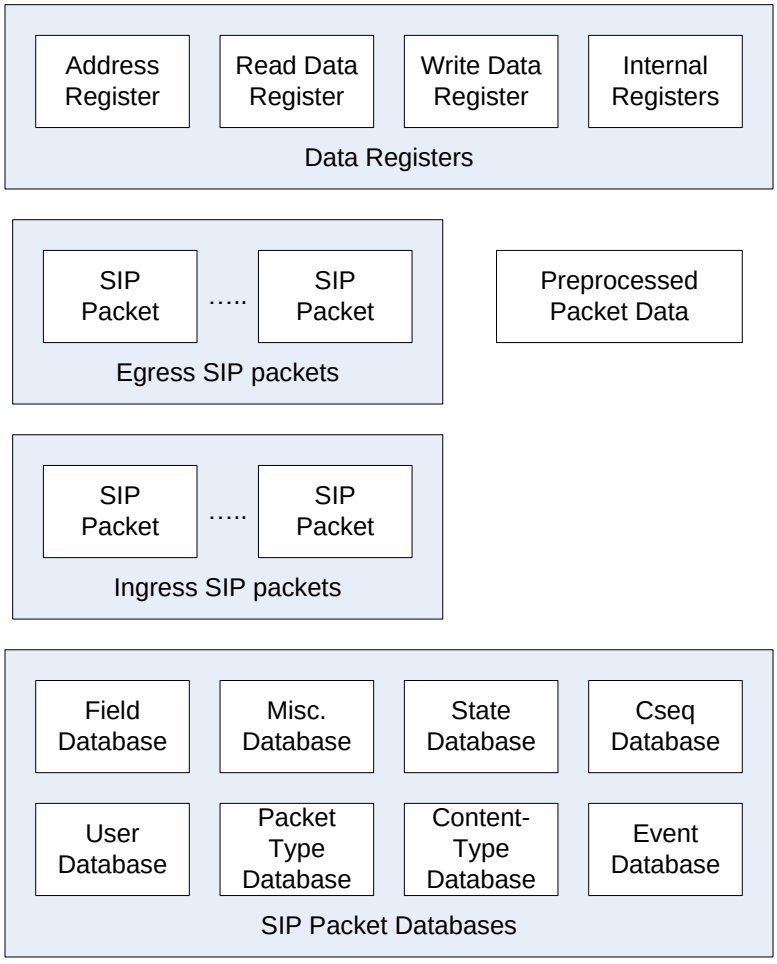

Figure 7.4: The Data Path of the Hardware Component of the RMCS.

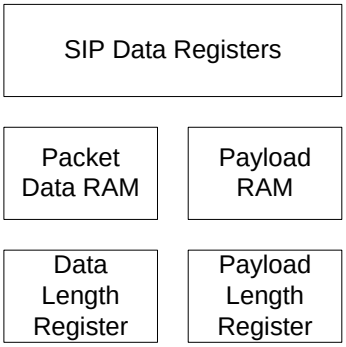

Figure 7.5: The Composition of a SIP Packet in the Hardware Component of the RMCS.

7.4.1 Data Structures

Each command issued to the hardware component is written to a specific address and may contain data. Separate registers are used to hold these data and collectively comprise the interface through which software applications exchange data with the hardware component. Internal registers are also implemented for the packet generation process. A sequence of data structures are available to hold incoming (ingress) or outgoing (egress) SIP packets. The structure of each SIP packet is illustrated in Fig.7.5.

Each SIP packet is comprised of a sequence of registers where one register contains a value representing a specific component of a SIP packet. RAM components are implemented to hold the ASCII representations of the SIP packet and its payload when it is available. Registers are used to indicate the length of these ASCII strings and specify the values of all SIP packet components. Separate registers are used for each of the following SIP packet values:

- The allow-events value.
- The application session value.
- The call identification number.
- The call identification sender.
- The contact identifier.

- The call sequence number.
- The call sequence packet type.
- The event value.
- The expires value.
- The maximum forwards value.
- The packet recipient.
- The packet recipient tag value.
- The packet sender.
- The packet sender tag value.
- The packet type.
- The sequence of available fields.
- The subscription state value.
- The via field branch value.
- The via field domain value.
- The via field protocol value.

Where applicable values are mapped to the index of a database containing the sequence of ASCII characters required to generate the packet. The structure of a database is shown in Fig. 7.6.

Each database has been implemented with ROM components for data components (ASCII strings), the length of each string and an offset in the data component ROM where the string begins. Databases have been implemented for each of the following categories of ASCII strings:

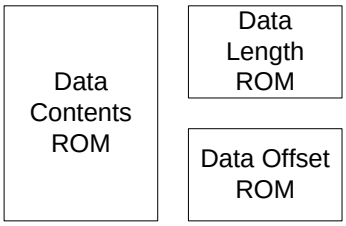

Figure 7.6: The Composition of a Database in the Hardware Component of the RMCS.

- Content Type: This database contains all supported values for the 'Content-Type' field of a SIP packet.
- Call Sequence Number: This database is used to maintain a list of call sequence numbers for each supported SIP packet type.
- Events: This database contains all supported values for the 'Event' and 'Allow-Events' fields of a SIP packet.
- Fields: This database contains all the field values used to begin each line of a SIP packet.
- Packet Type: This database contains all supported SIP packet types used in the SIP packet header and 'Cseq' field.
- State: This database contains all supported values for the 'Subscription-State' field of a SIP packet.
- Users: This database contains all supported values for the 'To', 'From' and 'Contact' fields of a SIP packet in addition to the domain specified in the 'Via' field.
- Miscellaneous Values: This database contains all necessary ASCII strings that are not implemented in other databases.

A component for preprocessed packet data has been implemented to facilitate the integration of packet parsing functionality into the RMCS hardware component as additional research is performed.

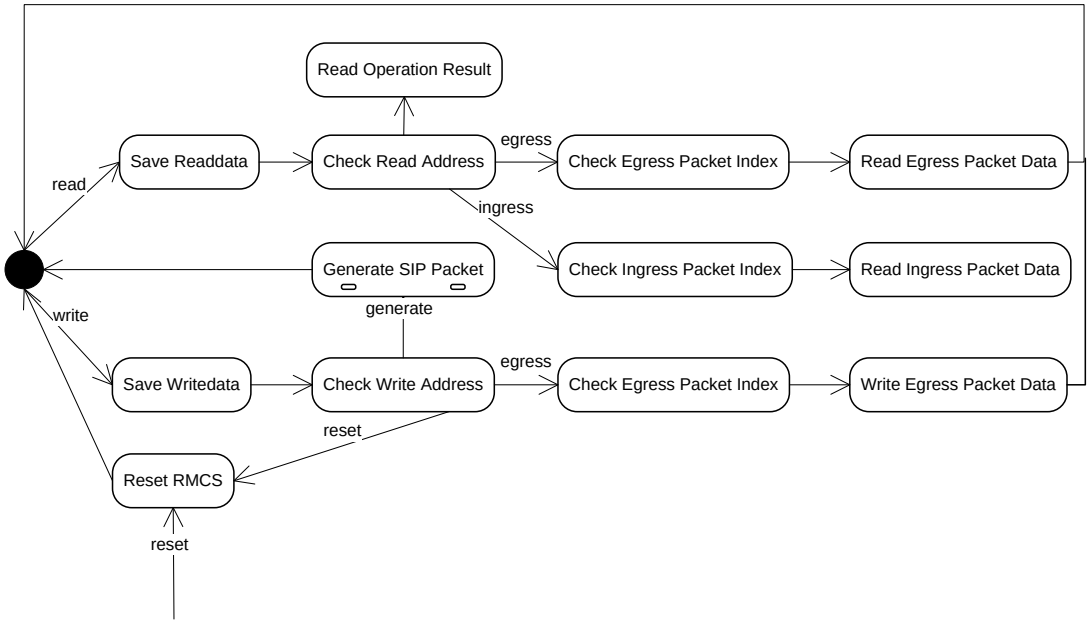

Figure 7.7: The Control Unit of the Hardware Component of the RMCS.

7.4.2 Algorithms

The algorithms presented in this section comprise the control unit of the hardware component for the RMCS. These algorithms are presented as finite state machines and represent the contribution of the thesis required for the generation of SIP packets in hardware.

A finite state machine describing the high level functionality of the control unit is illustrated in Fig. 7.7. The hardware component of the RMCS is reset through an asynchronous command that will prompt the removal of all data stored in the data path. Synchronous commands for the hardware components are read commands (for the retrieval of data from the data path) or write commands (to write data into the data path or begin SIP packet generation). The data and address of the command are verified in each instance before a command can be successfully executed. Read commands prompt the automatic transfer of the requested data from the data path to the hardware component interface. Write commands cause data to be written to a specified location in the data path. A request for the generation of a SIP packets causes the activation of a separate algorithm described by the state 'Generate SIP Packet' whose functionality is described through the sequence of algorithms below.

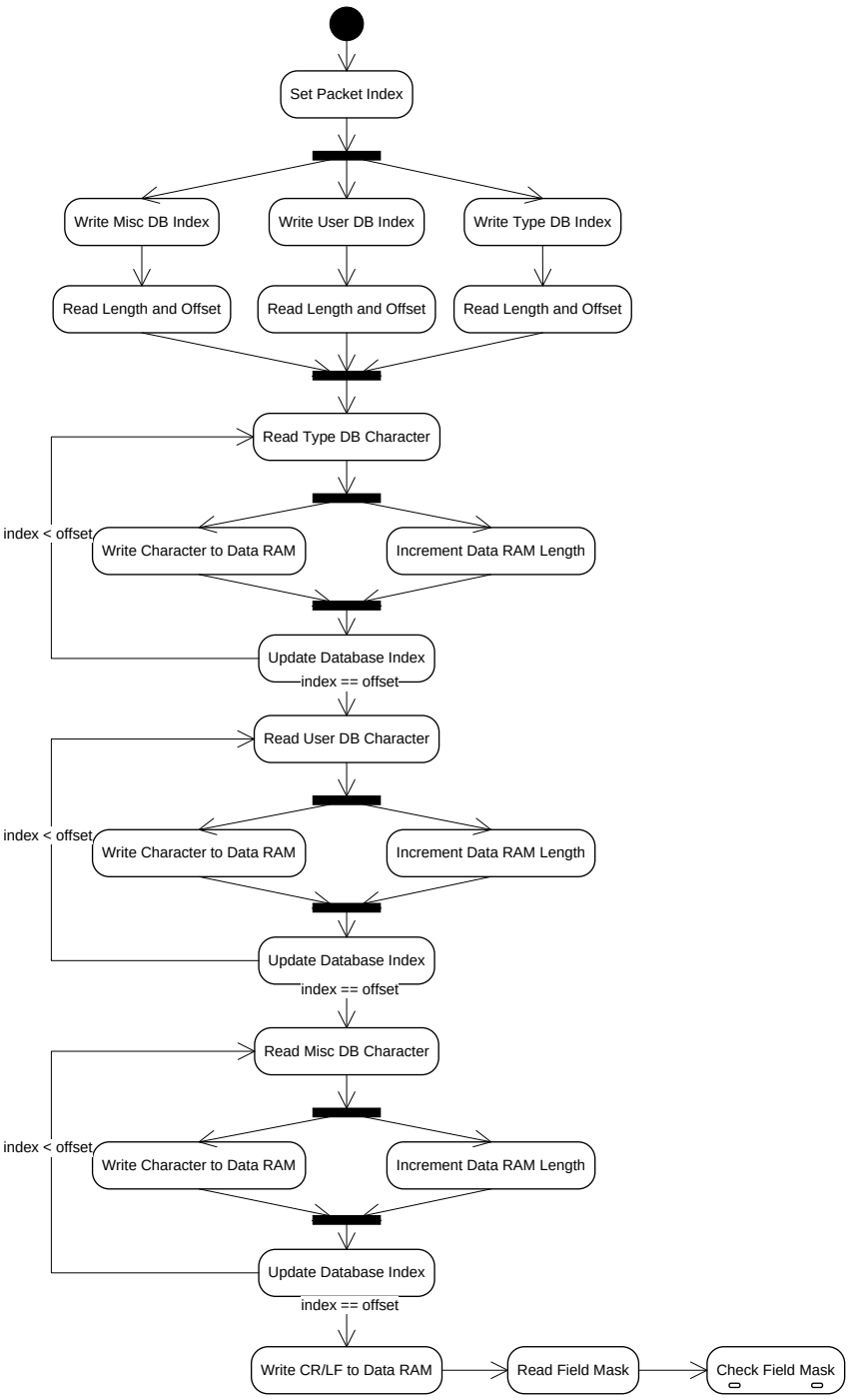

Figure 7.8: The Algorithm Used to Generate a SIP Packet Header in the Hardware Component of the RMCS.

Each line of a SIP packet is generated using a separate algorithm. The first line of a SIP packet is generated using the algorithm described by the finite state machine in Fig.

7.8. The first line of a SIP packet consists of a packet type, packet sender and a protocol denoting the version of SIP to be used for the packet in question. ASCII strings for the SIP packet type, SIP packet sender and protocol are all retrieved from the required databases by writing the necessary index values to retrieve the length and offset values before reading the contents. Once all the necessary ASCII strings have been retrieved they are written into the SIP packet data RAM sequentially before the line is terminated with a carriage return (CR) and line feed (LF) before the SIP field mask sequence is retrieved to determine the sequence of necessary lines in the SIP packet.

After the generation of the first line, a sequence of algorithms are generated to write the remainder of the SIP packet into memory. All other lines in the SIP packet begin with a field value such as 'To', 'From', etc. to denote how the information is to be interpreted. Therefore, all subsequent algorithms executed to generate a line in a SIP packet will contain functionality to read the ASCII representation of the field value from its database and write it to memory before the remainder of the line can be read from an arbitrary collection of databases. The separator between the field value and the rest of the line is always denoted by a colon and a space so these characters are part of the ASCII strings read from the field database to reduce the total time required to generate a SIP message.

The algorithm to generate data for the 'To' line is illustrated in Fig. 7.9. This line of the SIP packet is mandatory and relatively simple as it denotes the source of the SIP packet. This algorithm begins by retrieving the ASCII sequence of the 'To' field from the field database and appending it to the packet data. The ASCII sequence of the SIP packet sender is retrieved from the URL database and written to the packet data. If a 200 OK packet is being generated a tag value is appended to the packet by retrieving the 'tag' ASCII sequence from the database of miscellaneous data and converting the tag integer value to an ASCII string and appending the necessary characters to the packet data. A carriage return and line feed are written to the SIP packet to terminate the 'To' field character sequence. The packet sender and tag value are written to the egress packet of the SIP session data module. The same algorithm is used for the generation of the 'Contact' line.

The generation of the 'From' line of the SIP packet is performed using the algorithm illustrated in Fig. 7.10. This algorithm is similar to the one used in the generation of the

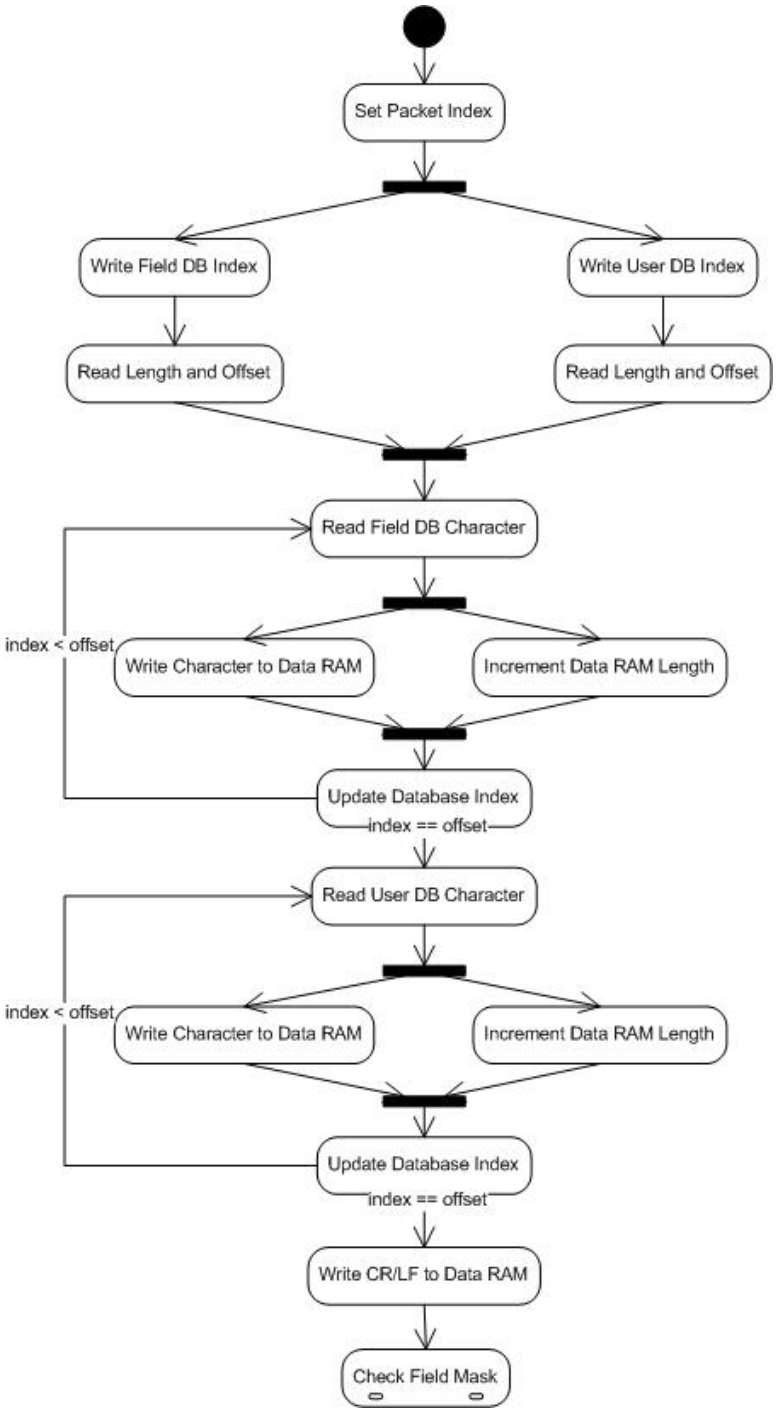

Figure 7.9: The Algorithm Used to Generate the 'To' and 'Contact' lines in the Hardware Component of the RMCS.

'To' line. The URL of the packet recipient is retrieved from the URL database and appended to the SIP packet. The tag value for the 'From' line is not optional for a 200 OK packet as

it is for the 'To' line so a random integer value is generated for the tag, converted into its ASCII representation and appended to the SIP packet along with the necessary string prefix to indicate that it is a tag value.

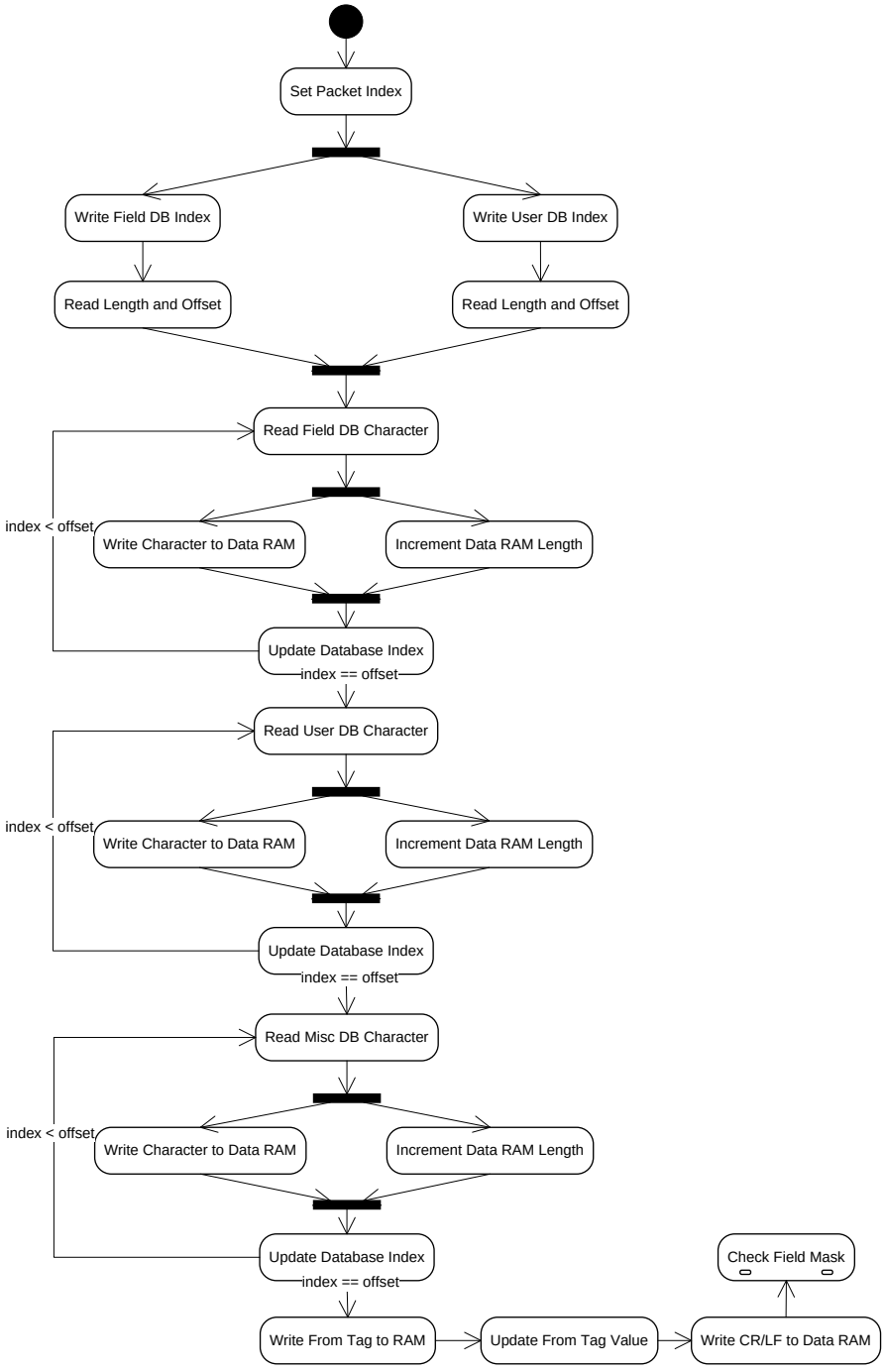

Figure 7.10: The Algorithm Used to Generate the 'From' line in the Hardware Component of the RMCS.

The algorithm of the 'Via' line generation is illustrated in Fig. 7.11. and begins its execution by retrieving the ASCII sequence of the 'Via' field from the field database and writing the resultant characters to the SIP packet along with the colon and space as the field delimiter. The 'Via' line contains the most separate pieces of information compared to other lines in a SIP packet and conversely its algorithm is the most complex. The ASCII sequence of the protocol (TCP or UDP) used to transmit the SIP packet is retrieved from the database of miscellaneous data and written to the packet data with a space unless a 200 OK packet is generated, in which case the protocol is retrieved from the ingress packet. The URL indicating the domain used to transmit the SIP packet is retrieved from the URL database and written to the packet data. A random integer value is generated for the branch and appended to the 'branch' prefix required for a SIP packet if a 200 OK packet is not being generated, otherwise the necessary value is retrieved from the ingress packet. This branch prefix is retrieved from the database of miscellaneous data and written to the SIP packet along with the random value after it has been converted into its ASCII representation. After all the aforementioned values have been written to the packet the line is terminated with a carriage return and a line feed. The protocol, domain and branch values written to the packet are saved with all other SIP packet information.

The algorithm for the generation of the call identifier, denoted with the field 'Call-ID', is demonstrated in Fig. 7.11. The call identifier information written for SIP packets by the RMCS is generated in the form of a random number with the URL of the SIP packet sender appended to the end of the identifier. After the Call-ID field name is retrieved from the field database and the delimiter has been written to the packet, a random number is generated, converted into its ASCII form and written to the packet data. If the packet being generated is a 200 OK packet then the number is retrieved from the packet data of the corresponding ingress packet. After writing the call identifier integer value a '@' character is appended to the SIP packet before writing the ASCII sequence of the appropriate URL. The line is terminated with a carriage return and a line feed before saving the call identifier value to the SIP session data module.

The CSeq line generation algorithm shown in Fig. 7.11 begins by retrieving the Cseq ASCII sequence from the field database and appending the colon and space as the delimiter.

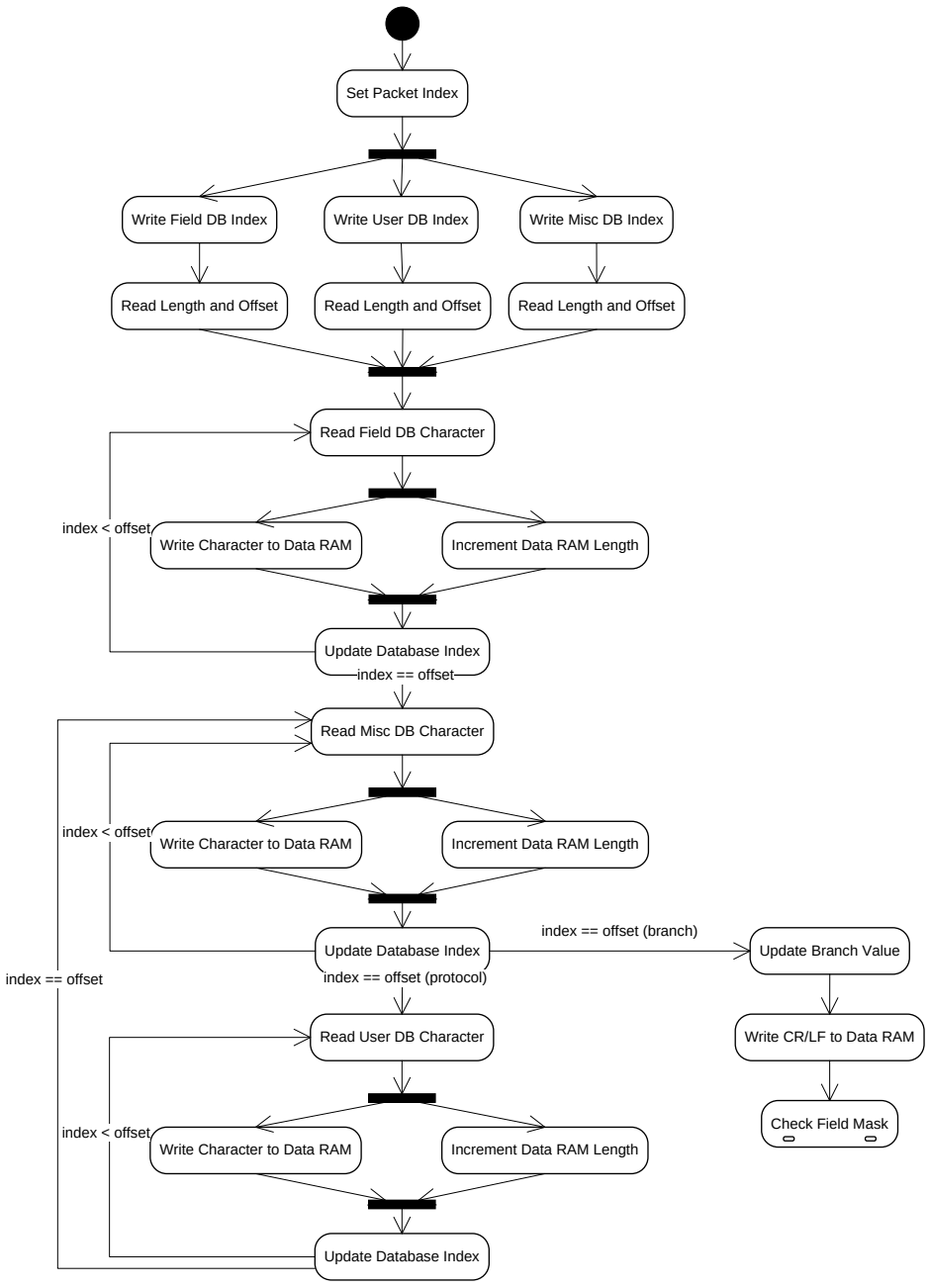

Figure 7.11: The Algorithm Used to Generate the 'Via' line in the Hardware Component of the RMCS.

The CSeq number is used to identify a transaction in a SIP dialog and its value must be maintained with the generation of each SIP packet. If a 200 OK packet is not being generated the Cseq number is retrieved from the Cseq database, converted into its ASCII form, written to the packet and saved to the SIP session data module. The CSeq number

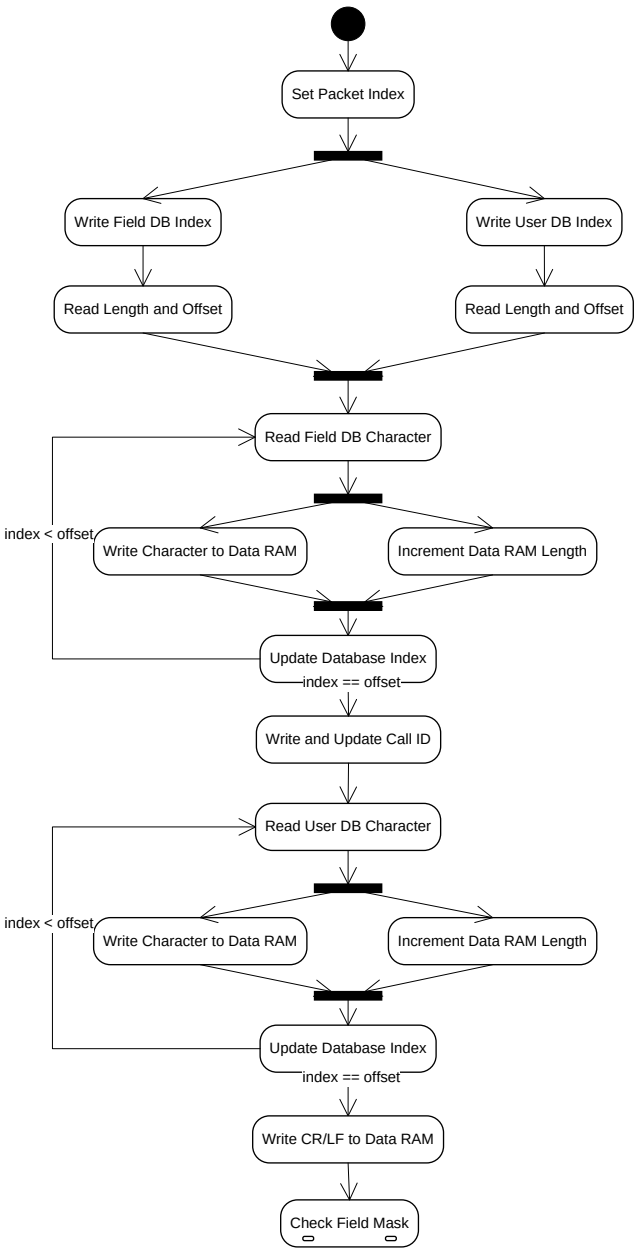

Figure 7.12: The Algorithm Used to Generate the 'Call-ID' line in the Hardware Component of the RMCS.

is otherwise retrieved from the ingress packet, written and saved using the same process. After a space is written to the SIP packet the ASCII data of the packet type is written to the packet using the packet type database. The algorithm terminates by writing a carriage return and line feed to the end of the line.

The algorithms for the generation of the 'Allow-Events', and 'Event' lines in a SIP mes-

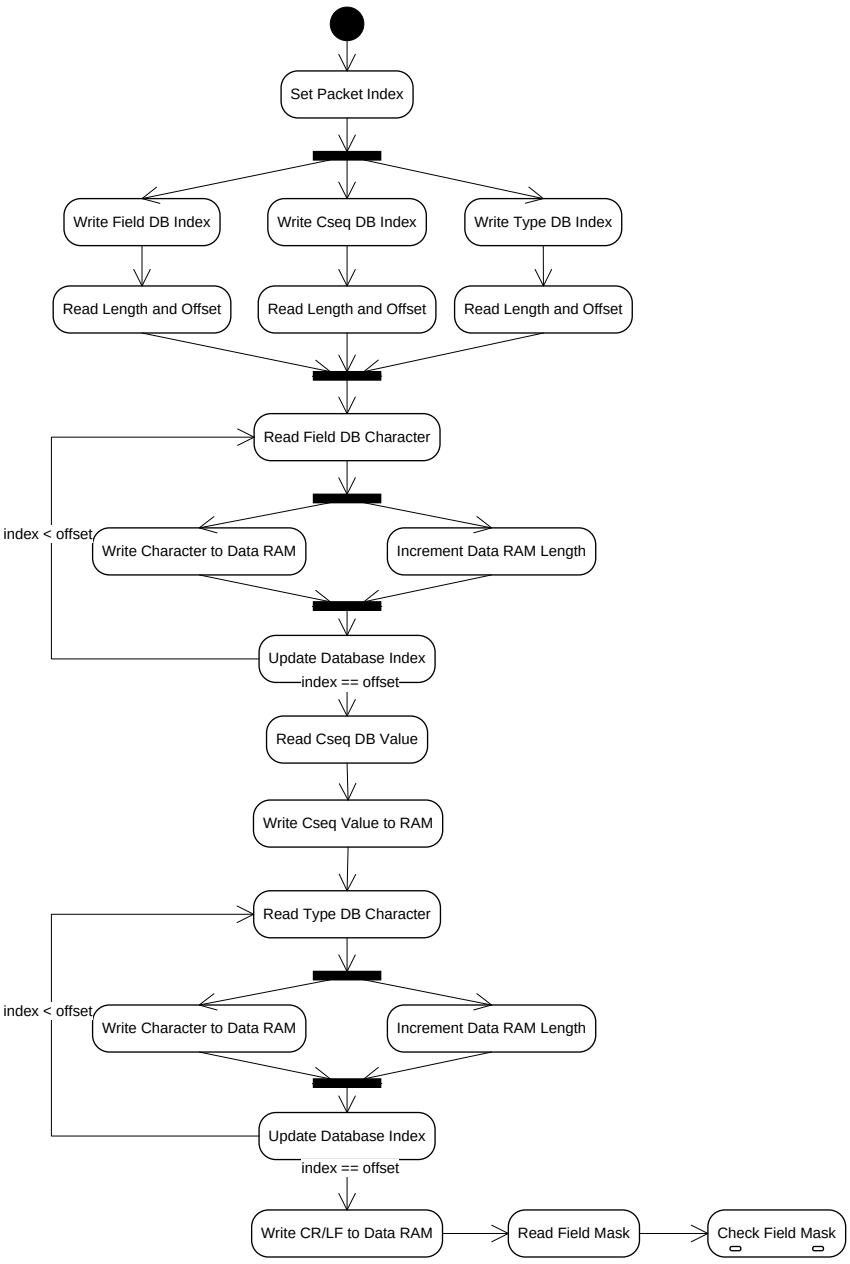

Figure 7.13: The Algorithm Used to Generate the 'Cseq' line in the Hardware Component of the RMCS.

sage are shown in Fig. 7.14. The 'Event' line is used to denote a sequence of events that may occur during the transmission of SIP data and could be supported by a client. The 'Allow-Events' header is used to describe the specific events that should be supported. These lines are typically used with notifications and subscriptions and are described with a comma separated list. The algorithm supports the generation of one ASCII string for the 'Event'

and 'Allow-Event' lines that is retrieved from the event database.

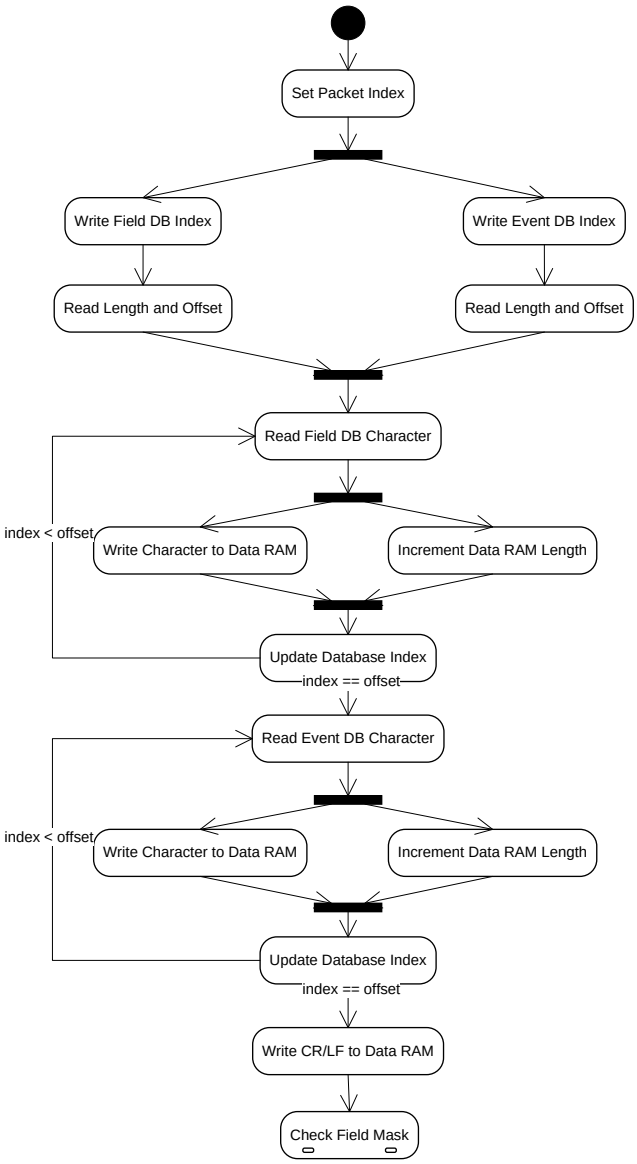

Figure 7.14: The Algorithm Used to Generate the 'Allow-Events' and 'Event' lines in the Hardware Component of the RMCS.

The algorithm for the generation of the content type line is shown in 7.15. The content type line is denoted with the field 'Content-Type' and is used to describe the type of payload transmitted with a SIP packet when one is present. The ASCII string representation of all possible content types are available in a database. The appropriate ASCII string is read from the database and written to memory after the field value has been written.

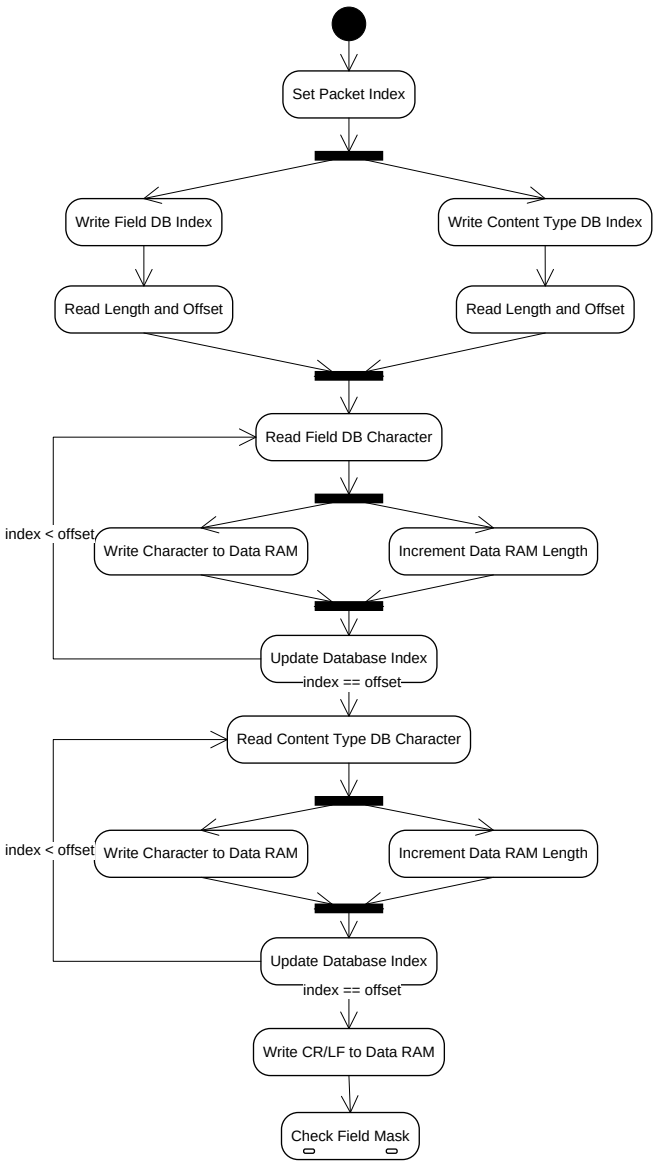

Figure 7.15: The Algorithm Used to Generate the 'Content-Type' line in the Hardware Component of the RMCS.

The algorithm for the generation of the subscription state line is shown in 7.16. The subscription state is denoted with the field 'Subscription-State' and is used to describe the current status of a subscription where applicable with values like 'active', 'pending', etc. As was the case with the content type, the ASCII string representation of all possible subscription states are available in a database. The appropriate ASCII string is read from the database and written to memory after the field value has been written.

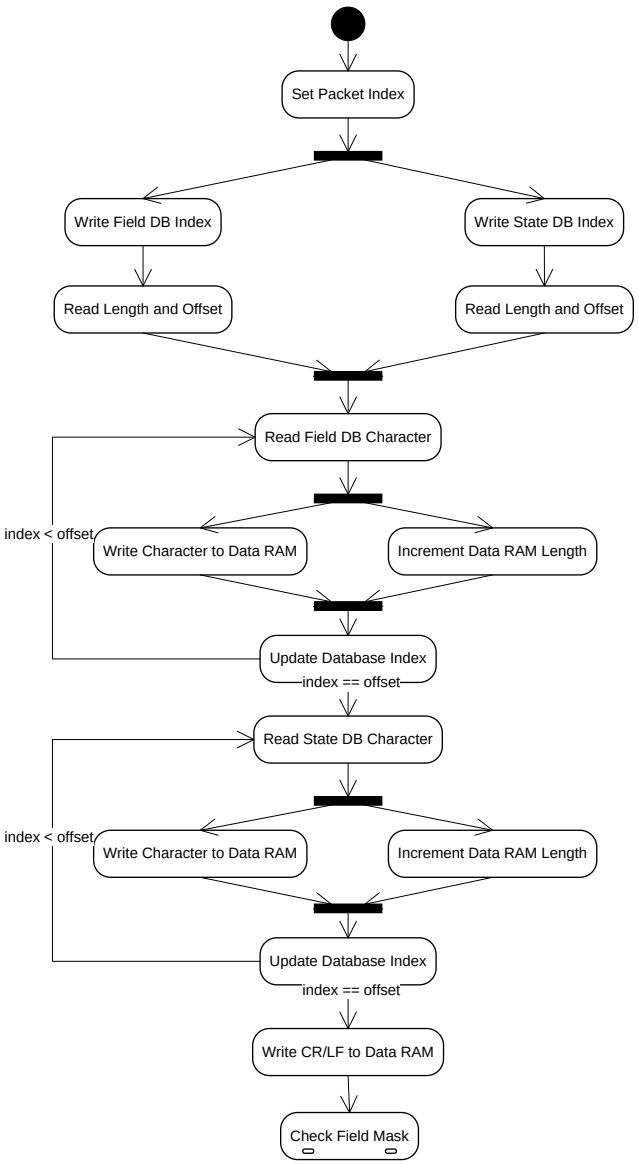

Figure 7.16: The Algorithm Used to Generate the 'Subscription-State' line in the Hardware Component of the RMCS.

The algorithm for the generation of the content length line is shown in 7.17. The content length describes the total number of bytes in the SIP packet payload when one is present. When a SIP payload is available the total number of bytes is converted into its ASCII representation and written to memory after the field value.

The algorithm for the generation of the expires line is shown in 7.17. The expires value denotes the amount of time a session is to remain active in seconds. This value is converted

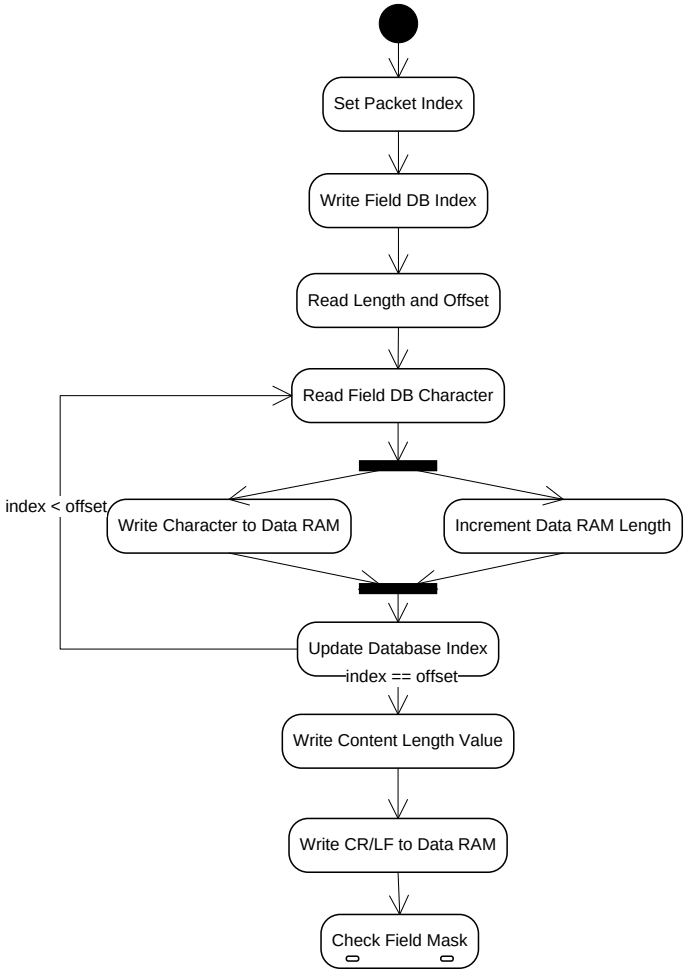

Figure 7.17: The Algorithm Used to Generate the 'Content-Length' line in the Hardware Component of the RMCS.

into its ASCII representation and written to memory after its field value when it is present.

The algorithm for the generation of the maximum forwards line is shown in 7.17. The mandatory maximum forwards line is denoted by the field 'Max-Forwards' and represents the total number of network devices that may process this SIP packet before it reach its destination and considered value. It is analogous to the time to live (TTL) field found in other communications protocols. This value is converted into its ASCII representation and written to memory after its field value.

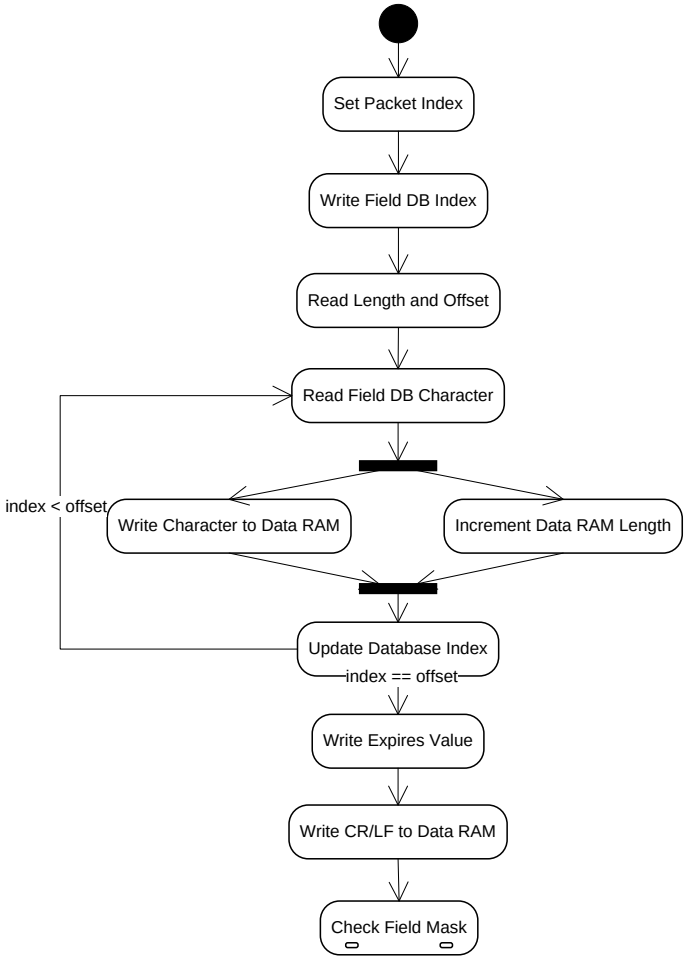

Figure 7.18: The Algorithm Used to Generate the 'Expires' line in the Hardware Component of the RMCS.

7.5 Performance

7.5.1 Resource Utilization

This section describes the resource utilization of the hardware component of RMCS as simulated on an Altera Cyclone IV device with great emphasis on the logic cells and memory required for implementation. A logic cell (also described as a logic element) is a generic term for a basic building block for Altera devices. On Cyclone devices, a logic cell consists of a four-input look up table (LUT), a programmable register and a carry chain. A carry chain is a dedicated architectural feature providing a high performance carry forward function between logic cells. The total number of resources used is described in Table 7.2.

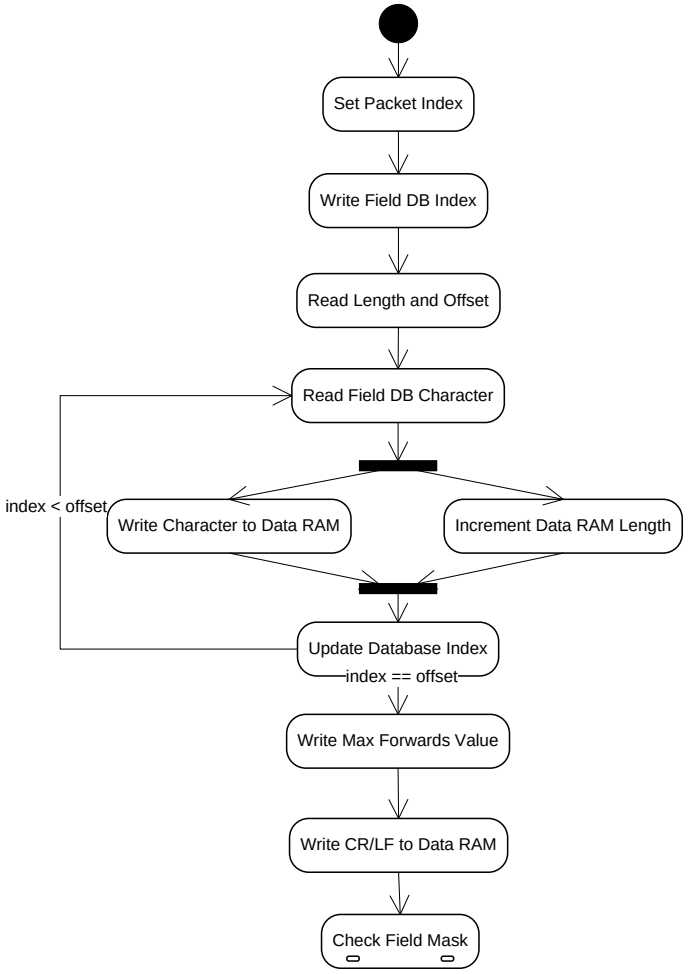

Figure 7.19: The Algorithm Used to Generate the 'Max-Forwards' line in the Hardware Component of the RMCS.

Table 7.2: RMCS Hardware Component Resource Utilization

Resource	Utilization
Logic Elements	44532
Registers	8220
Memory Bits	524352

7.5.2 Simulated Operations

The results described in this section will demonstrate the operations required to generate a SIP MESSAGE packet. The SIP MESSAGE packet used to generate the results in this section is shown in the image below:

```
MESSAGE sip:fadi@uottawa.ca SIP/2.0
To: sip:fadi@uottawa.ca
From: sip:ray@uottawa.ca;tag=1000
Via: SIP/2.0/UDP ncct.uottawa.ca:1234;branch=z9hG4bK2000
Call-ID: 3000@sip:ray@uottawa.ca
CSeq: 1 MESSAGE
Max-Forwards: 37
Content-Length: 18
Content-Type: text/plain

Hi Fadi, it's Ray!
```

Fig. 7.20 illustrates the simulated operations of the RMCS when information for a single session is cleared before data is written for packet generation. The signals at the top of the simulation (address, clk, read, reset, write, readdata, writedata and waitrequest) are defined in the Altera Avalon bus specification [2] and are required for the successful integration of the RMCS with an embedded system implemented with Altera FPGAs. The signals defined below describe information that can be read and written for a specific session. A high level command is written to the RMCS to reset all data for a specific session using the Avalon bus signals. A default value of zero is written to each register for the specific session. The waitrequest and write signals are deasserted to indicate the end of the operation. All previously defined values are overwritten for the registers in this operation. No operations were performed for this session before the reset operation so there are no defined values for any register until they are all set to zero.

Fig. 7.21 demonstrates the session data of the RMCS after all data required for SIP MESSAGE packet generation has been written through various operations. Non-zero values indicate the value that needs to be present in the SIP MESSAGE packet. Zero values indicate data that is not required to generate the SIP MESSAGE packet. No more than 32 bits of data can be written into or read from the RMCS at one time during a single operation. Therefore, writing all the data required to generate a single packet must be performed during several operations. To generate a SIP MESSAGE packet, separate commands are used to write data regarding the packet type, sender, receiver, protocol (TCP or UDP), domain and maximum number of forwards. Each of these commands requires one operation to write the necessary data where each value is mapped to a predefined value in a database except for

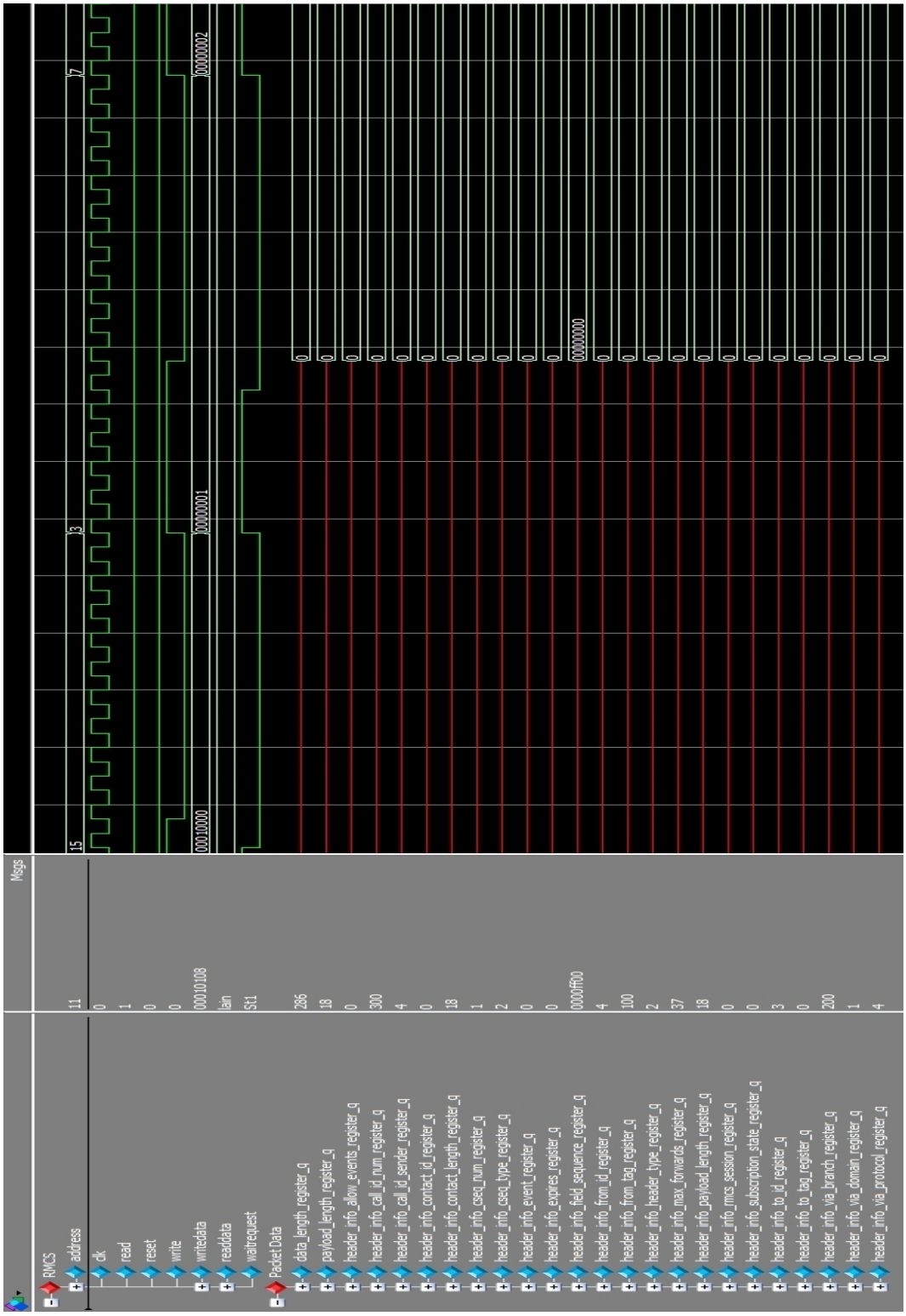

Figure 7.20: RMCS Operations Performed While Resetting Session Information.

the maximum number of forwards, which is used directly.

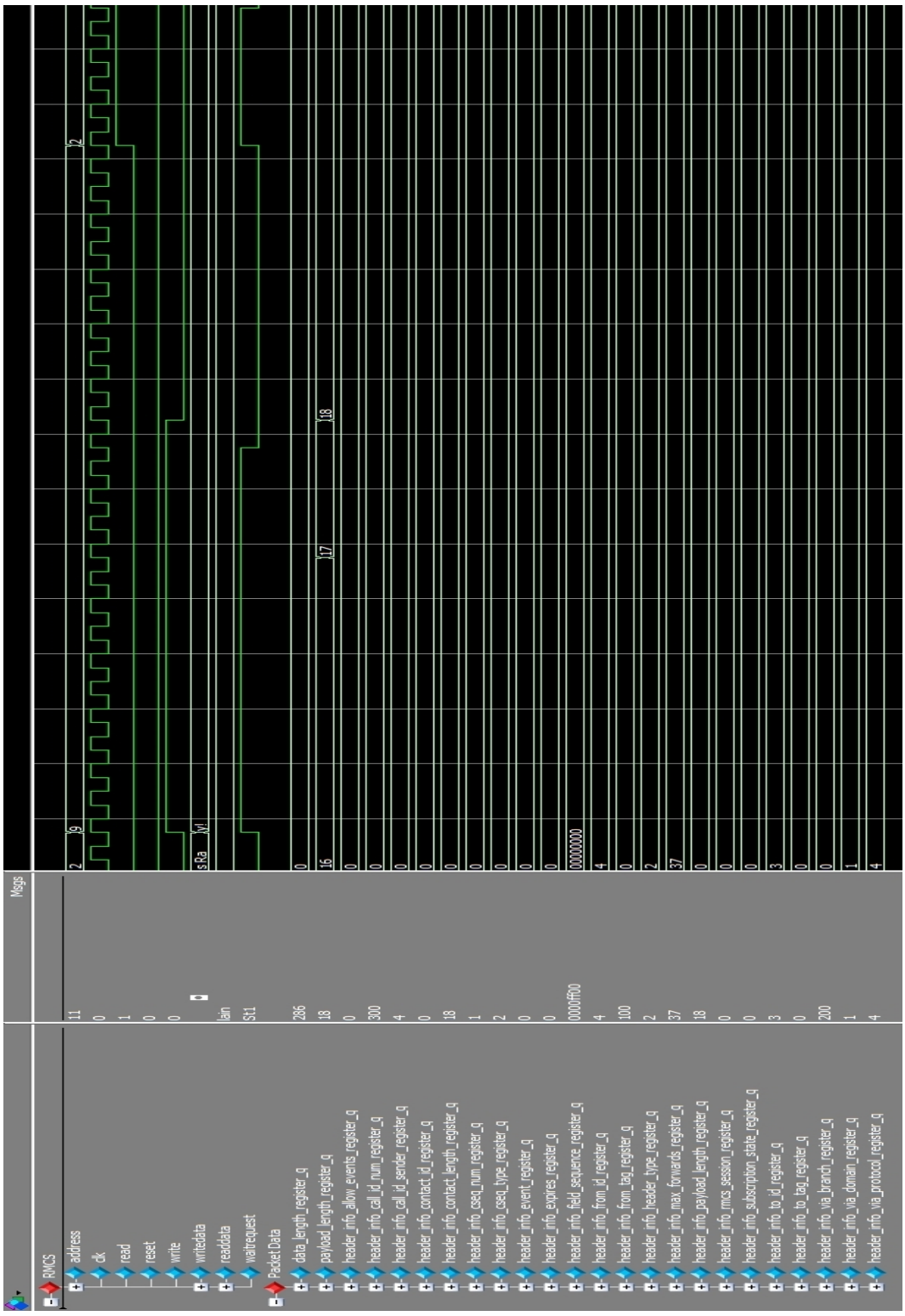

Figure 7.21: RMCS Operations Performed While Writing Data for Packet Generation.

Payload data must be written into the RMCS four characters at a time. So the number of operations required to write a SIP packet payload may be defined as ceil(x/4) where x represents the total number of characters in the payload. Undefined characters written at the end of payload must be written as null characters. An internal register is used to keep track of the total number of characters written for the payload.

Fig. 7.22 demonstrates the state of the RMCS after the SIP MESSAGE packet has been generated. Numerous databases are accessed to retrieve and write the necessary sequence of characters required for the SIP MESSAGE packet. The top most signals demonstrated for the SIP session data illustrate the last set of characters written to the SIP MESSAGE packet before the operation terminates. An internal variable is updated to keep track of the total number of characters (286) generated for the SIP MESSAGE packet. Certain values for the SIP session that were previously undefined have been given specific values by the end of the packet generation operation including data for the call identification number, a tag number for the 'From' field and a number appended to the branch component of the 'Via' field. All of this data is required by the SIP specification but is not required directly from the user so it has been generated by the packet generation algorithm of the RMCS and saved with the session. A bit map describing the sequence of fields written for this packet has also been generated and written to the session data.

Fig. 7.22 illustrates the operations required to read a generated packet from the RMCS. Since data can only be read four characters at a time it is necessary to write the data index into the RMCS with a separate instruction before the desired data can be retrieved. Four characters are automatically read from the specified index. Null characters are produced if data is read beyond the range of the SIP packet.

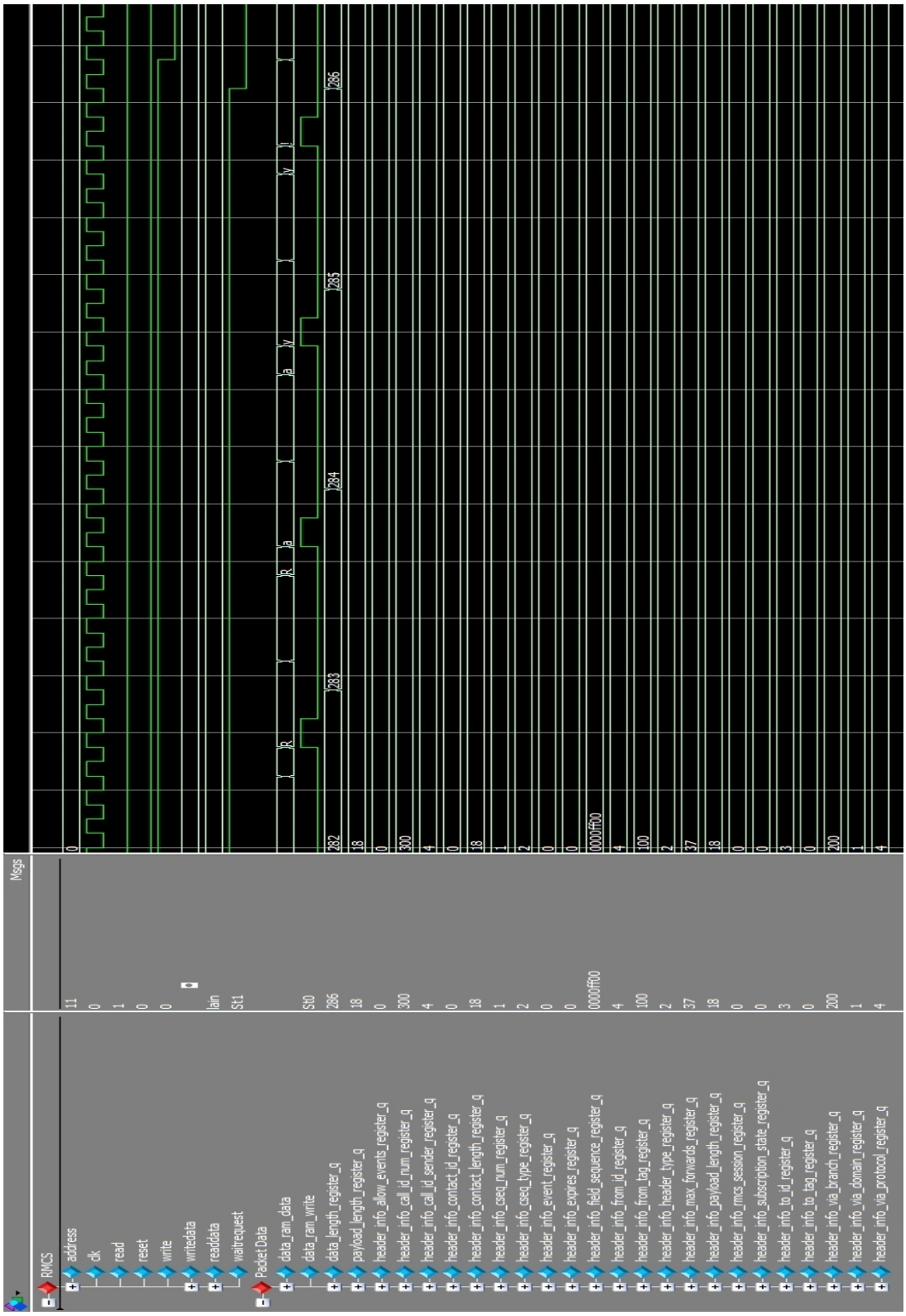

Figure 7.22: RMCS Operations Performed While Generating a SIP MESSAGE Packet.

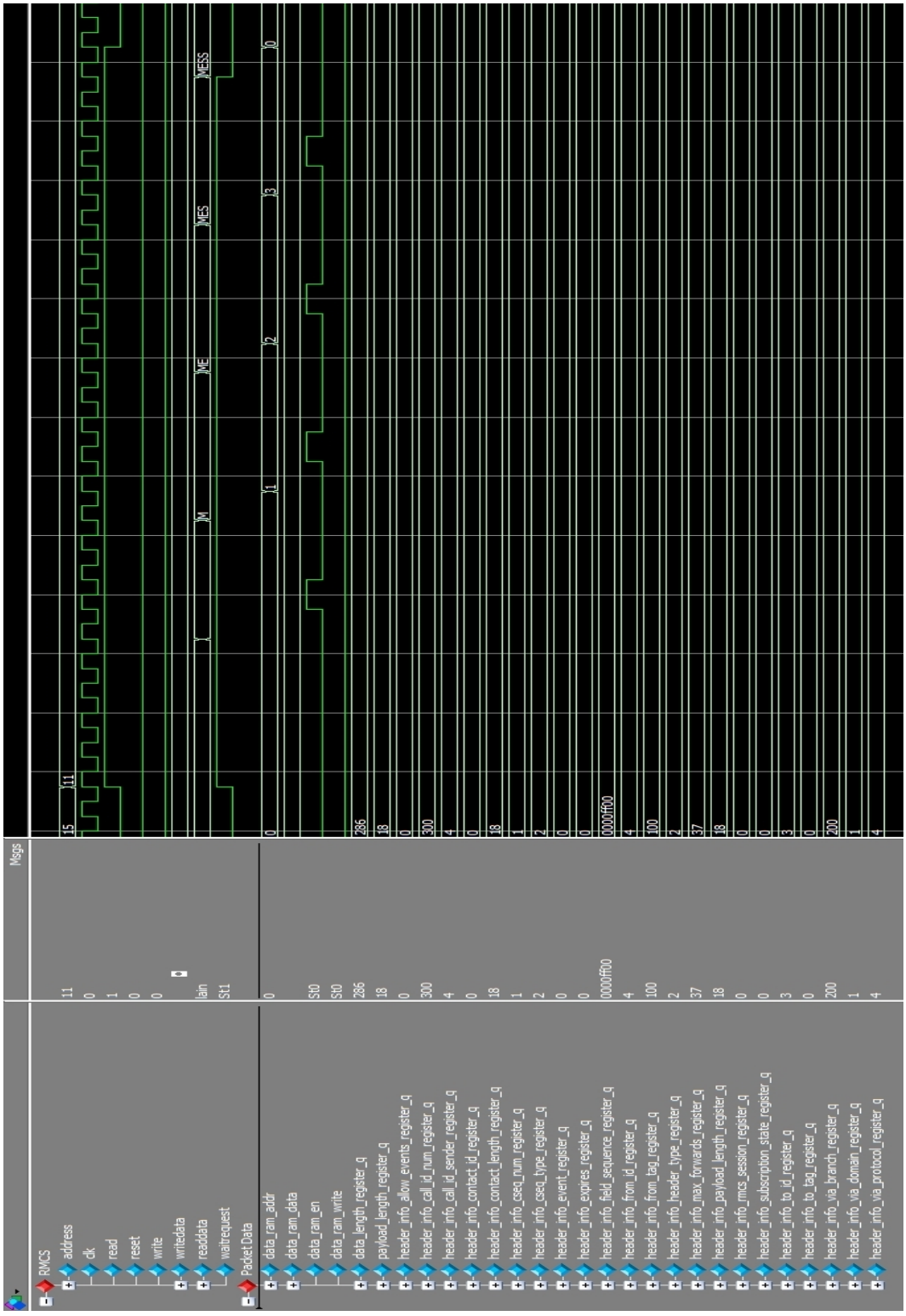

Figure 7.23: RMCS Operations Performed While Reading SIP Packet Data.

7.5.3 Performance

Table 7.3: RMCS Performance

Operation	Total Cycles Required	Cycles Required for Overhead
Asynchronous Reset	3	N/A
Synchronous Reset	5	4
Write Egress Packet Index	6	4
Session Reset	6	4
Write Other Packet Data	7	4
Read Operation Result	6	4
Write Payload Data	10,15,20,25	5
Generate MESSAGE Packet (286 characters)	2047	4
Generate NOTIFY Packet (370 characters)	2630	4
Generate SUBSCRIBE Packet (384 characters)	2724	4
Generate INVITE Packet (303 characters)	2161	4
Read Packet Length	8	4
Read SIP MESSAGE Data	10,15,20,25	5

Table 7.3 demonstrates the performance of the RMCS for the operations required to generate the packet described in the previous section. The sequence of operations is listed along with the number of cycles required to perform each operation and the overhead required. The overhead is defined as the number of cycles needed to read data into the RMCS, write data from the RMCS or otherwise indicate the end of an operation. An asynchronous reset refers to the assertion of the reset signal from the Avalon interface. This operation is required for proper initialization of the entire system and is not performed directly by the end user. This operation requires 3 cycles and does not incur any overhead as data is not read from or written to the Avalon interface. A synchronous reset operation is performed by an end user and requires 5 cycles to complete. Writing the packet index into the RMCS to identify the session of interest requires 6 cycles while all other write operations require 7 cycles. The extra cycle present in writing packet information accounts for the retrieval of the session identifier in each case where this operation is not necessary when parameter data is written to the RMCS. Writing payload data into the RMCS requires multiple cycles depending on

the number of characters that are saved. Writing one, two, three and four characters in a single operation requires ten, fifteen, twenty and twenty five cycles respectively. In each case 5 cycles are required for overhead while 5 cycles are needed to write each character to its appropriate location and update the total number of characters saved. Packet generation is the most intensive operation performed during the sequence of operations. While generating MESSAGE, NOTIFY, SUBSCRIBE and INVITE packets of various sizes, it was observed that the number of cycles required to generate an entire packet varied between 7.09 and 7.16 cycles per cycle for the entire packet. The result of an RMCS operation is read in 6 cycles while the packet length of a session requires 8 cycles to account for the proper identification of the session. The total number of cycles required to read packet data is identical to the number of cycles needed to write data. Ten, fifteen, twenty and twenty five cycles are required to read one, two, three and four characters of a SIP packet respectively.

The hardware simulations were performed with an Altera Stratix EP1S40F780C5 FPGA with a 50 MHz clock. Therefore generation of the aforementioned packets would occur in less than 55 microseconds with significant performance improvements possible with faster clocks. The same algorithms to generate SIP packets were implemented using the C programming language and a gcc compiler on a machine running Ubuntu 9.10 with a single 3.0 GHz processor. Numerous iterations of the algorithm in software revealed processing times varying between 15 microseconds and 26 microseconds with a substantial number of iterations requiring 15 microseconds or 16 microseconds. Therefore the software implementation requires at least 45,000 cycles compared to the 2724 cycles or less required to generate a SIP packet in hardware.

7.6 Conclusion

This chapter presented the RMCS used to exchange SIP data between architecture components through an IMS network. The RMCS is comprised of data processing algorithms and a hardware component to perform SIP packet generation. The data processing algorithms generate and parse SIP interface messages as required to interact with applications while all SIP data is parsed and generated through the RMCS. Numerous algorithms are implemented

in the RMCS to perform SIP packet generation. Simulated hardware results for the RMCS indicate a packet generation time of approximately 7 cycles per character for SIP packets.

Chapter 8

Results

This chapter discusses the results observed from the operation of the communications architecture. A scenario is simulated with a broker, data consumer, data producer and their corresponding RMCS to illustrate the latency observed from the communications architecture. The performance of the system is illustrated by several scenarios where successive data requests are made from a data producer with varying numbers of data sources generating several kilobytes of data in real-time. Observable trends in the latency are described where possible and conclusions are drawn on the factors that most affect the latency of the communications architecture. The times required to perform SIP tasks using an RMCS running on an FPGA are described in a separate section before the conclusion of this chapter. The sequence of messages required to retrieve data from a data producer is illustrated in an appendix at the end of the thesis.

8.1 Simulation Scenario

Fig. 8.1 illustrates the scenario that is simulated to generate the results described in this chapter. A data producer connected to an IMS network contains an arbitrary collection of data producers. Data consumers connected to the IMS network may be used to request data and transmit data in real-time to a collection of data consumers.

The implementation of this scenario is illustrated in Fig. 8.2. No DNS server has been implemented into the testbed to resolve SIP URLs with IP addresses. Therefore, each RMCS

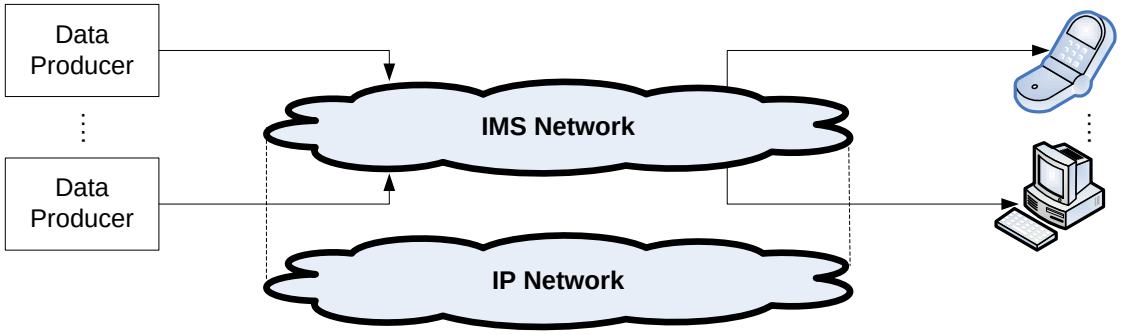

Figure 8.1: The Simulation Scenario used to Determine the Latency of the Communications Architecture.

and component transmits data to a predetermined IP address when exchanging data using UDP sockets.

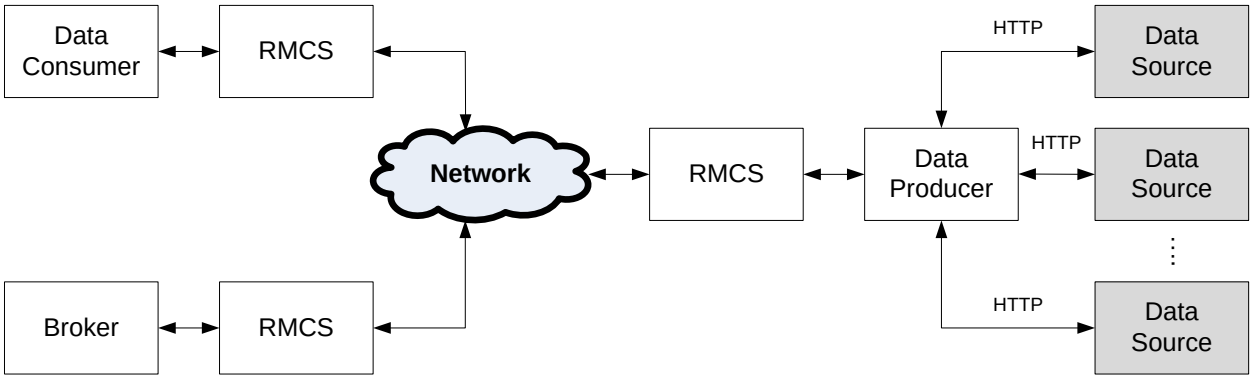

Figure 8.2: The Experimental Testbed Used to Verify the Functionality of the Communications Framework.

The simulation consists of multiple RMCS connected through a network exchanging data for a broker, data consumer and data producer. The data producer is connected to many data sources generating data according to the standard defined by the Sensor Observation Service [33]. Latency, or the time delay measured in a system, is formally defined in the simulation as the time required to receive all requested data after the request has been initiated in the data consumer. The time required at a data producer to query data from a data source through HTTP is not counted as part of the latency.

8.2 System Performance

The performance of the system is described by the latency observed through a data consumer when retrieving data from an increasing number of data sources connected to a single data producer. The objective of demonstrating the system performance in this manner is to illustrate the effect of more data being generated with successive data queries. The latency values for 100 consecutive queries are displayed for each collection of data sources. All latencies are observed running a simulated broker, data producer, data consumer and their respective RMCS on a machine running the Windows 7 Professional 32 bit operating system with 4.00 GB of RAM and an Intel Core 2 Duo 2.00 GHz CPU.

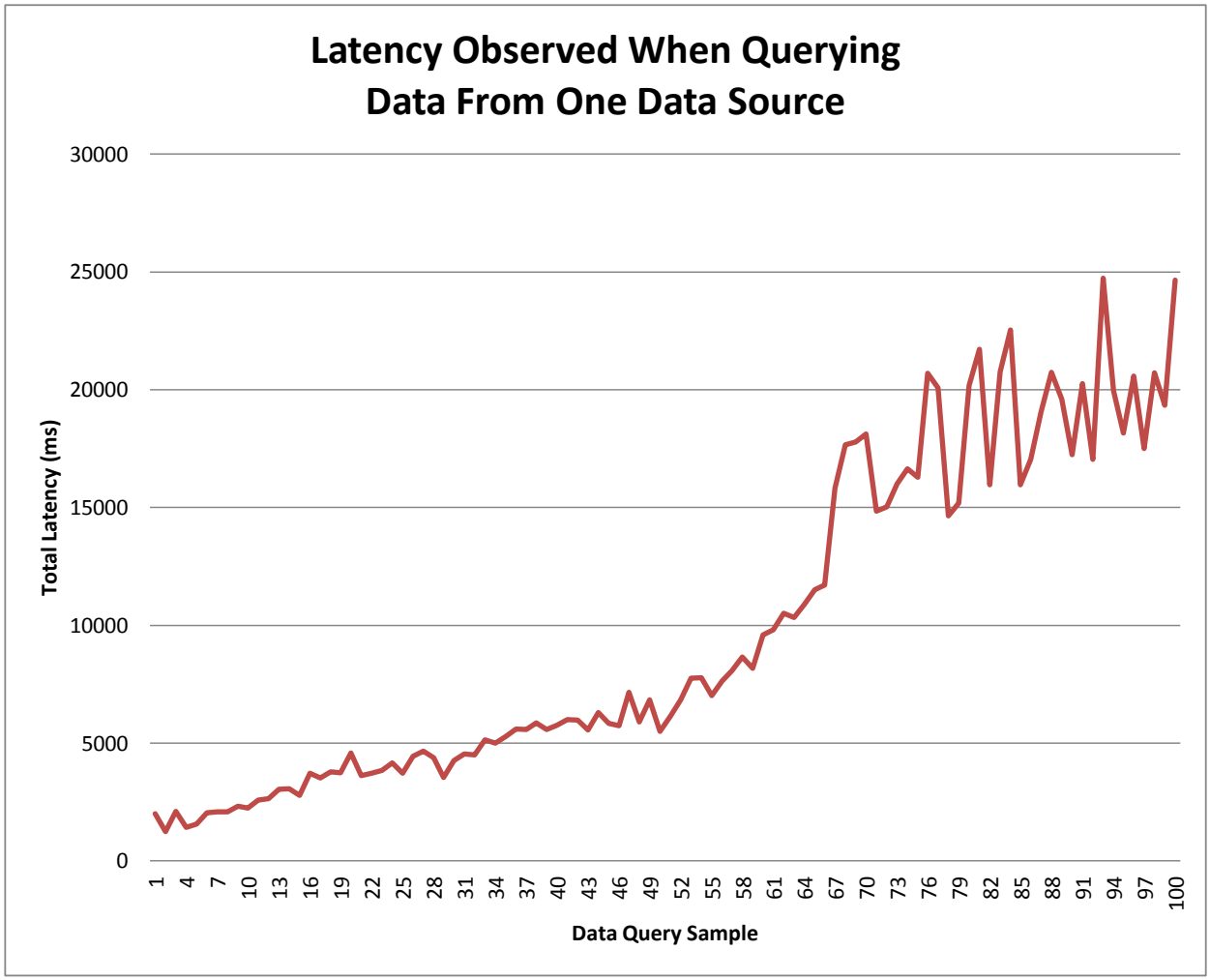

Figure 8.3: Latency Values Observed With Querying Data From One Data Source.

Fig. 8.3 illustrates the latency observed in the communications architecture when data is queried one hundred consecutive times from a data producer with one data source. Data is retrieved from different data sources with multiple queries where the total number of bytes returned ranged from 450 bytes to 24600 bytes. An approximately linear trend is observed for the latencies for the first 66 queries where the latencies increase from a minimum of 1246 milliseconds to 11707 milliseconds. The latencies for the last 33 queries are observed to vary significantly from a minimum of approximately 15000 milliseconds to a maximum of approximately 25000 milliseconds.

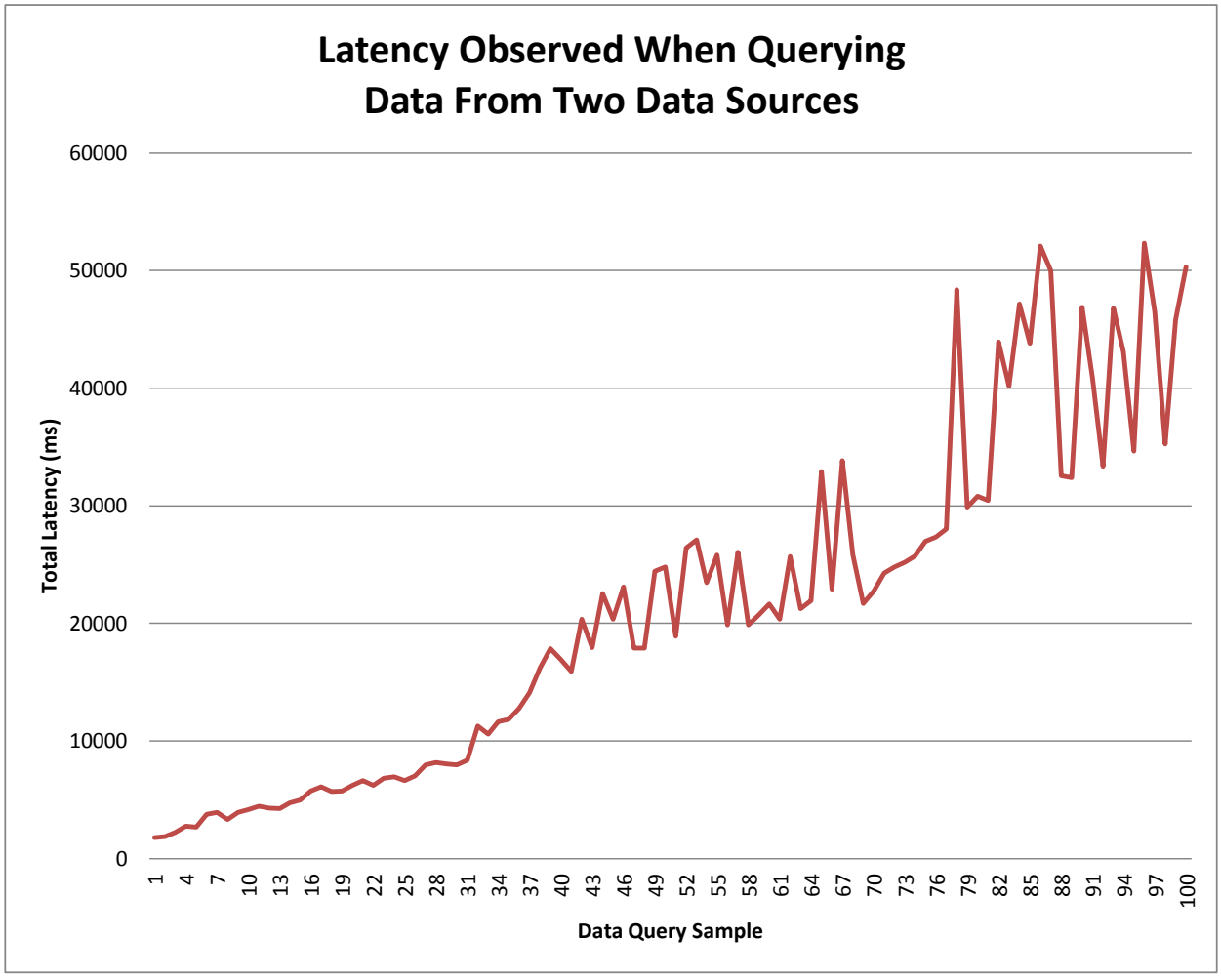

Figure 8.4: Latency Values Observed With Querying Data From Two Data Sources.

Fig. 8.4 illustrates the latency observed in the communications architecture when data is queried one hundred consecutive times from a data producer with two data sources. Data is

retrieved from different data sources with multiple queries where the total number of bytes returned ranged from 6419 bytes to 30568 bytes. The latencies observed in this scenario follow a more linear trend than the latencies observed from the previously discussed scenario. Significant variance is observed for the fortieth to seventieth queries and the last twenty queries. The minimum and maximum latency values are 1791 milliseconds and 50337 milliseconds respectively.

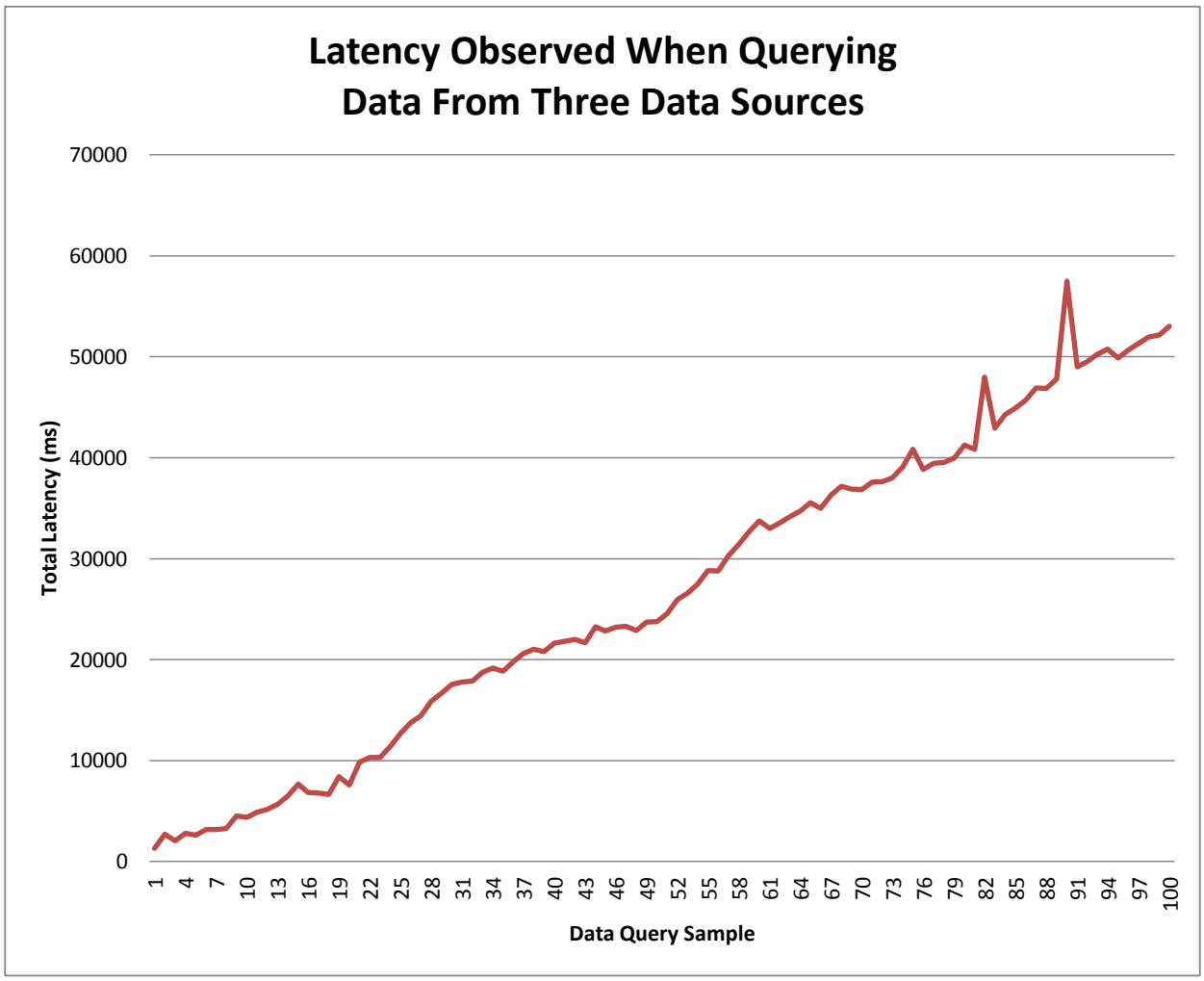

Figure 8.5: Latency Values Observed With Querying Data From Three Data Sources.

Fig. 8.5 illustrates the latency observed in the communications architecture when data is queried one hundred consecutive times from a data producer with three data sources. Data is retrieved from the three different data sources with multiple queries where the total number of bytes returned ranged from 8483 bytes to 35346 bytes. In contrast to the results

observed from the previous scenarios, it is observed that the latency increases in an extremely linear manner as multiple queries are performed. A linear approximation of the latency data reveals the following equation where the variable y represents the total latency observed in the communications architecture and x represents the number of the query that was performed:

$$y = 541.46x - 708.32, 5 \leq x \leq 100, \mathbf{x} \in \mathbf{I}$$

The smallest observed latency in this scenario was 1307 milliseconds while the largest observed latency was approximately 53.03 seconds. The linear approximation is valid from the fifth sample onward and indicates that as successive query is performed in this scenario, an additional 0.54 seconds of latency if required to retrieve data from the collection of data sources.

Fig. 8.6 illustrates the latency observed in the communications architecture when data is queried one hundred consecutive times from a data producer with four data sources. Data is retrieved from the same four data sources with each query where a total of 31423 or 31424 bytes of data are generated in 98 out of 100 queries. It is observed that the latency increases in a consistently linear manner as multiple queries are performed. A linear approximation of the latency data reveals the following equation where the variable y represents the total latency observed in the communications architecture and x represents the number of the query that was performed:

$$y = 1414.5x + 431.28, 5 \leq x \leq 100, \mathbf{x} \in \mathbf{I}$$

The linear approximation indicates that the smallest latency observed for the communications architecture in this scenario is approximately 1846 milliseconds while the largest latency observed is approximately 141.88 seconds. As successive queries are performed in this scenario, each additional query requires an additional 1.41 seconds to complete.

Fig. 8.7 illustrates the latency observed in the communications architecture when data is queried one hundred consecutive times from a data producer with five data sources. Data

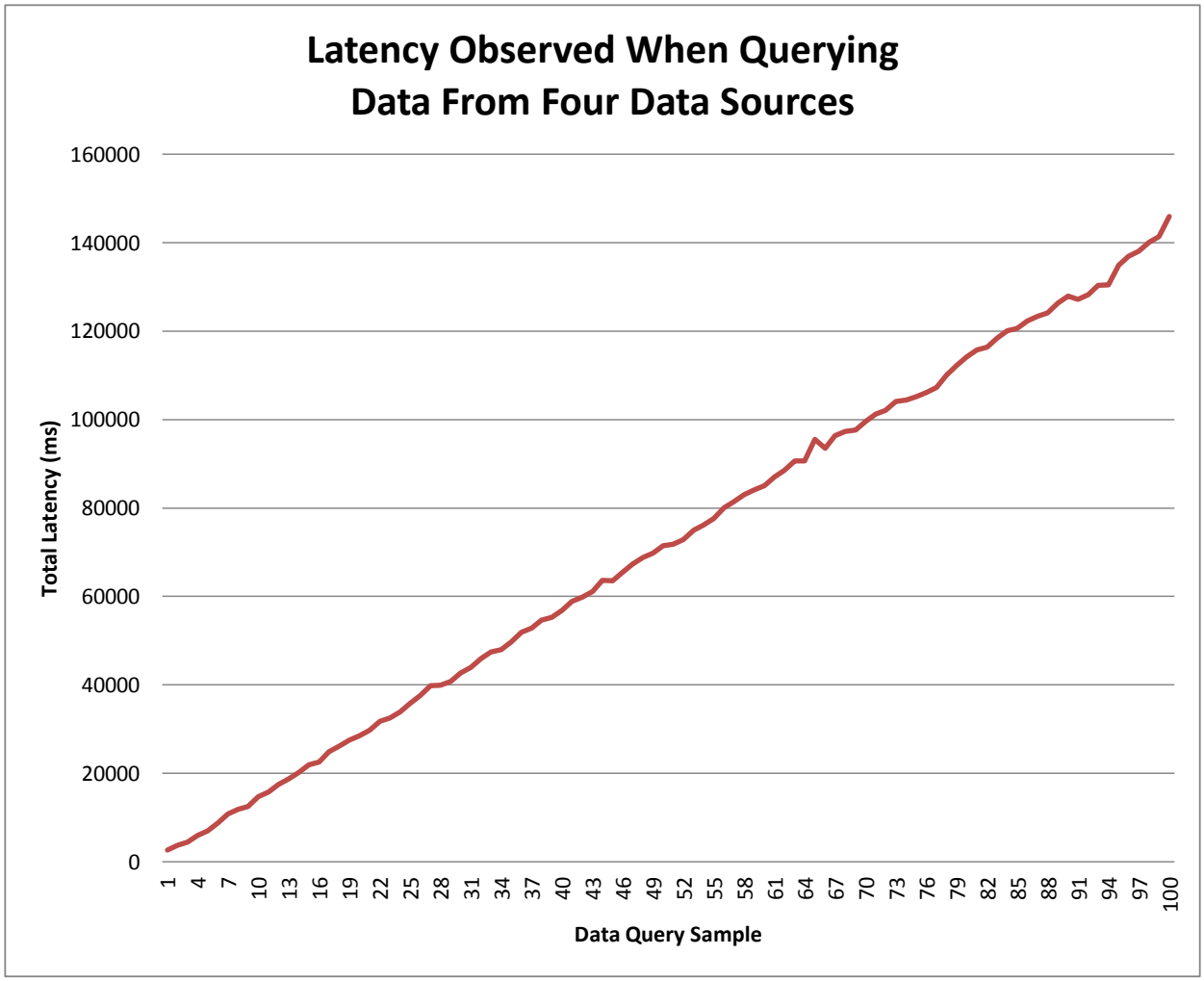

Figure 8.6: Latency Values Observed With Querying Data From Four Data Sources.

is retrieved from the same four data sources with each query where a total of 37265 bytes of data are generated in 99 out of 100 queries. It is observed that the latency increases in a consistently linear manner as multiple queries are performed. A linear approximation of the latency data reveals the following equation where the variable y represents the total latency observed in the communications architecture and x represents the number of the query that was performed:

$$y = 1649.4x + 33961, 21 \leq x \leq 100, \mathbf{x} \in \mathbf{I}$$

The first twenty latency values appeared to be relatively constant varying between 639

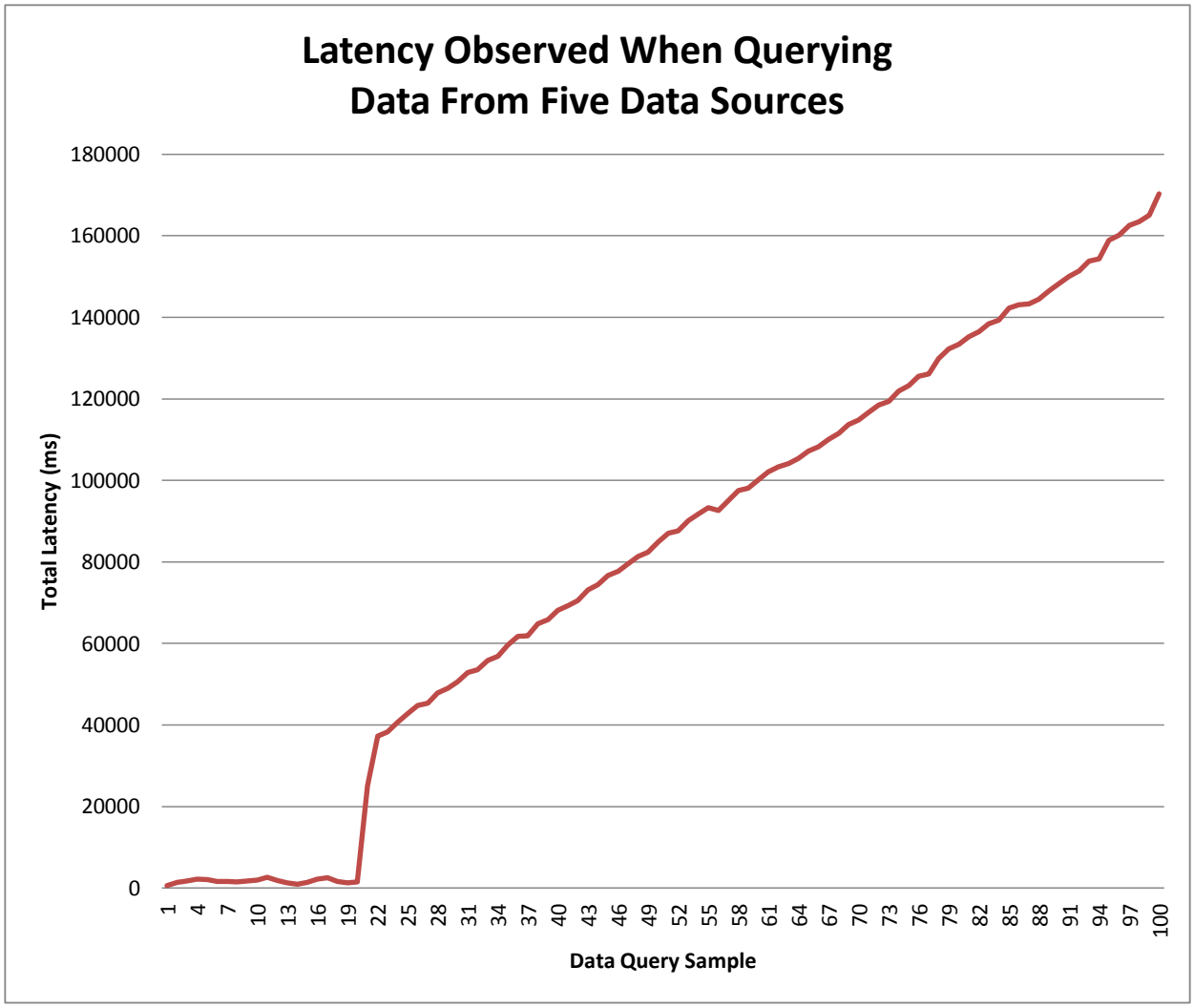

Figure 8.7: Latency Values Observed With Querying Data From Five Data Sources.

ms and 2527 ms with an average value of 1688 ms. The following eighty latency values are described with the linear approximation above. The smallest latency value among the latter eighty values is approximately 25096 milliseconds while the largest latency observed is approximately 170.3 seconds. As successive queries are performed in this scenario, each additional query requires an additional 1.65 seconds to complete for the last eighty queries.

Fig. 8.8 illustrates the latency observed in the communications architecture when data is queried one hundred consecutive times from a data producer with six data sources. Data is retrieved from the same six data sources with each query where a total of 23589 bytes of data are generated in 96 out of 100 queries. It is observed that the latency increases in a

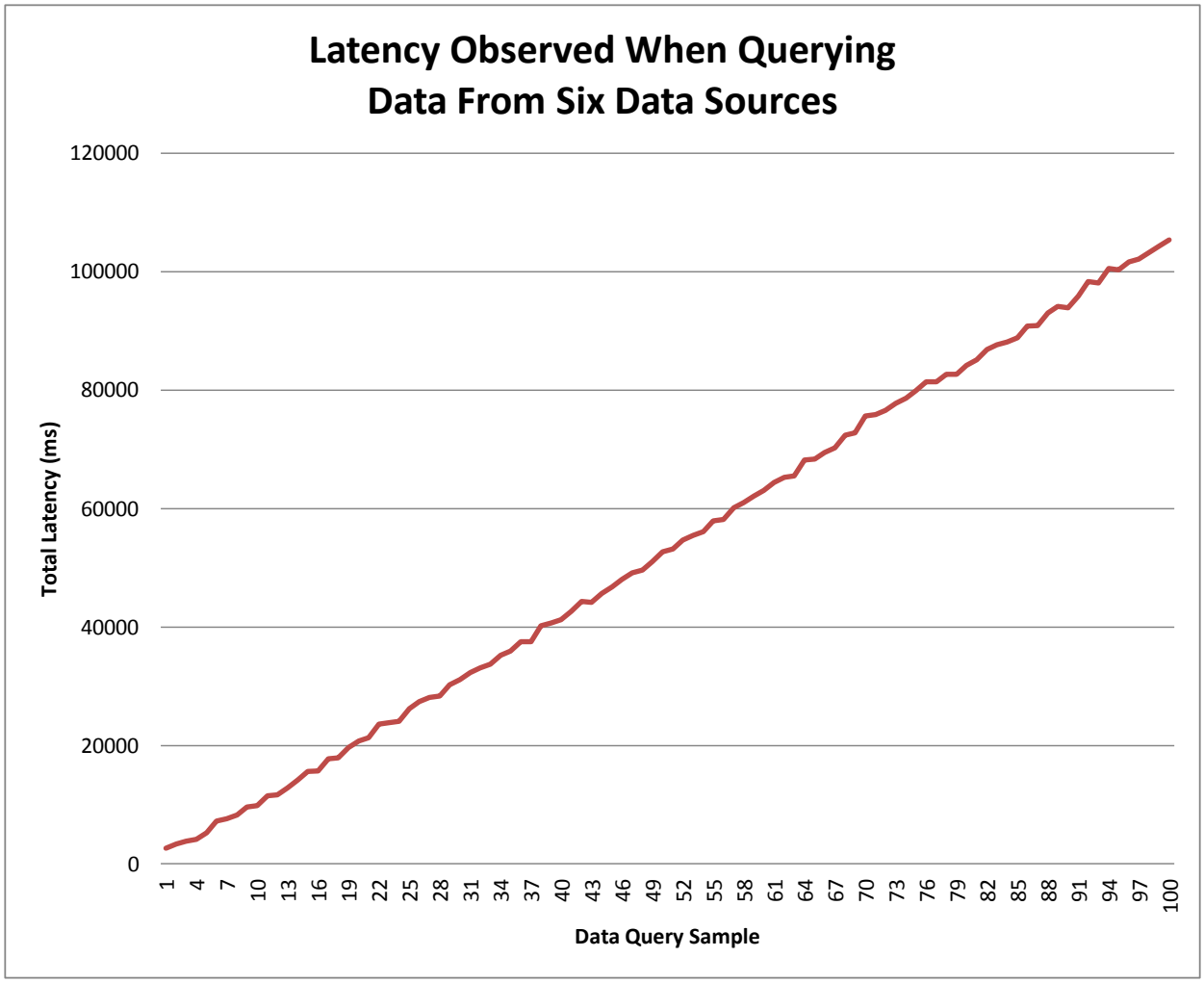

Figure 8.8: Latency Values Observed With Querying Data From Six Data Sources.

consistently linear manner as multiple queries are performed. A linear approximation of the latency data reveals the following equation where the variable y represents the total latency observed in the communications architecture and x represents the number of the query that was performed:

$$y = 1059.9x - 428.42, 1 \leq x \leq 100, \mathbf{x} \in \mathbf{I}$$

The linear approximation indicates that the smallest latency observed for the communications architecture in this scenario is approximately 631 milliseconds while the largest latency observed is approximately 106.42 seconds. As successive queries are performed in

this scenario, each additional query requires an additional 1.06 seconds to complete.

Attempts were made to determine the latency of the communications architecture with a data producer connected to seven or more data sources similar to those presented earlier in this section. The theoretical limit of data that can be transmitted by a UDP socket at one time is 65536 bytes. Failures occurred in the simulated scenarios when attempts were made to transmit more than 50000 bytes of data at one time. These failed attempts became apparent in scenarios when seven or more data sources were queried at one time. Therefore, no latency figures are presented for scenario with more than six data sources.

8.3 RMCS Processing Times for SIP Tasks

This section describes the times required to perform SIP related tasks using RMCS implemented on an Altera Stratix EP1S40F780C5 FPGA. Simulation scenarios from the previous section provide the basis for the data shown below. Tables 8.1, 8.2 and 8.3 describe the times required to perform SIP packet generation and parsing in RMCS deployed for the consumer, broker and producer respectively. The implementation of the RMCS on the FPGA is functionally identical to the simulated RMCS described in the previous section but has been optimized to account for the limited resources available in the FPGA. All times are shown in milliseconds.

Table 8.1 describes the times required to generate and parse various SIP packets at the consumer as data queries are generated and their responses are received. A SIP MESSAGE packet must be generated and transmitted to the broker through the network. This SIP MESSAGE must be successfully acknowledged with a SIP 0K packet. A successful response to the query is received through a SIP MESSAGE packet and this packet must also be acknowledged with a SIP OK packet. The time required to generate a SIP MESSAGE packet remains the same at 10 milliseconds as successively larger queries are generated while 70 milliseconds is always required to parse its SIP OK response. As the responses to these queries grows in size we observe the processing times to parse the response and generate an acknowledgment increase from 80 milliseconds to 110 milliseconds as the number of data sources increase from 3 to 6.

Table 8.1: Consumer Processing Times for SIP Packet Generation and Parsing

SIP Task	Number of Data Sources Queried					
	1	2	3	4	5	6
Generate a SIP MESSAGE packet for the broker with the data query	10 ms	10 ms	10 ms	10 ms	10 ms	10 ms
Parse a SIP OK packet from the broker	70 ms	70 ms	70 ms	70 ms	70 ms	70 ms
Parse a SIP MESSAGE packet from the broker with the data query response and generate a SIP OK packet	80 ms	80 ms	80 ms	90 ms	100 ms	110 ms

Table 8.2 describes the times required to generate and parse various SIP packets at the broker as data queries are forwarded from the consumer to the producer and responses are forwarded from the producer to the consumer. Data queries are received through SIP MESSAGE packets from the consumer and forwarded to the producer with a separate SIP message packet that is acknowledged with a SIP OK message. Similarly, data query responses are received through SIP MESSAGE packets from the producer and forwarded with another SIP MESSAGE packet to the consumer that is separately acknowledged with a SIP OK packet. It is observed that tasks related to the generation or parsing of SIP packets do not vary more than 10 milliseconds as the number of data sources increases where a query is performed or a SIP OK packet is transmitted. However, the time required to parsing a SIP MESSAGE packet with a query response and generate a SIP OK packet increases from 80 milliseconds to 110 milliseconds. The time required to generate a SIP MESSAGE with a data query response increases from 10 milliseconds to 80 milliseconds.

Table 8.3 describes the times required to generate and parse various SIP packets at the producer as data queries are received and their responses are generated. A SIP MESSAGE packet is used to receive a data query and this message must be acknowledged with a SIP OK packet. After a response for the query has been created a SIP MESSAGE packet must be generated to transmit the response and this packet must be acknowledged by the broker with a SIP OK packet. The only task requiring additional time for completion has the number of data sources increases is the SIP MESSAGE packet generation for the data query response.

Table 8.2: Broker Processing Times for SIP Packet Generation and Parsing

SIP Task	Number of Data Sources Queried					
	1	2	3	4	5	6
Parse a SIP MESSAGE packet from the consumer with the data query and generate a SIP OK packet	80 ms	70 ms	70 ms	80 ms	80 ms	70 ms
Generate a SIP MESSAGE packet for the producer with the data query	10 ms	10 ms	10 ms	10 ms	10 ms	10 ms
Parse a SIP OK packet from the producer	60 ms	70 ms	60 ms	70 ms	60 ms	70 ms
Parse a SIP MESSAGE packet from the producer with the data query response and generate a SIP OK packet	80 ms	80 ms	90 ms	100 ms	100 ms	110 ms
Generate a SIP MESSAGE packet for the consumer with the data query response	10 ms	30 ms	40 ms	50 ms	60 ms	80 ms
Parse a SIP OK packet from the consumer	70 ms	70 ms	70 ms	60 ms	70 ms	60 ms

The times needed to complete the other tasks do not vary more than 10 milliseconds as the number of data sources increases.

8.4 Conclusion

This chapter discussed the latency observed from the communications architecture defined in this thesis. A simulation was established with one broker, data consumer, data producer and three RMCS on a single PC. Multiple data sources were connected to the data producer and the latency observed when performing a query with the data consumer was observed under numerous scenarios. The latency observed by the data consumer tends to increase linearly over time as successive queries are performed for data from multiple data sources through a single data producer. The strongest linear trends were observed in scenarios where three or

Table 8.3: Producer Processing Times for SIP Packet Generation and Parsing

SIP Task	Number of Data Sources Queried					
	1	2	3	4	5	6
Parse a SIP MESSAGE packet from the broker with the data query and generate a SIP OK packet	80 ms	70 ms	70 ms	80 ms	70 ms	80 ms
Generate a SIP MESSAGE packet for the broker with the data query response	20 ms	30 ms	40 ms	60 ms	70 ms	90 ms
Parse a SIP OK packet from the broker	70 ms	60 ms	70 ms	70 ms	70 ms	70 ms

more data sources were queried for data and more than 10000 bytes of data were returned with each query. As the total amount of data was generated with each query, it was observed that subsequent queries would increase the latency observed by the data consumer by at least one second for each additional byte of data that was returned, especially in scenarios where the amount of data generated was greater than 20000 bytes. Based on the data presented in this chapter it can be concluded that the communications architecture presented in this thesis exhibits latency that increases linear with successive data queries, specifically in scenarios where the data generated is greater than 10000 bytes. These observations were made where the total data generated was either constant or varied throughout an entire scenario. It was observed that latencies followed more linear trends over time as additional data sources were queried. As SIP packet generation and parsing tasks are performed on an FPGA, additional time is required to generate and parse SIP MESSAGE packets for data query responses as additional data sources were queried. All other SIP packets were processed within approximately 10 milliseconds of the fastest observed time.

Chapter 9

Conclusion and Future Work

9.1 Conclusions

This thesis has defined a communications architecture permitting the dissemination of heterogeneous data through cellular and IP networks in real-time. Architecture components require the Reconfigurable Multimedia Collaborative System (RMCS) to exchange heterogeneous data through an IMS network. The RMCS is an embedded system defining various algorithms required to perform tasks necessary for data dissemination. Software algorithms are defined for session management and packet parsing while hardware algorithms for packet generation.

The communications architecture is based on the IP Multimedia Subsystem (IMS), a communications framework integrating users across cellular and IP networks. Data is exchanged between IMS connected users through Call Session Control Functions (CSCF) that make part of an IMS core. The Session Initiation Protocol (SIP) is used to pass data between CSCFs in an IMS core. Therefore all data exchanged between end users in the communications architecture must be mapped to SIP for it to be transmitted and received using IMS.

Synchronous and asynchronous communications are performed in the communications architecture using the instant messaging and subscription features of IMS respectively. Instant messages are exchanged in the IMS core using SIP MESSAGE packets while data is generated for a subscription using SIP NOTIFY packets. Subscription requests are made

using SIP SUBSCRIBE packets. The functionality to generate SIP packets, parse SIP packets and manage SIP sessions was implemented using a reconfigurable hardware platform to improve the performance of the communications architecture. The Reconfigurable Multimedia Collaborative System (RMCS) is an FPGA based implementation of the aforementioned SIP features that acts as an IMS interface for all end users in the communications architecture. The RMCS and many of its applications has been published as a patent and numerous publications in conferences and journals.

The SIP subscription mechanism was expanded to support the definition and transmission of heterogeneous data. Two new data types were implemented so end users could request new subscriptions: a data type for data description (permitting users to automatically receive updates about heterogeneous data types) and data values (allowing users to transmit heterogeneous data asynchronously through a SIP subscription). Defining the data description type for SIP subscriptions required defining a new type in the existing framework without any other modifications. However, defining the data value type for SIP subscriptions requires SIP SUBSCRIBE packets to carry a payload which is an unsupported feature in the SIP subscription framework and IMS. Proposed updates to the SIP subscription mechanism have been implemented into the RMCS so the communications architecture permits the creation of subscriptions for data descriptions and data values using reconfigurable hardware to improve its performance.

9.2 Suggestions for Future Research

Fig. 9.1 describes a experimental testbed for the communications architecture discussed in this thesis that could facilitate future research in this domain. The proposed experimental testbed consists of an open implementation of an IMS core where changes to the CSCFs and the Home Subscriber Server (HSS) could be performed directly to make further research contributions for IMS. The broker, data consumers and data producers would be connected directly to the P-CSCF and register with the IMS core to pass data using instant messages and subscriptions.

It is recommended that future research be performed such that the functionality for the

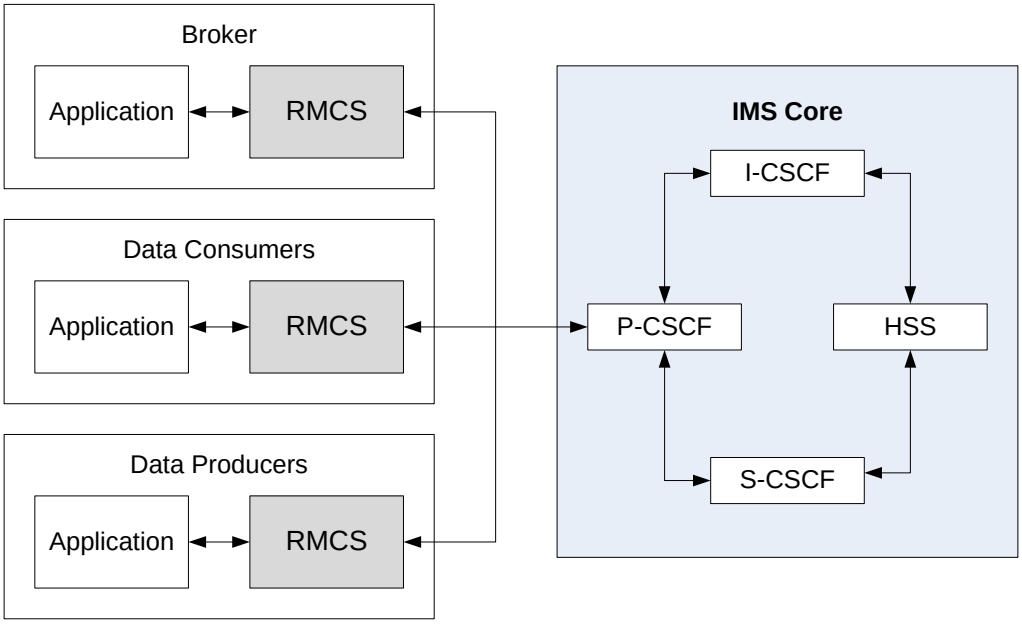

Figure 9.1: A Proposed Experimental Testbed for Future Research.

communications architecture be implemented in hardware as much as possible to facilitate novel research contributions and increase overall performance. The first step in this future research would be to turn the RMCS into a hardware component by implementing its software functionality into hardware. Algorithms for session management and SIP packet generation would need to be translated into hardware and integrated into the hardware component such that the software component of the RMCS could be eliminated except for the retrieval and transmission of external data if necessary.

Functionality for the broker, data consumer and data producer could be implemented into hardware to create a complete hardware architecture for the dissemination of heterogeneous data in real-time using IMS. This contribution would require the generation and parsing of the SIP packet payloads as XML documents in hardware as well as the definition of XML schemas to support heterogeneous data. Processing XML data in hardware would represent an extension of the research performed in [27] and [26]. Real-time XML schema definition would permit application specific contributions using the communications architecture like the one mentioned in [89].

A hardware architecture for real-time heterogeneous data dissemination that would be the

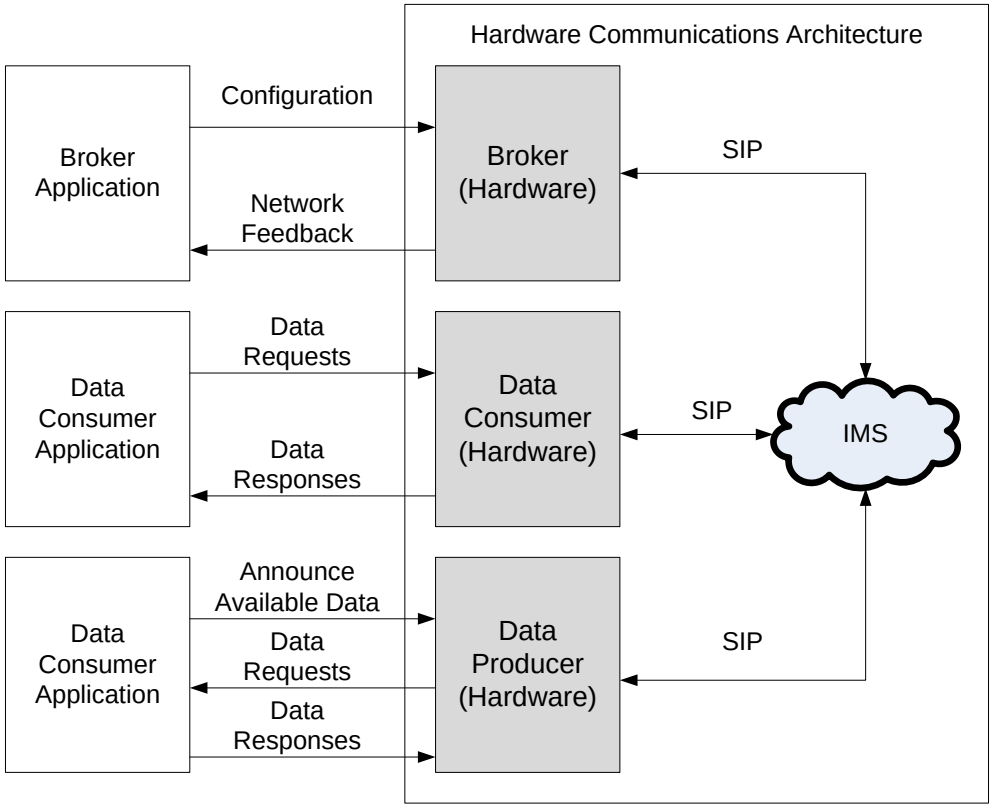

Figure 9.2: A Hardware Architecture for Real-Time Heterogeneous Data Dissemination.

result of suggested future research is depicted in Fig. 9.2. This architecture incorporates the functionality of the broker, data consumer and data producers into hardware platforms that can be integrated into embedded communications devices to define, request and transmit all data types through cellular and IP networks. The functionality of the RMCS is incorporated directly into each hardware component to send and receive data through an IMS network. Heterogeneous data would be defined and requested using XML. Applications would exist so end users interact with these components by defining available data, making requests, generating responses and providing configuration data as required.

Additional features supported by SIP should be considered for integration into the communications architecture and implemented in hardware. Numerous security features are supported by SIP and IMS that could be used directly in the communications architecture. A significant number of Internet authentication mechanisms are supported by SIP such as HTTP Digest authentication [35]. Encryption may be used to ensure message confidentiality

by using TLS [98]. TLS also provides message integrity to ensure that data has not been modified as it has been passed through an IMS core. Policies may also be implemented using SIP and IMS to ensure that the identify of the individual transmitting the message is correct.

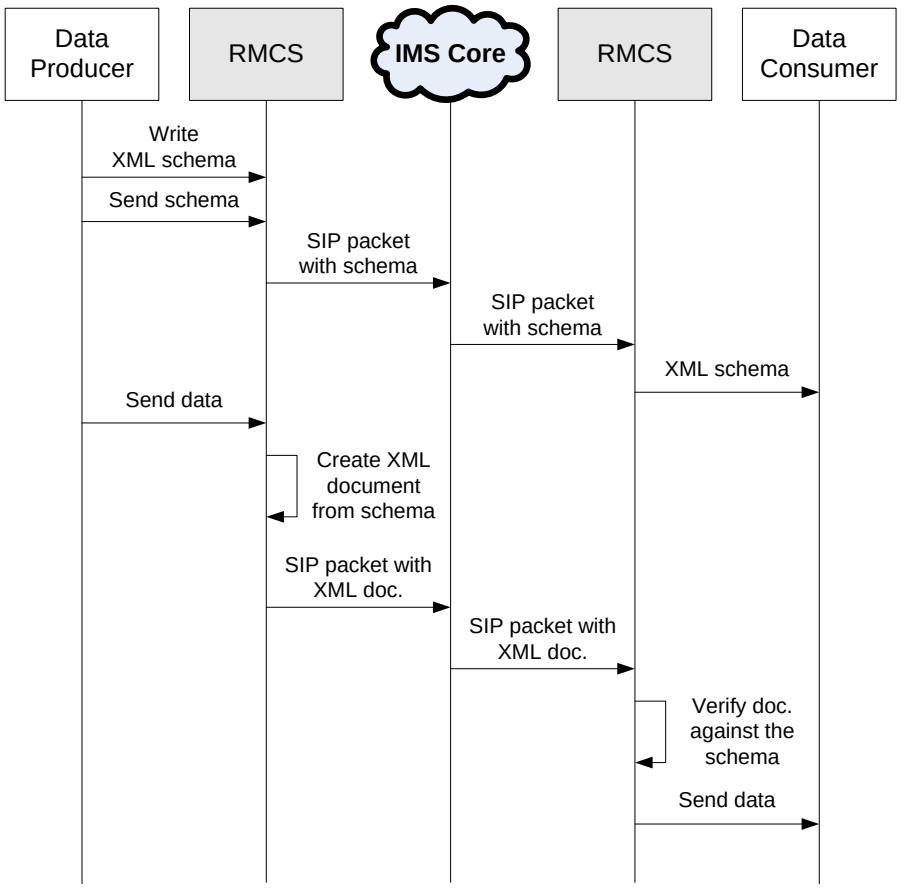

Figure 9.3: A Hardware Communications Architecture with Direct Support for XML Data and Schemas.

Heterogeneous data are often formatted as XML documents as was the case with the scenarios used to describe the performance of the communications architecture in the previous chapter. While the communications architecture permits the transmission of XML documents, no features are present to formally define or transmit XML schemas. A significant contribution to this research would be the ability to define XML schemas to describe the data that can be generated by a data producer and transmit the schema on request using the RMCS as shown in Fig. 9.3. Such a contribution would require significant XML

based functionality in hardware and would represent an extension of the research performed thus far in [27], [26]. The ability to process XML documents and determine their compliance with known schemas in hardware would remove the need to perform these tasks in the software. These contributions would not require any additional modifications to the IMS core as XML schemas and XML documents are transmitted as payloads using SIP packets. Real-time XML schema definition would permit application specific contributions using the communications architecture like those mentioned in [89].

Bibliography

[1] Zigbee Alliance. Zigbee alliance. http://www.zigbee.org, 2010.

[2] Altera Corporation. *Avalon Interface Specification*, April 2005.

[3] Y. Araki, M. Nguyen, and H. Morikawa. Design and implementation of sensor network on the ngn/ims. In *Proceedings of the IEEE International Symposium on Applications and the Internet (SAINT)*, pages 141–144, Turku, Finland, July 2008.

[4] M. El Barachi, A. Kadiwal, R. Glitho, F. Khendek, and R. Dssouli. A presence-based architecture for the integration of the sensing capabilities of wireless sensor networks in the ip multimedia subsystem. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC)*, pages 3116–3121, Las Vegas, Nevada, March 2008.

[5] M. El Barachi, A. Kadiwal, R. Glitho, F. Khendek, and R. Dssouli. The design and implementation of a gateway for ip multimedia subsystem/wireless sensor networks interworking. In *Proceedings of the IEEE Vehicular Technology Conference (VTC)*, pages 1–5, Barcelona, Spain, April 2009.

[6] P. Bellavista, E. Magistretti, U. Lee, and M. Gerla. Standard integration of sensing and opportunistic diffusion for urban monitoring in vehicular sensor networks: the mobeyes architecture. In *ISIE 2007. IEEE International Symposium on Industrial Electronics, 2007*, pages 2582 –2588, June 2007.

[7] Bluetooth. *Specification of the Bluetooth System*, 4.0 edition, December 2009.

[8] P. Bonnet, J. Gehrke, and P. Seshadri. Querying the physical world. *IEEE Personal Communications*, 7(5):10 –15, oct 2000.

[9] R. Brooks and T. Keiser. Mobile code daemons for networks of embedded systems. *IEEE Internet Computing*, 8(4):72–79, july-aug 2004.

[10] E. Burger and M. Dolly. A Session Initiation Protocol (SIP) Event Package for Key Press Stimulus (KPML). RFC 4730 (Proposed Standard), November 2006.

[11] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, and J. Arkko. Diameter Base Protocol. RFC 3588 (Proposed Standard), September 2003.

[12] G. Camarillo. The Session Initiation Protocol (SIP) Pending Additions Event Package. RFC 5362 (Proposed Standard), October 2008.

[13] B. Campbell, J. Rosenberg, H. Schulzrinne, C. Huitema, and D. Gurle. Session Initiation Protocol (SIP) Extension for Instant Messaging. RFC 3428 (Proposed Standard), December 2002.

[14] L. Caviglione and F. Davoli. Peer-to-peer middleware for bandwidth allocation in sensor networks. *IEEE Communications Letters*, 9(3):285–287, march 2005.

[15] J. Choi, D. Shin, and D.l Shin. Research and implementation of the context-aware middleware for controlling home appliances. *IEEE Transactions on Consumer Electronics*, 51(1):301–306, feb. 2005.

[16] P. K. Chrysanthis, V. Liberatore, and K. Pruhs. Middleware support for multicast-based data dissemination: A working reality. In *Proceedings of the Eighth International Workshop on Object-Oriented Real-Time Dependable Systems, 2003. (WORDS 2003).*, pages 265 – 272, January 2003.

[17] P. K. Chrysanthis, V. Liberatore, and K. Pruhs. Middleware for scalable data dissemination. In Q. Mahmoud, editor, *Middleware for Communications*. John Wiley & Sons, 2005.

[18] Y. Chu, S. Rao, and H. Zhang. A case for end system multicast (keynote address). *SIGMETRICS Perform. Eval. Rev.*, 28:1–12, June 2000.

[19] M. Cilia, L. Fiege, C. Haul, A. Zeidler, and A. P. Buchmann. Looking into the past: Enhancing mobile publish/subscribe middleware. In *Proceedings of the 2nd international workshop on Distributed event-based systems*, DEBS '03, pages 1–8, New York, NY, USA, 2003. ACM.

[20] D. Das. Exploring integration of sensor applications with ims. In *Proceedings of the IEEE International Conference on IP Multimedia Subsystem Architecture and Applications*, pages 1–3, Bangalore, India, December 2007.

[21] N. Davies, S. P. Wade, A. Friday, and G. S. Blair. Limbo: A tuple space based platform for adaptive mobile applications. In *Proceedings of the International Conference on Open Distributed Processing/Distributed Platforms (ICODP/ICDP '97)*, pages 291–302, 1997.

[22] M. Day, B. Cain, G. Tomlinson, and P. Rzewski. A Model for Content Internetworking (CDI). RFC 3466 (Informational), February 2003.

[23] S. E. Deering. Multicast routing in internetworks and extended lans. In *Symposium proceedings on Communications architectures and protocols*, SIGCOMM '88, pages 55–64, New York, NY, USA, 1988. ACM.

[24] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008.

[25] K. Edwards. *Core Jini*. Prentice Hall PTR, second edition, December 2000.

[26] F. El-Hassan and D. Ionescu. A hardware architecture of an xml/xpath broker for content-based publish/subscribe systems. In *Proceedings of the 2010 International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, pages 138 –143, December 2010.

[27] F. El-Hassan, R. Peterkin, M. Abou-Gabal, and D. Ionescu. A high performance architecture of an xml processor for sip-based presence. In *Proceedings of ITNG09 6th International Conference on Information Technology: New Generations*, Las Vegas, USA, April 2009.

[28] A. Moizard et al. The gnu osip library, 2002.

[29] L. Fang, P. Antsaklis, L. Montestruque, B. McMickell, M. Lemmon, S. Yashan, H. Fang, I. Koutroulis, M. Haenggi, M. Xie, and X. Xie. Design of a wireless assisted pedestrian dead reckoning system - the navmote experience. *IEEE Transactions on Instrumentation and Measurement*, 54(6):2342–2358, dec. 2005.

[30] T. Finin, R. Fritzson, D. McKay, and R. McEntire. Kqml as an agent communication language. In *Proceedings of the third international conference on Information and knowledge management*, CIKM '94, pages 456–463, New York, NY, USA, 1994. ACM.

[31] J. Flinn, D. Narayanan, and M. Satyanarayanan. Self-tuned remote execution for pervasive computing. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems*, pages 61–, Washington, DC, USA, 2001. IEEE Computer Society.

[32] C. Fok, G. Roman, and C. Lu. Agilla: A mobile agent middleware for self-adaptive wireless sensor networks. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 4:16:1–16:26, July 2009.

[33] 52 North Initiative for Geospatial Open Source Software GmbH. 52 north. http://52north.org/, 2010.

[34] WiMax Forum. Wimax forum. http://www.wimaxforum.org, 2010.

[35] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. HTTP Authentication: Basic and Digest Access Authentication. RFC 2617 (Draft Standard), June 1999.

[36] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1994.

[37] S. Ganeriwal, L. Balzano, and M. Srivastava. Reputation-based framework for high integrity sensor networks. *ACM Transactions on Sensor Networks*, 4:1–37, June 2008.

[38] L. Garcés-Erice, D. Bauer, and P. Scotton. A flexible and scalable message broker for sensor network integration. In *Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE*, COMSWARE '09, pages 4:1–4:9, New York, NY, USA, 2009. ACM.

[39] D. Garlan, D. P. Siewiorek, A. Smailagic, and P. Steenkiste. Project aura: toward distraction-free pervasive computing. *IEEE Pervasive Computing*, 1(2):22–31, apr-jun 2002.

[40] G. Gehlen, F. Aijaz, M. Sajjad, and B. Walke. A mobile context dissemination middleware. In *Fourth International Conference on Information Technology, 2007. ITNG '07.*, pages 155 –160, April 2007.

[41] G. Gehlen, F. Aijaz, M. Sajjad, and B. Walke. A rule based publish/subscribe context dissemination middleware. In *Wireless Communications and Networking Conference, 2007.WCNC 2007. IEEE*, pages 2541–2546, March 2007.

[42] H. Gjermundrod, D. E. Bakken, C. H. Hauser, and A. Bose. Gridstat: A flexible qos-managed data dissemination framework for the power grid. *IEEE Transactions on Power Delivery*, 24(1):136–43, January 2009.

[43] G. Hackmann, C. Julien, J. Payton, and G.C. Roman. Supporting generalized context interactions. In *In Proceedings of the 4th International Workshop on Software Engineering for Middleware*, pages 91–106, 2004.

[44] S. Hadim and N. Mohamed. Middleware: middleware challenges and approaches for wireless sensor networks. *IEEE Distributed Systems Online*, 7(3):1–1, march 2006.

[45] Q. Han and N. Venkatasubramanian. Autosec: An integrated middleware framework for dynamic service brokering. *IEEE Distributed Systems Online*, 2(7):22–31, 2001.

[46] Q. Han and N. Venkatasubramanian. Information collection services for qos-aware mobile applications. *IEEE Transactions on Mobile Computing*, 5(5):518–535, may 2006.

[47] S. Han, R. Kumar, R. Shea, E. Kohler, and M. Srivastava. A dynamic operating system for sensor nodes. In *Proceedings of the ACM Conference on Mobile Systems, Applications and Services (MobiSys)*, Seattle, Washington, June 2005.

[48] M. Handley, S. Floyd, B. Whetten, R. Kermode, L. Vicisano, and M. Luby. The Reliable Multicast Design Space for Bulk Data Transfer. RFC 2887 (Informational), August 2000.

[49] M. Handley, V. Jacobson, and C. Perkins. SDP: Session Description Protocol. RFC 4566 (Proposed Standard), July 2006.

[50] W.B. Heinzelman, A.L. Murphy, H.S. Carvalho, and M.A. Perillo. Middleware to support sensor network applications. *Network, IEEE*, 18(1):6 – 14, jan/feb 2004.

[51] S. Helal, W. Mann, H. El-Zabadani, J. King, Y. Kaddoura, and E. Jansen. The gator tech smart house: a programmable pervasive space. *IEEE Computer*, 38(3):50–60, march 2005.

[52] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. *SIGPLAN Not.*, 35:93–104, November 2000.

[53] H. Holbrook and D. Cheriton. Ip multicast channels: Express support for large-scale single-source applications. *SIGCOMM Comput. Commun. Rev.*, 29:65–78, August 1999.

[54] O. Holder, I. Ben-Shaul, and H. Gazit. System support for dynamic layout of distributed applications. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*, pages 403–411, 1999.

[55] R. C. Hsu, C. Liu, W. Huang, and J. Yang. An embedded software approach for the development of sip-based voip server. In *Proceedings of the 11th Asia-Pacific Software Engineering Conference*, pages 688–694, Nov. 30-Dec. 3 2004.

[56] IBM. mqtt.org. http://mqtt.org/, August 2011.

[57] Open Geospatial Consortium Inc. Ogc - open geospatial consortium inc. http://www.gsmworld.org/, 2010.

[58] D. Ionescu, R. Peterkin, F. El-Hassan, and M. Abou-Gabal. Reconigurable multimedia collaborative system. United States Patent Application 20100057851, March 2010. `http://www.freepatentsonline.com/y2010/0057851.html`.

[59] C. Jaikaeo, C. Srisathapornphat, and C.-C. Shen. Querying and tasking in sensor networks. In *Proceedings of the SPIE 14th Annual International Symposium on Aerospace/Defense Sensing, Simulation and Controls*, Orlando, Florida, April 2000.

[60] R. Jana, S. Jora, C.W. Rice, Y.-F. Chen, L. Hart, and P. Emery. Empowering the battlefield with a mobile middleware platform. In *Military Communications Conference, 2003. MILCOM 2003. IEEE*, volume 2, pages 1337–1342, October 2003.

[61] A. Johnston and O. Levin. Session Initiation Protocol (SIP) Call Control - Conferencing for User Agents. RFC 4579 (Best Current Practice), August 2006.

[62] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. Rubenstein. Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet. *SIGPLAN Not.*, 37:96–107, October 2002.

[63] C. Julien and G.-C. Roman. Egospaces: facilitating rapid development of context-aware mobile applications. *IEEE Transctions on Software Engineering*, 32(5):281–298, may 2006.

[64] S. Kaplan and J.R. Davies. Micro-controller based internet phone. In *Proceedings of the 7th AFRICON Conference in Africa*, pages 307–311, September 2004.

[65] M. Kim, F. Iseki, and K. Kim. Sensor network to collect data in field to match satellite data. In *Proceedings of the IEEE International Conference on Advanced Communication Technology (ICACT)*, pages 648–653, Gangwon-Do, South Korea, February 2008.

[66] Y. B. Kim, M. Kim, and Y. J. Lee. Cosmos: A middleware platform for sensor networks and a u-healthcare service. In *Proceedings of the 2008 ACM symposium on Applied computing*, SAC '08, pages 512–513, New York, NY, USA, 2008. ACM.

[67] Q. Le-Trung, A. Taherkordi, F. Eliassen, H. N. Pham, T. Skeie, and P.E. Engelstad. Dcm-arch: An architecture for data, control, and management in wireless sensor networks. In *Advanced Information Networking and Applications, 2009. AINA '09. International Conference on*, pages 898–905, May 2009.

[68] U. Lee, E. Magistretti, M. Gerla, P. Bellavista, and A. Corradi. Dissemination and harvesting of urban data using vehicular sensing platforms. *IEEE Transactions on Vehicular Technology*, 58(2):882 –901, February 2009.

[69] S. Li, L. Shuoqi, Y. Lin, S. Son, J. Stankovic, and Y. Wei. Event detection services using data service middleware in distributed sensor networks. *Telecommunication Systems*, 26:351–368, 2004. 10.1023/B:TELS.0000029046.79337.8f.

[70] W. Li, W. Zhang, V. Liberatore, V. Penkrot, J. Beaver, M. A. Sharaf, S. Roychowdhury, P. K. Chrysanthis, and K. Pruhs. An optimized multicast-based data dissemination middleware: A demonstration. In *Proceedings of the 19th International Conference on Data Engineering, 2003.*, pages 762–764, March 2003.

[71] T. Liu and M. Martonosi. Impala: a middleware system for managing autonomic, parallel sensor systems. *SIGPLAN Not.*, 38:107–118, June 2003.

[72] X. Liu, M. Corner, and P. Shenoy. Seva: Sensor-enhanced video annotation. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)*, 5:1–26, August 2009.

[73] Y. Lu, W. Zhang, Z. Qin, Y. Meng, and H. Yu. Deftrfid: A lightweight and distributed rfid middleware. In *2010 Sixth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, pages 181–186, December 2010.

[74] R. Mahy. A Message Summary and Message Waiting Indication Event Package for the Session Initiation Protocol (SIP). RFC 3842 (Proposed Standard), August 2004.

[75] B. Manning and R. Colella. DNS NSAP Resource Records. RFC 1706 (Informational), October 1994.

[76] J. Moons, C. De Backer, and H. Mannaert. Components for a pervasive information dissemination architecture. In *ICSNC '06. International Conference on Systems and Networks Communications, 2006.*, pages 34–39, October 2006.

[77] G. Muhl. *Large-Scale Content-Based Publish/Subscribe Systems*. PhD thesis, Darmstadt University of Technology, 2002.

[78] G. Muhl, A. Ulbrich, and K. Herrman. Disseminating information to mobile clients using publish-subscribe. *Internet Computing, IEEE*, 8(3):46–53, May-June 2004.

[79] A.L. Murphy, G.P. Picco, and G.-C. Roman. Lime: a middleware for physical and logical mobility. In *Proceedings of the 21st International Conference on Distributed Computing Systems,*, pages 524–533, apr 2001.

[80] P. Nie, J. Jin, and Y. Gong. Mires++: a reliable, energy-aware clustering algorithm for wireless sensor networks. In *Proceedings of the 13th ACM international conference on Modeling, analysis, and simulation of wireless and mobile systems*, MSWIM '10, pages 178–186, New York, NY, USA, 2010. ACM.

[81] R. Nimmelapelli. Fpga implementation of a sip message processor. Master's thesis, North Carolina State University, 2006.

[82] B. D. Noble and M. Satyanarayanan. Experience with adaptive mobile applications in odyssey. *Mobile Networks and Applications*, 4:245–254, December 1999.

[83] Inc. Object Management Group. Documents associated with data distribution services, v1.2. http://www.omg.org/spec/DDS/1.2/, January 2007.

[84] Inc. Object Management Group. Documents associated with corba, 3.1. http://www.omg.org/spec/CORBA/3.1/, January 2008.

[85] R. Persson and J. Vitella. Enhanced multi media adaptor (emma). Master's thesis, Lulea University of Technology, December 2009.

[86] R. Peterkin, M. Abou-Gabal, F. El-Hassan, and D. Ionescu. Hardware implementation of session initiation protocol servers and clients. In *Proceedings of the ISCC IEEE Symposium on Computers and Communications*, Sousse, Tunisia, July 2009.

[87] R. Peterkin, F. El-Hassan, and D. Ionescu. A reconfigurable architecture for ip multimedia subsystem session setup. In *Proceedings of the 2010 International Joint Conference on Computational Cybernetics and Technical Informatics (ICCC-CONTI)*, pages 637–642, Timisoara, Romania, May 2010.

[88] R. Peterkin and D. Ionescu. An architecture for heterogeneous data dissemination using ims. In *Proceedings of the 26th IEEE International Conference on Advance Information Networking and Applications (AINA-2012)*, Fukuoka, Japan, March 2012.

[89] R. Peterkin, D. Ionescu, and V. Groza. An architecture for mobile sensor network control using ims and reconfigurable hardware. In *Electronic Design, Test and Application (DELTA), 2011 Sixth IEEE International Symposium on*, pages 269–274, January 2011.

[90] M. Poikselka and G. Mayer. *The IMS: IP Multimedia Concepts and Services*. Wiley, third edition, 2009.

[91] K. Rikitake, Y. Araki, Y. Kawahara, M. Minami, and H. Morikawa. Ngn/ims-based ubiquitous health monitoring system. In *Proceedings of the IEEE Consumer Communications and Networking Conference (CCNC)*, pages 1–2, Las Vegas, Nevada, January 2009.

[92] L. Rising. *Design Patterns in Communications Software*. Cambridge University Press, 2001.

[93] G.-C. Roman, C. Julien, and Q. Huang. Network abstractions for context-aware mobile computing. In *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, pages 363–373, New York, NY, USA, 2002. ACM.

[94] J. Rosenberg. A Presence Event Package for the Session Initiation Protocol (SIP). RFC 3856 (Proposed Standard), August 2004.

[95] J. Rosenberg. A Session Initiation Protocol (SIP) Event Package for Registrations. RFC 3680 (Proposed Standard), March 2004.

[96] J. Rosenberg. A Watcher Information Event Template-Package for the Session Initiation Protocol (SIP). RFC 3857 (Proposed Standard), August 2004.

[97] J. Rosenberg. The Extensible Markup Language (XML) Configuration Access Protocol (XCAP). RFC 4825 (Proposed Standard), May 2007.

[98] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), June 2002. Updated by RFCs 3265, 3853, 4320, 4916.

[99] J. Rosenberg, H. Schulzrinne, and R. Mahy. An INVITE-Initiated Dialog Event Package for the Session Initiation Protocol (SIP). RFC 4235 (Proposed Standard), November 2005.

[100] P. Schramm, E. Naroska, P. Resch, J. Platte, H. Linde, G. Stromberg, and T. Sturm. A service gateway for networked sensor systems. *IEEE Pervasive Computing*, 3(1):66–74, jan.-march 2004.

[101] A. Smith, H. Balakrishnan, M. Goraczko, and N. Priyantha. Tracking moving devices with the cricket location system. In *Proceedings of the 2nd international conference on Mobile systems, applications, and services*, MobiSys '04, pages 190–202, New York, NY, USA, 2004. ACM.

[102] E. Souto, G. Guimar aes, G. Vasconcelos, M. Vieira, N. Rosa, and C. Ferraz. A message-oriented middleware for sensor networks. In *Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing*, MPAC '04, pages 127–134, New York, NY, USA, 2004. ACM.

[103] R. Sparks. The Session Initiation Protocol (SIP) Refer Method. RFC 3515 (Proposed Standard), April 2003.

[104] H. Sugano, S. Fujimoto, G. Klyne, A. Bateman, W. Carr, and J. Peterson. Presence Information Data Format (PIDF). RFC 3863 (Proposed Standard), August 2004.

[105] Y. Sun and H. Ma. Auvim: A middleware for audio/video sensor networks. In *Proceedings of the international workshop on Middleware for sensor networks*, MidSens '06, pages 37–42, New York, NY, USA, 2006. ACM.

[106] Nokia Telecommunications. *Third Generation (3G) Wireless White Paper*, March 2000.

[107] Real time Outbreak and Disease Surveillance Laboratory at the University of Pittsburgh. Rods laboratory. https://www.rods.pitt.edu/site/, 2011.

[108] C. Toomey and W. Mark. Satellite image dissemination via software agents. *IEEE Expert*, 10(5):44 –51, October 1995.

[109] Trillium Digital Systems Inc. *Enhanced Data Rates for GSM Evolution EDGE*, 1999.

[110] L. Valcarenghi, L. Foschini, F. Paolucci, P. Castoldi, and F. Cugini. Topology discovery services for monitoring the global grid. *IEEE Communications Magazine*, 44(3):110–117, march 2006.

[111] W. Zhang, V. Liberatore, J. Beaver, P.K. Chrysanthis, and K. Pruhs. Scalable data dissemination using hybrid methods. In *IEEE International Symposium on Parallel and Distributed Processing, 2008. IPDPS 2008.*, pages 1–12, April 2008.

Appendix A

Communications Protocol Message Formats

The subsections below describe the requests, responses and alerts defined by the communications protocol defined in this thesis. Every request requires at least one response before the session can be terminated. Alerts are transmitted from one communicating entity to another with no response required.

A.1 Requests

A.1.1 Add Data Descriptions

This request contains the information required to add a sequence of data descriptions to the broker. This message is used in a communications session between a data producer and the broker.

Components

- Session Id: A session identifier data structure used by communicating entities to exchange data.
- Data Descriptions: A list of data description data structures describing the data descriptions to be added to the broker.

A.1.2 Query Data

This request contains the information required to request data from a sequence of data sources. This message is used in communications sessions between data producers, data consumers and brokers.

Components

- Session Id: A session identifier data structure used by communicating entities to exchange data.
- Data Ids: A list of data identifier data structures listing the data descriptions to be removed from the broker.
- Time Period: An integer value describing the frequency with which this request should be automatically repeated in seconds. A value of zero indicates that the query should be performed one time.

A.1.3 Remove Data Descriptions

This request contains the information required to remove a sequence of data descriptions from the broker. This message is used in a communications session between a data producer and the broker.

Components

- Session Id: A session identifier data structure used by communicating entities to exchange data.
- Data Ids: A list of data identifier data structures listing the data descriptions to be removed from the broker.

A.2 Responses

A.2.1 Add Data Descriptions

This response identifies the data descriptions that were successfully add to the broker and rejected from the broker.

Components

- Session Id: A session identifier data structure used by communicating entities to exchange data.
- Accepted Data Ids: A list of data identifier data structures listing the data descriptions that were successfully added to the broker.
- Rejected Data Ids: A list of data identifier data structures listing the data descriptions that were not successfully added to the broker.

A.2.2 Query Data

This response contains all the requested data from a previously initiated data request while listing all the data values that could not be requested.

Components

- Session Id: A session identifier data structure used by communicating entities to exchange data.
- Data Values: A list of data value data structures listing the requested data.
- Rejected Data Ids: A list of data identifier data structures listing data descriptions of values that could not be successfully retrieved.

A.2.3 Remove Data Descriptions

This response identifies the data descriptions that were successfully removed from the broker and not removed from the broker.

Components

- Session Id: A session identifier data structure used by communicating entities to exchange data.
- Accepted Data Ids: A list of data identifier data structures listing the data descriptions that were successfully removed from the broker.
- Rejected Data Ids: A list of data identifier data structures listing the data descriptions that were not successfully removed from the broker.

A.3 Alerts

A.3.1 Data Descriptions

This alert is used to transmit data about a sequence of data descriptions. This alert is used by the broker to transmit information about data descriptions to subscribed data consumers.

Components

- Session Id: A session identifier data structure used by communicating entities to exchange data.
- Data Descriptions: A list of data description data structures listing all the available information about a sequence of data sources.

Appendix B

RMCS Commands

The list below describes all the commands that can be executed through the RMCS. Each command is described with all the necessary inputs, possible outputs and a result to indicate success or failure. The RMCS can be used to generate SIP MESSAGE, NOTIFY and SUBSCRIBE packets. Each of these commands requires the sender, receiver and a payload (if applicable) to be provided for the packet to be generated. Additional information is required for notifications and subscriptions like the subscription state and the length of the subscription. An error is produced if any of the parameters are not formatted properly, otherwise a successful result is generated.

- Generate MESSAGE packet: This command is used to generate a SIP MESSAGE packet.
  - Inputs: The 'To' field and 'From' field.
  - Outputs: An RMCS session identifier for the new packet.
  - Result: Errors for incomplete or incorrectly formatted parameters or success if no error was detected.
- Generate NOTIFY packet: This command is used to generate a SIP NOTIFY packet.
  - Inputs: The 'To' field, 'From' field, RMCS session identifier, notification state and payload.

  - Outputs: An RMCS session identifier for the new packet.
  - Result: Errors for incomplete or incorrectly formatted parameters or success if no error was detected.

- Generate SUBSCRIBE packet: This command is used to generate a SIP SUBSCRIBE packet.
  - Inputs: The 'To' field, 'From' field, RMCS session identifier, subscription event, expires value and payload.
  - Outputs: An RMCS session identifier for the new packet.
  - Result: Errors for incomplete or incorrectly formatted parameters or success if no error was detected.

- Parse SIP packet: This command is used to parse a packet formatted in SIP and generate any appropriate responses.
  - Inputs: A buffer containing the SIP formatted packet.
  - Outputs: An RMCS session identifier for the new packet.
  - Result: n error if the SIP packet was not formatted properly or success if no error was detected.

- Read egress header data: This command is used to read data about a SIP packet header that was generated by the RMCS.
  - Inputs: An RMCS session identifier and an index indicating the type of data to be read.
  - Outputs: The desired component of the SIP packet header.
  - Result: An error indicating the session identifier or the data index is invalid or success if no error was detected.

- Read ingress header data: This command is used to read data about a SIP packet header that was received by the RMCS.

  - Inputs: An RMCS session identifier and an index indicating the type of data to be read.
  - Outputs: The desired component of the SIP packet header.
  - Result: An error indicating the session identifier or the data index is invalid or success if no error was detected.

- Read ingress payload data: This command is used to read the payload of a SIP packet that was received by the RMCS.
  - Inputs: An RMCS session identifier.
  - Outputs: A buffer of data containing the requested payload.
  - Result: An empty buffer is returned if no payload was generated. An error indicating the session identifier or success if no error was detected.

- Read subscription session identifer: This command is used to obtain the subscription session identifier associated with a notification session.
  - Inputs: The RMCS notification session identifier.
  - Outputs: The subscription session identifer. An empty buffer is returned if no payload was generated.
  - Result: An error indicating the session identifier or success if no error was detected.

- Reset: This command is used to delete all data stored in the RMCS and terminate all sessions.
  - Inputs: None.
  - Outputs: None.
  - Result: None.

- Terminate session: This command is used to terminate a session in the RMCS.
  - Inputs: An RMCS session identifier.

  - Outputs: None.
  - Result: An error if the session identifer is invalid and success otherwise.

- Write configuration data: This command is used to write data into the RMCS that is used during packet generation that would not ordinarily change with each packet generation request.
  - Inputs: The transmission protocol (TCP or UDP) and the 'max-forwards' value of a SIP header.
  - Outputs: None.
  - Result: An error if any input parameter values are invalid or success if no error was detected.

The parsing of a SIP packet may be performed with the execution of a single command. The entire SIP packet must be written to the buffer of the RMCS, including the terminating carriage return and line feed of a SIP header where no payload is present. An RMCS session identifier is returned when the SIP packet has been successfully parsed. An error is generated if the SIP packet is syntactically incorrect.

When the RMCS is reset, information regarding all sessions is automatically removed. Configuration information may be written regarding the maximum number of hops a packet may travel in the IMS network and the underlying transmission protocol (TCP or UDP). The subscription identifier related to a notification session can be read from the RMCS in addition to information about any egress or ingress packet of an active session. The following values may be read about a SIP packet when executing commands to read egress header data or read ingress header data:

- Allows-Events: An index indicating the event used in the 'Allows-Events' SIP header field.
- Call-ID Number: The numerical value used in the 'Call-ID' SIP header field.
- Call-ID Type: An index indicating the packet type used in the 'Call-ID' SIP header field.

- Contact: An index indicating the SIP URL used in the 'Contact' SIP header field.
- Content Length: An integer value used in the 'Content-Length' SIP header field.
- Content Type: An index indicating the content type used in the 'Content-Type' SIP header field.
- Cseq Number: The numerical value used in the 'Cseq' SIP header field.
- Cseq Type: An index indicating the packet type used in the 'Cseq' SIP header field.
- Event: An index indicating the event type used in the 'Event' SIP header field.
- Expires: The numerical value used in the 'Expires' SIP header field.
- From identifer: An index indicating the SIP URL used in the 'From' SIP header field.
- From tag: The numerical value used as the tag in the 'From' SIP header field.
- Header type: An index indicating the type of SIP packet.
- Max-Forwards: The numerical value used in the 'Max-Forwards' SIP header field.
- Subscription State: An index indicating the subscription state used in the 'Subscription-State' SIP header field.
- To identifier: An index indicating the SIP URL used in the 'To' SIP header field.
- To tag: The numerical value used as the tag in the 'To' SIP header field.
- Via branch: The numerical value appended to the branch tag of the 'Via' SIP header field.
- Via domain: An index indicating the URL used as the domain in the 'Via' SIP header field.
- Via protocol: An index indicating the protocol used in the 'Via' SIP header field.

Appendix C

Message Exchange for Data Dissemination

Fig. C.1 and the accompanying messages describe the sequence of messages exchanged when data is retrieved from a data producer by a data producer. A data request is transmitted from the data consumer to the broker through its RMCS and a SIP MESSAGE packet. The data query is forwarded to the data broker where the data necessary for a response is generated. The data from the requested data sources is sent from the data producer to the broker through an instant message and subsequently forwarded to the data consumer through another instant message.

M1
Message Type: Request
Request Type: Create Instant Message
To: sip:ray@uottawa.ca
From: sip:fadi@uottawa.ca
Creator Id: sip:fadi@uottawa.ca
Session Id: 2
Instant Message:
<Requesttype="QueryData"sessioncreator="sip:fadi@uottawa.ca"sessionid="2"timePeriod="0"><DataId><Name>temperature</Name><User>sip:moe@uottawa.ca</User></DataId></Request>

S1
MESSAGE sip:ray@uottawa.ca SIP/2.0
To: sip:ray@uottawa.ca

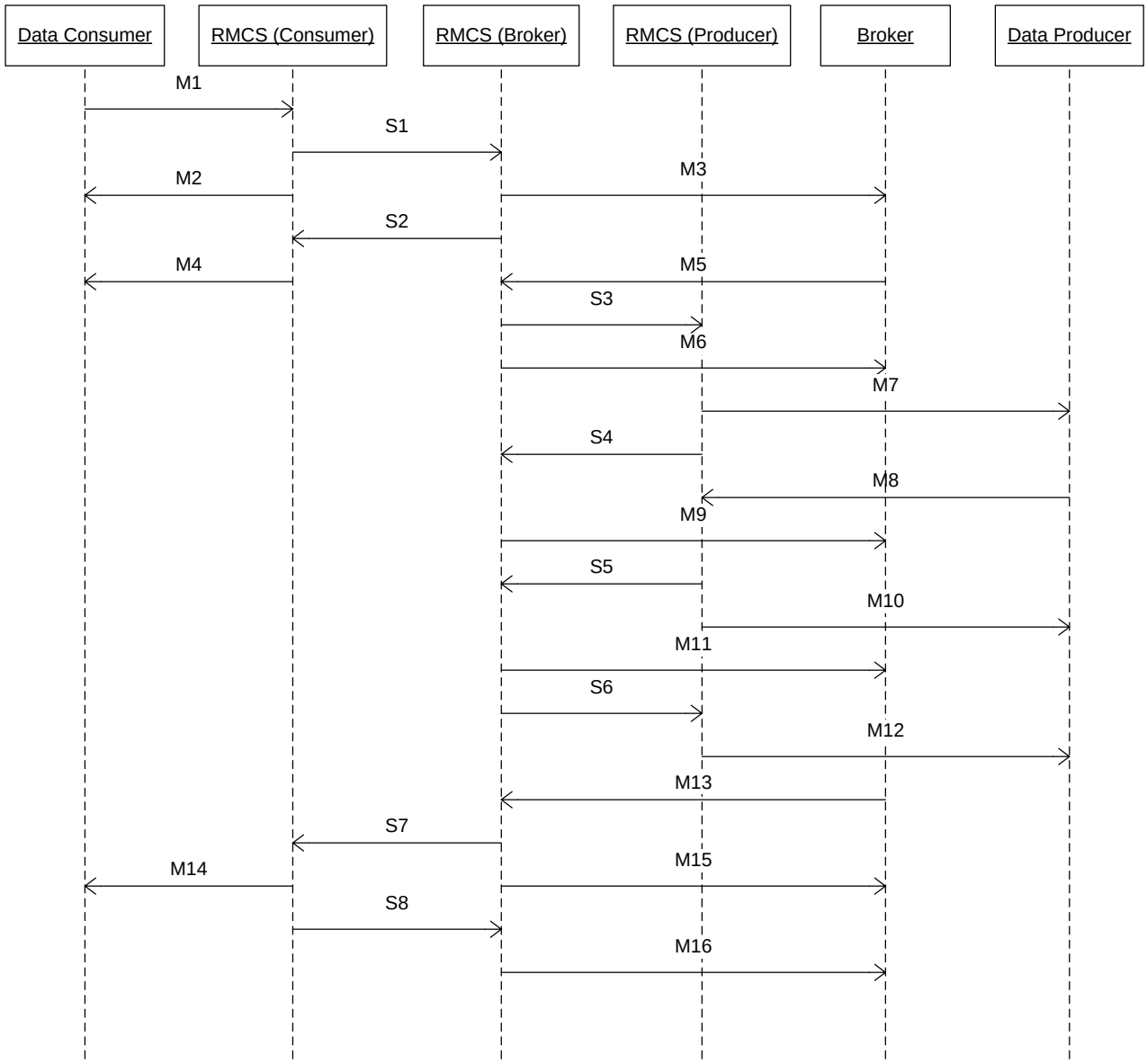

Figure C.1: The Sequence of Messages Exchanged to Query a Data Source for Data.

From: sip:fadi@uottawa.ca;tag=1001
Via: SIP/2.0/UDP ncct.uottawa.ca:1234;branch=z9hG4bK2001
Call-ID: 3001@sip:fadi@uottawa.ca
CSeq: 1 MESSAGE
Max-Forwards: 44
Content-Length: 174
Content-Type: text/plain

<Requesttype=”QueryData”sessioncreator=”sip:fadi@uottawa.ca”sessionid=”2”timePeri
od=”0”><DataId><Name>temperature</Name><User>sip:moe@uottawa.ca</User><

/DataId></Request>

M2
Message Type: Response
Response Type: Session Created
Application Session Creator: sip:fadi@uottawa.ca
Application Session Id: 2
RMCS Session Id: 2

M3
Message Type: Alert
Alert Type: New Message
To: sip:ray@uottawa.ca
From: sip:fadi@uottawa.ca
Instant Message:
<Requesttype="QueryData"sessioncreator="sip:fadi@uottawa.ca"sessionid="2"timePerio
d="0"><DataId><Name>temperature</Name><User>sip:moe@uottawa.ca</User></
DataId></Request>

S2
SIP/2.0 200 OK
To: sip:ray@uottawa.ca;tag=4002
From: sip:fadi@uottawa.ca;tag=1001
Via: SIP/2.0/UDP ncct.uottawa.ca:1234;branch=z9hG4bK2001
Call-ID: 3001@sip:fadi@uottawa.ca
CSeq: 1 MESSAGE
Content-Length: 174
Content-Type: text/plain

M4
Message Type: Alert
Alert Type: Session Terminated
RMCS Session Id: 2

M5
Message Type: Request
Request Type: Create Instant Message
To: sip:moe@uottawa.ca
From: sip:ray@uottawa.ca
Creator Id: sip:ray@uottawa.ca
Session Id: 2
Instant Message:

<Requesttype="QueryData"sessioncreator="sip:ray@uottawa.ca"sessionid="2"timePerio
d="0"><DataId><Name>temperature</Name><User>sip:moe@uottawa.ca</User></
DataId></Request>

S3
MESSAGE sip:moe@uottawa.ca SIP/2.0
To: sip:moe@uottawa.ca
From: sip:ray@uottawa.ca;tag=1002
Via: SIP/2.0/UDP ncct.uottawa.ca:1234;branch=z9hG4bK2002
Call-ID: 3001@sip:ray@uottawa.ca
CSeq: 2 MESSAGE
Max-Forwards: 44
Content-Length: 173
Content-Type: text/plain

<Requesttype="QueryData"sessioncreator="sip:ray@uottawa.ca"sessionid="2"timePeri
od="0"><DataId><Name>temperature</Name><User>sip:moe@uottawa.ca</User><
/DataId></Request>

M6
Message Type: Response
Response Type: Session Created
Application Session Creator: sip:ray@uottawa.ca
Application Session Id: 2
RMCS Session Id: 1

M7
Message Type: Alert
Alert Type: New Message
To: sip:moe@uottawa.ca
From: sip:ray@uottawa.ca
Instant Message:
<Requesttype="QueryData"sessioncreator="sip:ray@uottawa.ca"sessionid="2"timePerio
d="0"><DataId><Name>temperature</Name><User>sip:moe@uottawa.ca</User></
DataId></Request>

S4
SIP/2.0 200 OK
To: sip:moe@uottawa.ca;tag=4001
From: sip:ray@uottawa.ca;tag=1002
Via: SIP/2.0/UDP ncct.uottawa.ca:1234;branch=z9hG4bK2002
Call-ID: 3001@sip:ray@uottawa.ca
CSeq: 2 MESSAGE
Content-Length: 173

Content-Type: text/plain

M8
Message Type: Request
Request Type: Create Instant Message
To: sip:ray@uottawa.ca
From: sip:moe@uottawa.ca
Creator Id: sip:ray@uottawa.ca
Session Id: 2
Instant Message:
<Responsetype="Data"sessioncreator="sip:ray@uottawa.ca"sessionid="2"><DataValue><DataId><Name>temperature</Name><User>sip:moe@uottawa.ca</User></DataId><Value>17.2974</Value></DataValue></Response>

M9
Message Type: Alert
Alert Type: Session Terminated
RMCS Session Id: 1

S5
MESSAGE sip:ray@uottawa.ca SIP/2.0
To: sip:ray@uottawa.ca
From: sip:moe@uottawa.ca;tag=1001
Via: SIP/2.0/UDP ncct.uottawa.ca:1234;branch=z9hG4bK2001
Call-ID: 3001@sip:moe@uottawa.ca
CSeq: 2 MESSAGE
Max-Forwards: 44
Content-Length: 200
Content-Type: text/plain

<Responsetype="Data"sessioncreator="sip:ray@uottawa.ca"sessionid="2"><DataValue><DataId><Name>temperature</Name><User>sip:moe@uottawa.ca</User></DataId><Value>17.2974</Value></DataValue></Response>

M10
Message Type: Response
Response Type: Session Created
Application Session Creator: sip:ray@uottawa.ca
Application Session Id: 2
RMCS Session Id: 1

M11

Message Type: Alert
Alert Type: New Message
To: sip:ray@uottawa.ca
From: sip:moe@uottawa.ca
Instant Message:
<Responsetype="Data"sessioncreator="sip:ray@uottawa.ca"sessionid="2"><DataValue><DataId><Name>temperature</Name><User>sip:moe@uottawa.ca</User></DataId><Value>17.2974</Value></DataValue></Response>

S6
SIP/2.0 200 OK
To: sip:ray@uottawa.ca;tag=4003
From: sip:moe@uottawa.ca;tag=1001
Via: SIP/2.0/UDP ncct.uottawa.ca:1234;branch=z9hG4bK2001
Call-ID: 3001@sip:moe@uottawa.ca
CSeq: 2 MESSAGE
Content-Length: 200
Content-Type: text/plain

M12
Message Type: Alert
Alert Type: Session Terminated
RMCS Session Id: 1

M13
Message Type: Request
Request Type: Create Instant Message
To: sip:fadi@uottawa.ca
From: sip:ray@uottawa.ca
Creator Id: sip:fadi@uottawa.ca
Session Id: 2
Instant Message:
<Responsetype="Data"sessioncreator="sip:fadi@uottawa.ca"sessionid="2"><DataValue><DataId><Name>temperature</Name><User>sip:moe@uottawa.ca</User></DataId><Value>17.2974</Value></DataValue></Response>

S7
MESSAGE sip:fadi@uottawa.ca SIP/2.0
To: sip:fadi@uottawa.ca
From: sip:ray@uottawa.ca;tag=1003
Via: SIP/2.0/UDP ncct.uottawa.ca:1234;branch=z9hG4bK2003
Call-ID: 3002@sip:ray@uottawa.ca
CSeq: 3 MESSAGE

Max-Forwards: 44
Content-Length: 201
Content-Type: text/plain

<Responsetype="Data"sessioncreator="sip:fadi@uottawa.ca"sessionid="2"><DataValue><DataId><Name>temperature</Name><User>sip:moe@uottawa.ca</User></DataId><Value>17.2974</Value></DataValue></Response>

M14
Message Type: Alert
Alert Type: New Message
To: sip:fadi@uottawa.ca
From: sip:ray@uottawa.ca
Instant Message:
<Responsetype="Data"sessioncreator="sip:fadi@uottawa.ca"sessionid="2"><DataValue><DataId><Name>temperature</Name><User>sip:moe@uottawa.ca</User></DataId><Value>17.2974</Value></DataValue></Response>

M15
Message Type: Response
Response Type: Session Created
Application Session Creator: sip:fadi@uottawa.ca
Application Session Id: 2
RMCS Session Id: 1

S8
SIP/2.0 200 OK
To: sip:fadi@uottawa.ca;tag=4001
From: sip:ray@uottawa.ca;tag=1003
Via: SIP/2.0/UDP ncct.uottawa.ca:1234;branch=z9hG4bK2003
Call-ID: 3002@sip:ray@uottawa.ca
CSeq: 3 MESSAGE
Content-Length: 201
Content-Type: text/plain

M16
Message Type: Alert
Alert Type: Session Terminated
RMCS Session Id: 1

The list below describes all the commands that can be executed through the RMCS. Each command is described with all the necessary inputs, possible outputs and a result to indicate success or failure. The RMCS can be used to generate SIP MESSAGE, NOTIFY and

SUBSCRIBE packets. Each of these commands requires the sender, receiver and a payload (if applicable) to be provided for the packet to be generated. Additional information is required for notifications and subscriptions like the subscription state and the length of the subscription. An error is produced if any of the parameters are not formatted properly, otherwise a successful result is generated.

- Generate MESSAGE packet: This command is used to generate a SIP MESSAGE packet.
  - Inputs: The 'To' field and 'From' field.
  - Outputs: An RMCS session identifier for the new packet.
  - Result: Errors for incomplete or incorrectly formatted parameters or success if no error was detected.
- Generate NOTIFY packet: This command is used to generate a SIP NOTIFY packet.
  - Inputs: The 'To' field, 'From' field, RMCS session identifier, notification state and payload.
  - Outputs: An RMCS session identifier for the new packet.
  - Result: Errors for incomplete or incorrectly formatted parameters or success if no error was detected.
- Generate SUBSCRIBE packet: This command is used to generate a SIP SUBSCRIBE packet.
  - Inputs: The 'To' field, 'From' field, RMCS session identifier, subscription event, expires value and payload.
  - Outputs: An RMCS session identifier for the new packet.
  - Result: Errors for incomplete or incorrectly formatted parameters or success if no error was detected.
- Parse SIP packet: This command is used to parse a packet formatted in SIP and generate any appropriate responses.

  - Inputs: A buffer containing the SIP formatted packet.
  - Outputs: An RMCS session identifier for the new packet.
  - Result: n error if the SIP packet was not formatted properly or success if no error was detected.

- Read egress header data: This command is used to read data about a SIP packet header that was generated by the RMCS.
  - Inputs: An RMCS session identifier and an index indicating the type of data to be read.
  - Outputs: The desired component of the SIP packet header.
  - Result: An error indicating the session identifier or the data index is invalid or success if no error was detected.

- Read ingress header data: This command is used to read data about a SIP packet header that was received by the RMCS.
  - Inputs: An RMCS session identifier and an index indicating the type of data to be read.
  - Outputs: The desired component of the SIP packet header.
  - Result: An error indicating the session identifier or the data index is invalid or success if no error was detected.

- Read ingress payload data: This command is used to read the payload of a SIP packet that was received by the RMCS.
  - Inputs: An RMCS session identifier.
  - Outputs: A buffer of data containing the requested payload.
  - Result: An empty buffer is returned if no payload was generated. An error indicating the session identifier or success if no error was detected.

- Read subscription session identifer: This command is used to obtain the subscription session identifier associated with a notification session.

  - Inputs: The RMCS notification session identifier.
  - Outputs: The subscription session identifer. An empty buffer is returned if no payload was generated.
  - Result: An error indicating the session identifier or success if no error was detected.

- Reset: This command is used to delete all data stored in the RMCS and terminate all sessions.
  - Inputs: None.
  - Outputs: None.
  - Result: None.

- Terminate session: This command is used to terminate a session in the RMCS.
  - Inputs: An RMCS session identifier.
  - Outputs: None.
  - Result: An error if the session identifer is invalid and success otherwise.

- Write configuration data: This command is used to write data into the RMCS that is used during packet generation that would not ordinarily change with each packet generation request.
  - Inputs: The transmission protocol (TCP or UDP) and the 'max-forwards' value of a SIP header.
  - Outputs: None.
  - Result: An error if any input parameter values are invalid or success if no error was detected.

The parsing of a SIP packet may be performed with the execution of a single command. The entire SIP packet must be written to the buffer of the RMCS, including the terminating carriage return and line feed of a SIP header where no payload is present. An RMCS session

identifier is returned when the SIP packet has been successfully parsed. An error is generated if the SIP packet is syntactically incorrect.

When the RMCS is reset, information regarding all sessions is automatically removed. Configuration information may be written regarding the maximum number of hops a packet may travel in the IMS network and the underlying transmission protocol (TCP or UDP). The subscription identifier related to a notification session can be read from the RMCS in addition to information about any egress or ingress packet of an active session. The following values may be read about a SIP packet when executing commands to read egress header data or read ingress header data:

- Allows-Events: An index indicating the event used in the 'Allows-Events' SIP header field.
- Call-ID Number: The numerical value used in the 'Call-ID' SIP header field.
- Call-ID Type: An index indicating the packet type used in the 'Call-ID' SIP header field.
- Contact: An index indicating the SIP URL used in the 'Contact' SIP header field.
- Content Length: An integer value used in the 'Content-Length' SIP header field.
- Content Type: An index indicating the content type used in the 'Content-Type' SIP header field.
- Cseq Number: The numerical value used in the 'Cseq' SIP header field.
- Cseq Type: An index indicating the packet type used in the 'Cseq' SIP header field.
- Event: An index indicating the event type used in the 'Event' SIP header field.
- Expires: The numerical value used in the 'Expires' SIP header field.
- From identifer: An index indicating the SIP URL used in the 'From' SIP header field.
- From tag: The numerical value used as the tag in the 'From' SIP header field.

- Header type: An index indicating the type of SIP packet.
- Max-Forwards: The numerical value used in the 'Max-Forwards' SIP header field.
- Subscription State: An index indicating the subscription state used in the 'Subscription-State' SIP header field.
- To identifier: An index indicating the SIP URL used in the 'To' SIP header field.
- To tag: The numerical value used as the tag in the 'To' SIP header field.
- Via branch: The numerical value appended to the branch tag of the 'Via' SIP header field.
- Via domain: An index indicating the URL used as the domain in the 'Via' SIP header field.
- Via protocol: An index indicating the protocol used in the 'Via' SIP header field.