XML PARSER FOR SIP (Session Initiation Protocol) MESSAGE BODIES

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Masters of Engineering
in
Software Engineering

Thapar University, Patiala

By:
Pavneet Bajaj
(80631016)

Under the supervision of:
Dr. Maninder Singh

MAY 2008

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, **"**XML Parser for SIP message bodies**"**, in partial fulfillment of the requirements for the award of degree of Master of Engineering in Software Engineering submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Mr. Maninder Singh* (Astt. Professor, Thapar University, Patiala) and *Mr. Maheshwar Reddy* (Senior Project manager, Aricent Inc.) and refers other researcher's works which are duly listed in the reference section.

The matter presented in this thesis has not been submitted for the award of any other degree of this or any other university.

(Pavneet Bajaj)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

(Dr. Maninder Singh)
Astt. Professor, CSE Deptt.
Thapar University
Patiala

Countersigned by

(Dr. SEEMA BAWA)
Professor & Head
Computer Science & Engineering Department
Thapar University, Patiala

(Dr. R.K.SHARMA)
Dean Academic Affairs
Thapar University,
Patiala.

ACKNOWLEDGEMENT

First and foremost, I would like to express my sincere gratitude to my advisor Mr. Maninder Singh, Lecturer, Computer Science & Engineering Department for his immense help, guidance, stimulating suggestions and encouragement all the time. He always provide a motivating and enthusiastic atmosphere to work with, it is a great pleasure to work under his supervision.

I would like to thank the Department of Computer Science & Engineering Thapar University who gave me this opportunity. The most valuable thing I acquired from the two year studying at Thapar University is to protect, survive and endure myself in this world with increased confidence and additional faith in my abilities to achieve. I am also thankful to the authors whose works I have consulted and quoted in this work.

I also express my gratitude to Mr. Maheshwar Reddy of Aricent Inc, Bangalore, my external guide, for his invaluable, constructive, and skilled guidance, useful and timely suggestions and encouragement at every step. I highly appreciate the friendly and cooperative team with which I worked, who helped me in each and every step of my project and Aricent Inc. who gave me this opportunity and helped me, complete my work successfully.

I am very thankful to my respectable family for their fortitude. Nothing would have ever been possible without my parent's unconditional love and encouragement.

Pavneet Bajaj
(80631016)

ABSTRACT

Telephony service today is provided for the most part over circuit-switched networks, which are referred to as Public Switched Telephone Networks (PSTN). A new trend that is emerging in recent years is to provide telephony service over IP networks, known as IP Telephony or Voice over IP.

The high cost of long-distance and international voice calls is the crux of the issue in telephony world today. VOIP has cost advantage as it uses the IP networks for communication. Another very important benefit of VOIP is the integration of voice and data applications, which can result in more effective business processes. SIP (Session Initiation Protocol), an application layer signaling protocol is used for VOIP signaling needs. It is a client-server base protocol and provides the necessary protocol mechanisms so that the end user systems, proxy servers and other network entities can provide different services like call forwarding, invitations to multicast conference, personal mobility etc. Network entities implementing SIP, like SIP server, use XML parsers for their functioning. By embedding the XML parser in the SIP server code, server XML document management and few other services can be provided.

An XML parser is an essential tool for any serious programming effort involving XML documents. The parser is the link between the XML document representing data and the application code .When the XML data is received by the SIP server, the XML parser translates all the information about the user requirements to data structures. Those will be used to perform the right decision when another request is coming. Thus to use the XML meaningfully in an application, it needs to be parsed and the relevant data extracted. There are a variety of ways to achieve this like Simple API for XML (SAX) or Document Object Model (DOM).

The intent of this thesis is to give an overview of XML parser technology and its usage in the VOIP domain. It attempts to enhance the vision of parsers by explaining different types of parsers, their use and how a validating parser can be advantageous. In order to illustrate the usage of a validating parser in a SIP environment, RXP Parser which is an open source validating XML parser written in C has been used to demonstrate the parsing of XML message bodies from SIP messages.

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

CHAPTER-1
INTRODUCTION

1.1 Motivation (Purpose of Thesis)

The Extensible Markup Language (XML) is a general-purpose specification for creating custom markup languages. It is classified as an extensible language because it allows its users to define their own elements. Its primary purpose is to facilitate the sharing of structured data across different information systems, particularly via the Internet, and it is used both to encode documents and to serialize data. XML is now being widely used and has become a mature technology. It has been recommended by World Wide Web Consortium.

In the real world, computer systems and databases contain data in incompatible formats. One of the most time consuming challenges for developers has been to exchange data between such systems over the Internet. Converting the data to XML can greatly reduce this complexity and create data that can be read by different types of applications [1]. XML is the choice, when information needs to be exchanged across several (incompatible) hardware or software platforms or applications [2]. It provides a basic syntax that can be used to share information between different kinds of computers, different applications, and different organizations without needing to pass through many layers of conversion.

XML provides a simple format that is flexible enough to accommodate diverse needs. But to use the XML meaningfully in an application, it needs to be parsed and the relevant data extracted. So an XML parser is needed which is a piece of software that reads XML files and makes the information from those files available to applications and programming languages [3].

In the context of SIP, XML messages can be used for representing schemas used for automatic GUI generation and configuration of sub-events, filter for events subscribed to ,specification of alerting methods and performing remote procedure calls (SOAP) etc. Since XML ensures device neutrality, is generic in its application, allows for more information and provides for automated action, XML can be used to build a

feature rich SIP entity. XML parsers help in the interpretation of these XML message for the benefit of the SIP entity.

The main motivation behind using a validating XML parser instead of a non-validating one is that the work effort required to include support for any new parsing functionality gets significantly reduced thereby reducing the time to market. In case of a validating XML parser, validations gets done by the XML parser itself. Hence the SIP entity does not have the overhead to perform the validations, thereby reducing the effort and time required to add new features, which is a crucial business advantage.

1.1 Organization of Thesis

The rest of this thesis is organized as follows. The second chapter gives an overview of general concepts used in the thesis work introduced. This is theoretical background information about VOIP technology, Session Initiation Protocol and XML Parsers. The third chapter provides a brief literature survey i.e. the state of the art relevant to our thesis is reviewed in this chapter. Fourth chapter concentrates on problem statement after going through literature survey. The fifth chapter provides the solution to problem discussed in previous chapter. It is all about the explaining the architecture of RXP and practical demonstration of RXP including explanation of modifications made in it and the experimental results obtained. The sixth chapter is about testing RXP by giving valid and invalid inputs to it corresponding to four RFC's (RFC 3858, RFC 3994, RFC4661, RFC 4662). Finally the last chapter discusses the conclusions and future scope of the work.

CHAPTER-2
BACKGROUND INFORMATION

2.1 Voice over Internet Protocol

Voice over Internet Protocol is a protocol optimized for the transmission of voice through the Internet or other packet switched networks. VoIP is often used abstractly to refer to the actual transmission of voice (rather than the protocol implementing it).

Voice over Internet Protocol has been a subject of interest almost since the first computer network. By 1973, voice was being transmitted over the early Internet. The technology for transmitting voice conversations over the Internet has been available to end-users since at least the early 1980s. In 1996, a shrink-wrapped software product called Vocaltec Internet Phone provided VoIP along with extra features such as voice mail and caller ID. However, it did not offer a gateway to the PSTN, so it was only possible to speak to other Vocal Tec Internet Phone users. In 1997, Level 3 began development of its first soft switch. Soft switches were designed to replace traditional hardware telephone switches by serving as gateways between telephone networks [4].

VoIP has become popular largely because of the cost advantages to consumers over traditional telephone. Some cost savings are due to utilizing a single network to carry voice and data, especially where users have underused network capacity that can carry VoIP at no additional cost. VoIP to VoIP phone calls are sometimes free, while VoIP calls connecting to public switched telephone networks (VoIP-to-PSTN), may have a cost that is borne by the VoIP user.

Voice-over-IP systems carry telephony signals as digital audio, typically reduced in data rate using speech data compression techniques, encapsulated in a data packet stream over IP. VoIP services convert our voice into a digital signal that travels over the Internet. If we are calling a regular phone number, the signal is converted to a regular telephone signal before it reaches the destination. VoIP can allow us to make a call directly from a computer, a special VoIP phone, or a traditional phone connected to a special adapter. In addition, wireless "hot spots" in locations such as airports,

parks, and cafes allow us to connect to the Internet and may enable us to use VoIP service wirelessly [5].

There are three different "flavors" of VoIP service in common use today [6]:-

ATA -- The simplest and most common way is through the use of a device called an ATA (analog telephone adaptor). The ATA allow us to connect a standard phone to our computer or our Internet connection for use with VoIP. The ATA is an analog-to digital converter. It takes the analog signal from our traditional phone and converts it into digital data for transmission over the Internet. We simply crack the ATA out of the box, plug the cable from our phone that would normally go in the wall socket into the ATA, and we're ready to make VoIP calls. Some ATA's may ship with additional software that is loaded onto the host computer to configure it; but in any case, it's a very straightforward setup.

IP Phones -- These specialized phones look just like normal phones with a handset, cradle and buttons. But instead of having the standard RJ-11 phone connectors, IP phones have an RJ-45 Ethernet connector. IP phones connect directly to router and have all the hardware and software necessary right onboard to handle the IP call. Wi-Fi phones allow subscribing callers to make VoIP calls from any Wi-Fi hot spot.

Computer-to-computer -- This is certainly the easiest way to use VoIP. We don't even have to pay for long-distance calls. There are several companies offering free or very low-cost software that we can use for this type of VoIP. All we need is the software, a microphone, speakers, a sound card and an Internet connection, preferably a fast one like we would get through a cable or DSL modem. Except for normal monthly ISP fee, there is usually no charge for computer-to-computer calls, no matter the distance.

There are two types of PSTN-to-VoIP services: Direct inward dialing (DID) and access numbers. DID will connect a caller directly to the VoIP user, while access numbers require the caller to provide an extension number for the called VoIP user.

VoIP can facilitate tasks and provide services that may be more difficult to implement or more expensive using the PSTN. Examples include [1]:

- The ability to transmit more than one telephone call over the same broadband connection. This can make VoIP a simple way to add an extra telephone line to a home or office.
- Conference calling, call forwarding, automatic redial, and caller ID; zero- or near-zero-cost features that traditional telecommunication companies (Telco's) normally charge extra for. Secure calls using standardized protocols (such as Secure Real-time Transport Protocol). Most of the difficulties of creating a secure phone connection over traditional phone lines, like digitizing and digital transmission, are already in place with VoIP. It is only necessary to encrypt and authenticate the existing data stream.
- Location independence - Only an Internet connection is needed to get a connection to a VoIP provider. For instance, call center agents using VoIP phones can work from anywhere with a sufficiently fast and stable Internet connection.
- Integration with other services available over the Internet, including video conversation, message or data file exchange in parallel with the conversation, audio conferencing, managing address books, and passing information about whether others (e.g. friends or colleagues) are available to interested parties.
- Advanced Telephony features such as call routing, screen pops, and IVR implementations are easier and cheaper to implement and integrate. The fact that the phone call is on the same data network as a users PC opens a new door to possibilities.

2.2 Session Initiation Protocol

Over the last couple of years, the Voice over IP community has adopted SIP as its protocol of choice for signaling. SIP is an RFC standard (RFC 3261) from the Internet Engineering Task Force (IETF), the body responsible for administering and developing the mechanisms that comprise the Internet. SIP is still evolving and being

extended as technology matures and SIP products are socialized in the marketplace [7].

Session Initiation Protocol (SIP) is an application-layer control (signaling) protocol for creating, modifying, and terminating sessions with one or more participants. These sessions include Internet telephone calls, multimedia distribution, and multimedia conferences. SIP invitations used to create sessions carry session descriptions that allow participants to agree on a set of compatible media types. SIP makes use of elements called proxy servers to help route requests to the user's current location, authenticate and authorize users for services, implement provider call-routing policies, and provide features to users. SIP also provides a registration function that allows users to upload their current locations for use by proxy servers [8]. SIP runs on top of several different transport protocols.

The protocol can be used for creating, modifying and terminating two-party (unicast) or multiparty (multicast) sessions consisting of one or several media streams. The modification can involve changing addresses or ports, inviting more participants, adding or deleting media streams, etc. The SIP protocol is situated at the session layer in the OSI model, and at the application layer in the TCP/IP model. SIP is designed to be independent of the underlying transport layer. SIP clients typically use TCP or UDP to connect to SIP servers and other SIP endpoints. SIP is primarily used in setting up and tearing down voice or video calls. However, it can be used in any application where session initiation is a requirement. These include Event Subscription and Notification, Terminal mobility and so on [9].

There SIP supports five facets of establishing and terminating multimedia communications:

User location: determination of the end system to be used for communication;
User availability: determination of the willingness of the called party to engage in communications;
User capabilities: determination of the media and media parameters to be used;
Session setup: "ringing", establishment of session parameters at both called and calling party;

Session management: including transfer and termination sessions, modifying session parameters, and invoking services.

A simple Session establishment can be show in the figure 2.1:

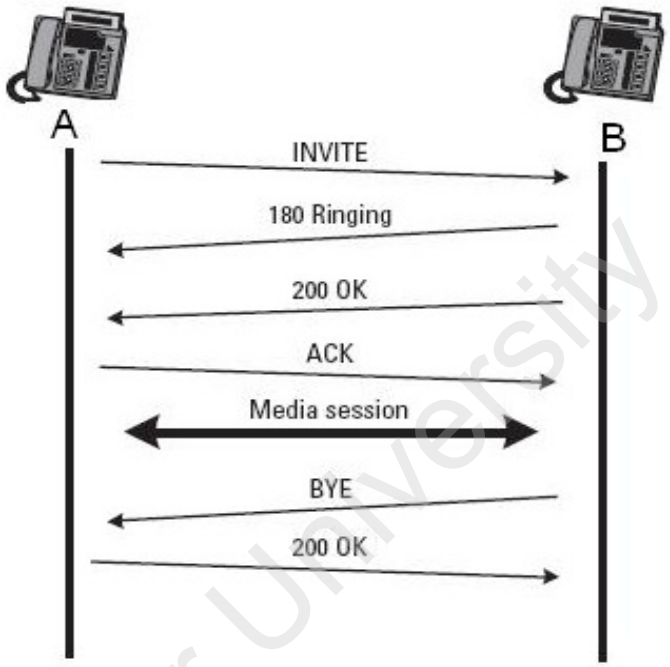

Figure 2.1 Session Establishment [10]

Figure 2.1 shows the SIP message exchange between two SIP-enabled devices. The two devices could be SIP phones, hand-held, palmtops, or cell phones. It is assumed that both devices are connected to an IP network such as the Internet and know each other's IP address.

A motivating goal for SIP was to provide a signaling and call setup protocol for IP-based communications that can support a superset of the call processing functions and features present in the public switched telephone network (PSTN). SIP by itself does not define these features; rather, its focus is call-setup and signaling. However, it has been designed to enable the building of such features in network elements known as Proxy Servers and User Agents. These are features that permit familiar telephone-like operations: dialing a number, causing a phone to ring, hearing ring back tones or a

busy signal. Implementation and terminology are different in the SIP world but to the end-user, the behavior is similar.

Today, software SIP endpoints are common. SIP also requires proxy and registrar network elements to work as a practical service. Although two SIP endpoints can communicate without any intervening SIP infrastructure, which is why the protocol is described as peer-to-peer, this approach is impractical for a public service. There are various implementations that can act as proxy and registrar. SIP makes use of elements called proxy servers to help route requests to the user's current location, authenticate and authorize users for services, implement provider call-routing policies, and provide features to users. Figure 2.2 shows the message exchange when a SIP proxy server is used.

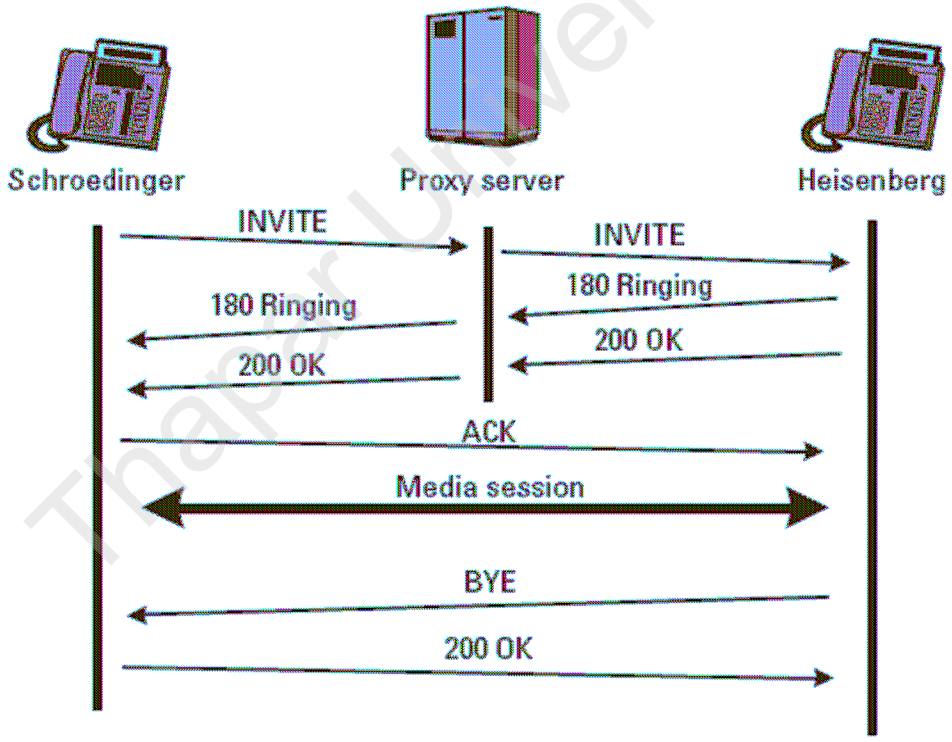

Figure 2.2: SIP call example with proxy server [10]

SIP can be regarded as the enabler protocol for telephony and voice over IP (VoIP) services. The following features of SIP play a major role in the enablement of IP telephony and VoIP [11]:

- Name Translation and User Location: Ensuring that the call reaches the called party wherever they are located. Carrying out any mapping of descriptive information to location information. Ensuring that details of the nature of the call (Session) are supported.
- Feature Negotiation: This allows the group involved in a call (this may be a multi-party call) to agree on the features supported recognizing that not all the parties can support the same level of features. For example video may or may not be supported; as any form of MIME type is supported by SIP, there is plenty of scope for negotiation.
- Call Participant Management: During a call a participant can bring other users onto the call or cancel connections to other users. In addition, users could be transferred or placed on hold.
- Call feature changes: A user should be able to change the call characteristics during the course of the call. For example, a call may have been set up as voice-only, but in the course of the call, the users may need to enable a video function. A third party joining a call may require different features to be enabled in order to participate in the call.
- Media negotiation: The inherent SIP mechanisms that enable negotiation of the media used in a call enable selection of the appropriate codec for establishing a call between the various devices. This way, less advanced devices can participate in the call, provided the appropriate codec is selected.

2.3 XML (Extensible Markup Language)

2.3.1 Introduction

The Extensible Markup Language (XML) is a general-purpose specification for creating custom markup languages [12]. XML is a pared-down version of SGML, designed especially for Web documents. It allows designers to create their own customized tags, enabling the definition, transmission, validation, and interpretation of data between applications and between organizations [13]. Its primary purpose is to facilitate the sharing of structured data across different information systems, particularly via the Internet, and it is used both to encode documents and to serialize data. XML offers a widely adopted standard way of representing text and data in a

format that can be processed without much human or machine intelligence. Information formatted in XML can be exchanged across platforms, languages, and applications, and can be used with a wide range of development tools and utilities [14]. It started as a simplified subset of the Standard Generalized Markup Language (SGML), and is designed to be relatively human-legible. XML is recommended by the World Wide Web Consortium. It is a fee-free open standard. The W3C recommendation specifies both the lexical grammar and the requirements for parsing.

2.3.2 Degree of correctness of XML Documents [15]

There exist two classes of correctness for XML documents:

Well-formed documents

The first class forms the well formed documents. These documents meet the syntax rules of XML1.0 specification. If the document doesn't meet these syntax rules, it's not a XML document and is rejected by an XML processor. Some xml syntax rules are:

- Every document must have a single unique root element that encloses all other elements.
- All elements must be correctly nested.
- All elements must have corresponding start and end tags.
- Attribute values must be enclose with in single or double quotes.

Valid Documents

The second class forms the valid documents. These documents are well formed and meet in addition the constraints of a DTD or schema. For example, if a document contains an undefined element, then it is not valid; a validating xml processor is not allowed to process it. This class of correctness is required if XML documents must be exchanged between applications reliably. Only validating XML processor perform these checks, non-validating XML Processors ignore the DTD.

2.3.3 Benefits of using XML [16]

XML offers many advantages like:

Simplicity-XML documents are built upon a core set of basic nested structures. While the structures themselves can grow complex as layers and layers of detail are added, the mechanisms underlying those structures require very little implementation effort, from either authors or developers. Furthermore, XML rigid set of rules helps make documents more readable to both humans and machines.

Extensibility- XML is extensible in two senses. First, it allows developers to create their own Document Type Definition (DTD), effectively creating 'extensible' tag sets that can be used for multiple applications. Second, XML itself is being extended with several additional standards that add styles (e.g. XSL), linking, and referencing ability to the core XML set of capabilities.

Interoperability.-XML can be used on a wide variety of platforms and interpreted with a wide variety of tools. Because the document structures behave consistently, parsers that interpret them can be built at relatively low cost in any of a number of languages such as C++, C, JavaScript, Tcl, and Python.

Openness- XML documents are considerably open and anyone can parse a well formed XML document.

Applications-XML can be used in a couple of different ways. One is for data interchange between humans and machines, such as from a Web server to a user's browser. The other is for data exchange between applications, or from machine to machine.

2.3.4 DTD

DTD is the grammar of an XML page. It is an acronym that stands for Document Type Definition. It contains the elements, attributes, entities, and notations used in the XML document [17]. The purpose of a DTD (Document Type Definition) is to define the legal building blocks of an XML document. It defines the document structure with a list of legal elements and attributes. A DTD can be declared inline inside an XML document, or as an external reference [18]. XML provides an application independent

way of sharing data. With a DTD, independent groups of people can agree to use a common DTD for interchanging data. Our application can use a standard DTD to verify that data that we receive from the outside world is valid. We can also use a DTD to verify our own data. In a DTD, the structure of a class of documents is described via element and attribute-list declarations. Element declarations name the allowable set of elements within the document, and specify whether and how declared elements and runs of character data may be contained within each element. Attribute-list declarations name the allowable set of attributes for each declared element, including the type of each attribute value, if not an explicit set of valid value [19].

2.3.5 Schema

XSD (XML Schema Definition), a Recommendation of the World Wide Web Consortium (W3C), specifies how to formally describe the elements in an Extensible Markup Language document. This description can be used to verify that each item of content in a document adheres to the description of the element in which the content is to be placed. In general, a schema is an abstract representation of an object's characteristics and relationship to other objects. An XML schema represents the interrelationship between the attributes and elements of an XML object (for example, a document or a portion of a document) [20]. XML Schema is an XML-based alternative to DTD which defines elements that can appear in a document, attributes that can appear in a document, which elements are child elements, the order and number of child elements, data types for elements and attributes, default and fixed values for elements and attributes and also defines whether an element is empty or can include text and data types.

2.3.6 Comparison of XSD and DTD

XML Schemas are the Successors of DTDs [21]. XML Schemas are used in most Web applications as a replacement for DTDs. Here are some reasons:

- XML Schemas are extensible to future additions. So we can reuse our Schema in other Schemas ,create our own data types derived from the standard types and also reference multiple schemas in the same document.
- XML Schemas are richer and more powerful than DTDs.

- XML Schemas are written in XML which means that it doesn't require intermediary processing by a parser and we don't have to learn a new language.
- XML Schemas support multiple data types so it is easier to convert data between different data types, easier to work with data from a database, easier to define data facets (restrictions on data) and easier to define data patterns (data formats).
- XML Schemas support namespaces. XML namespaces provide a simple method for qualifying element and attribute names used in Extensible Markup Language documents by associating them with namespaces identified by URI references.

2.4 Parsing

2.4.1 Introduction

Parsing is the act of splitting up information into its component parts and to analyze it in an orderly way, as analyzing the grammatical structure of a sentence, a character string, or a line of code and separate it into its parts [22].

A parser breaks data into smaller elements, according to a set of rules that describe its structure. Most data can be decomposed to some degree. For example, a phone number consists of an area code, prefix and suffix [23].

(800) 555-1234
(123) 555-4321
(999) 888-7777

The structure of these items may be described informally as: "A phone number consists of a three-digit area code, enclosed by parentheses, followed by a three-digit prefix, followed by a dash, followed by a four-digit suffix." This description can be expressed more formally as a grammar. A grammar is a set of rules that describe the structure, or syntax, of a particular type of data. The following grammar describes the syntax of phone numbers:

phone_number ::= "(" area_code ")" prefix "-" suffix;

area_code ::= numeric<3>;

prefix ::= numeric<3>;

suffix ::= numeric<4>;

Once the syntax of a data source has been described by grammar rules, a parser can use the grammar to parse the data source; that is, to break data elements such as phone numbers into smaller elements, such as area codes. The output of the parser is a parse tree. The parse tree expresses the hierarchical structure of the input data. For example, the following parse tree is generated when phone number "(800) 555-1234" is parsed, using the grammar shown above:

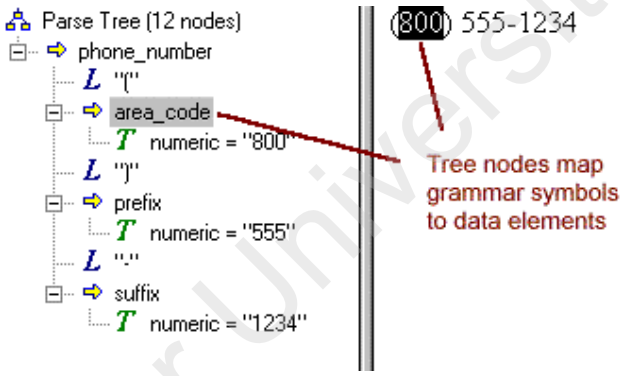

Figure 2.3: Parse Tree

Parsing is the process of matching grammar symbols to elements in the input data, according to the rules of the grammar. The resulting parse tree is a mapping of grammar symbols to data elements. Each node in the tree has a label, which is the name of a grammar symbol; and a value, which is an element from the input data. Parsing is a very important part of many computer science disciplines. For example, compilers must parse source code to be able to translate it into object code. Likewise, any application that processes complex commands must be able to parse the commands. This includes virtually all end-user applications [24].

In computer technology, a parser is a program, usually part of a compiler, that receives input in the form of sequential source program instructions, interactive online commands, markup tags, or some other defined interface and breaks them up into parts (for example, the nouns (objects), verbs (methods), and their attributes or

options) that can then be used for developing further actions or for creating the instructions that form an executable program[25].

All applications that read input have a parser of some kind, otherwise they'd never be able to figure out what the information means. Microsoft Word contains a parser which runs when we open a .doc file and checks that it can identify all the hidden codes. Give it a corrupted file and we'll get an error message. XML applications are just the same: they contain a parser which reads XML and identifies the function of each the pieces of the document i.e. information depicted by content of all elements and attributes and it then makes that information available in memory to the rest of the program.

2.4.2 Definition of XML parser

Software that reads an XML document identifies all the XML tags, validates the tags against an XML schema or DTD and passes the data to the application [26].

An XML parser is the piece of software that reads XML files and makes the information from those files available to applications and programming languages. The XML parser is responsible for testing whether a document is well-formed and, if given a DTD or XML schema, it will also check for validity of documents (i.e., it determines if the document follows the rules of the DTD or schema) and provide application [27].

2.4.3. Categories of Parser

XML can be used on a wide variety of platforms and interpreted with a wide variety of tools. Because the document structures behave consistently, parsers that interpret them can be built at relatively low cost in any of a number of languages such as C++, C, JavaScript, Tcl, and Python.

XML parsers come in two flavours [28]:

Non-Validating Parser

Non-Validating parser does not check a document against any DTD or Schema, it only checks that the document is well-formed, i.e., the document is properly marked

up according to XML syntax rules.

Validating Parser

Validating parsers perform more rigorous checks. In addition to check whether document is well-formed, the parser also verifies that the document conforms to a specific DTD (either internal or external to the XML file being parsed) or Schema and report violations of the constraints expressed by the declarations in the DTD, and failures to fulfill the validity constraints given in this specification. To accomplish this, validating XML processors must read and process the entire DTD or Schema and all external parsed entities referenced in the document.

2.4.4 Types of Parsers

An XML parser is a tool or library, basically a piece of software that checks if the document is syntactically well-formed and some parsers are also able to check the document's validity by running additional checks on it. There are two types of parsers, one is event based parsers which allow the application to process the contents during paring whereas tree based parsers only reveal contents after parsing is complete [29].

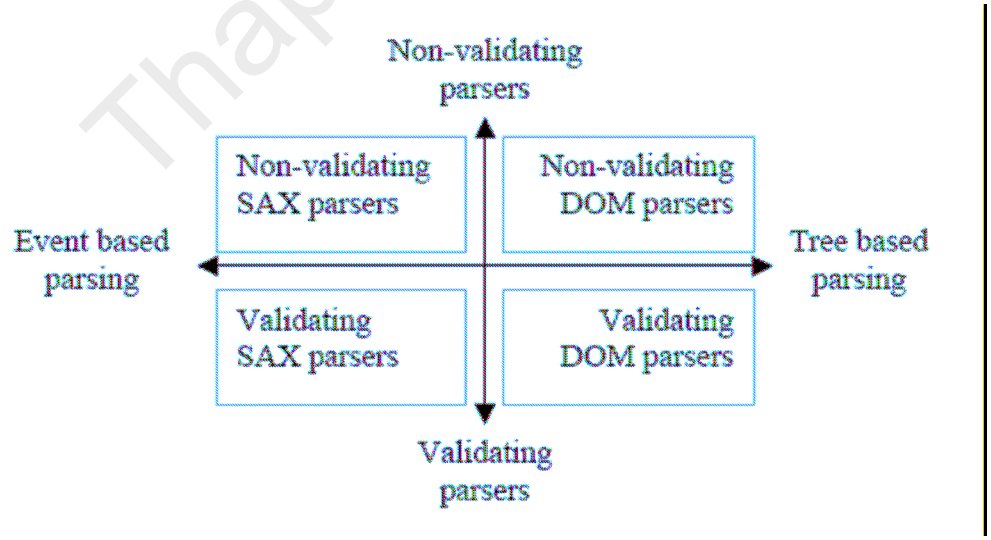

Figure 2.4: Types of XML Parsers [29]

Event-Based Parsing

Event-based parsers provide a data-centric view of XML. When an element is encountered, the idea is to process it and then forget about it. The event-based parser returns the element, its list of attributes, and the content. This is more efficient for many types of applications, especially searches. It requires less code and less memory since there is no need to build a large tree in memory as a particular element, attribute, and/or content sequence in an XML document is scanned [30]. An event-based API, reports parsing events (such as start and end of elements) directly to the application through callbacks, and does not usually build an internal tree. The application implements handlers to deal with the different events, much like handling events in a graphical user interface. SAX is the best known example of such API [31].

SAX Parser:

- SAX (Simple API for XML) parser does not create any internal structure. Instead, it takes the occurrences of components of an input document as events, and tells the client what it reads as it reads through the input document.
- It traverses the entire document from beginning to end, notifying the application that runs it each time it recognizes a syntax construction. This is done by means of call back methods contained in the event handler interfaces ContentHandler, ErrorHandler, DTDHandler and EntityResolver.
- A SAX parser serves the client application always only with pieces of the document at any given time.

Figure 2.5 depicts the relationship between various components that go into parsing an XML document with a SAX parser [32].

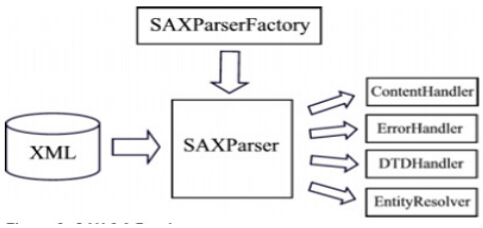

Figure 2.5: SAX Parser

Tree-Based Parsing

On the other hand, tree-based parsers (DOM) provide a document-centric view of XML. In tree-based parsing, an in-memory tree is created for the entire document, which is extremely memory-intensive for large documents. All elements and attributes are available at once, but not until the entire document has been parsed. This technique is useful if we need to navigate around the document and perhaps change various document chunks. Tree-based APIs map an XML document into an internal tree structure, and then allow an application to navigate that tree. The Document Object Model (DOM) working group at the World-Wide Web Consortium (W3C) maintains a recommended tree-based API for XML and HTML documents, and there are many such APIs from other sources [31].

DOM Parser:

- A DOM (Document Object Model) parser creates a tree structure in memory from an input document and then waits for requests from client.
- It is a set of interfaces that parse an XML document into a tree structure of objects and allows for random access to particular pieces of data from the XML document which is not possible with a SAX parser.
- A DOM parser always serves the client application with the entire document no matter how much is actually needed by the client.

Figure 2.6 depicts the tree structure created by DOM: [32]

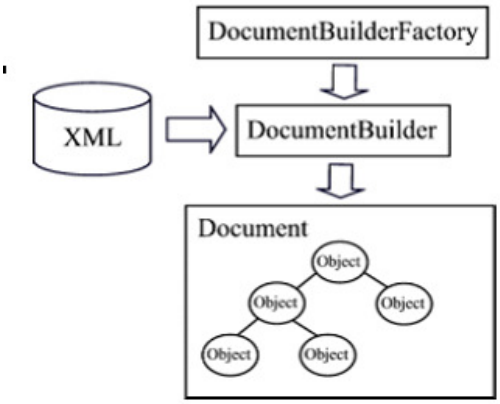

Figure 2.6: DOM Parsing

Comparison of SAX and DOM parser

A SAX parser, is much more space efficient in case of a big input document (because it creates no internal structure). It runs faster and is easier to learn than DOM parser because its API is really simple. But from the functionality point of view, it provides a fewer functions, which means that the user themselves have to take care of more, such as creating their own data structures.

The DOM parser offers a convenient way for reading, analyzing, manipulating and writing back XML files. Since it always reads the whole file, before further processing can take place, using the DOM parser may lead to difficulties when processing huge XML files. The SAX parser, on the other hand, does not generate a data representation of the XML content, so there is some more programming required, compared to the DOM parser. A DOM parser is rich in functionality. It creates a DOM tree in memory and allows you to access any part of the document repeatedly and allows you to modify the DOM tree.

2.4.5 Parsing Process

A general interaction between an application and XML Parser is shown in the figure 2.7.

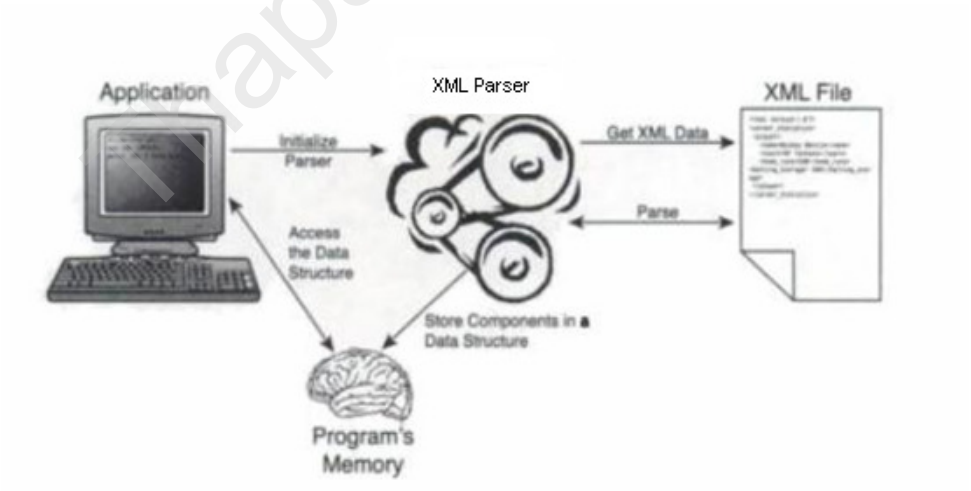

Figure 2.7: Parsing Process [33]

An application creates an instance of a parser by invoking its initialization routine. The parser Input module then takes XML file and DTD (incase of validating parser) as input. It then gives the input to the Processing Engine. The Processing Engine then checks for well formedness of the XML data and validates the XML against the DTD. It then populates the tag, value and attributes into its internal structure and returns it back to the application. The application can then choose how it wants to use the data.

2.4.6 XML Parsers in Use

An XML parser is an essential tool for any serious programming effort involving XML documents. By embedding the XML parser in the SIP server code, server XML document management and few other services can be provided. Most of the SIP entities available today make use of a non-validating parser. Since there are many benefits associated with the usage of a validating parser and the predominant non-usage of validating parsers in SIP frameworks prevents the leveraging of these benefits, it is seen as a gap that needs to be bridged. Hence as part of this work it is proposed to analyze the possibility of using validating parser in SIP environment.

HXML, expat, libxml, MXP1 are few examples of some of the non-validating parsers. Parsifal, RXP, Xerces, IBM's XML for Java EA2, fxp, Oracle's XML Parser for Java v2 are few examples of validating XML parsers. As the SIP server code is totally written in C, it is better to choose an XML parser also written in C. There are many non-validating parsers in C like expat, libxml etc. The expat XML parser by James Clark is fairly small, written in C, runs on UNIX or W32 for parsing XML documents [34]. Expat is a stream-oriented parser. To use the Expat library, programs first register handler functions with Expat. When Expat parses an XML document, it calls the registered handlers as it finds relevant tokens in the input stream. These tokens and their associated handler calls are called events. Typically, programs register handler functions for XML element start or stop events and character events [35].

Expat proves itself to be quite capable, but it is not without limitations. Expat was among the first XML parsers available and, as a result, its interfaces reflect the expectations of users at the time it was written. Expat does not implement the DOM, SAX, or XPath language interfaces (things that most modern XML users take for

granted) because either the given interface did not exist or was still being heavily evaluated and not considered "standard" at the time it was written.

By contrast libxml2 another XML C parser and toolkit was developed for the Gnome project. It is free software available under the MIT License. It was written after the DOM, XPath, and SAX interfaces became common, and so it implements all three. In-memory trees can be built by parsing documents stored in files, strings, and so on, or generated from a series of SAX events. Those trees can then be operated on using the W3C DOM and XPath interfaces or used to generate SAX events that are handed off to external event handlers. This added flexibility, which reflects current XML processing expectations, makes LibXML a strong contender [36].

Both expat and libxml are non-validating parsers. These parsers do not check a document against any DTD, they only checks if the document is well-formed, i.e., if the document is properly marked up according to XML syntax rules. A Validating parser in addition to checking whether a document is well-formed also verifies if the document conforms to a specific DTD (either internal or external to the XML file being parsed). Validating parsers available in C are Parsifal and RXP. Since Parsifal is a SAX based parser it reports parsing events (such as the start and end of elements) directly to the application through callbacks so RXP has been chosen. It is a Validating DOM parser for evaluation.

RXP is a validating XML parser available under the GPL. It is written in C language and validates XML against DTD. It is thread-safe and supports Unicode, ISO 8859-1, ISO 8859-9 and XML Namespaces. RXP supports internationalization. (Char data type whose pointer is used to store input is 8 or 16 byte depending on compile time flag option). It currently supports Win 32, Solaris and Linux platform [37].

Features of RXP

- RXP works by building a compact in-memory tree structure of the XML document being parsed. Failures in parsing are failures in tree building, and a successful parse gives a data structure that is like a DOM representation of XML.

- RXP provides feature to Insert declared default values for omitted attributes and also provides verbose mode.
- If there is any validity/well-formedness errors then RXP exit after the first validity error. However, RXP provides feature by which just warning can be printed on an error condition. If the document is well-formed but not valid, RXP returns 2. If the document is not well-formed, or a system error occurs, 1 is returned. Otherwise 0 is returned.
- RXP provides namespace validation as well.
- RXP can be run in silent mode. It will be useful for benchmarking.
- RXP uses DTD for validation. Schema support is not there.

CHAPTER 3
PROBLEM DEFINITION

During the literature survey it was identified that XML parsing capabilities are essential to build a feature rich SIP entity. It was also seen that the usage of non-validating parsers was predominant and replacing these with validating parsers offered a lot of benefits like reduced code size, lesser efforts and reduced time to market.

As part of this thesis work it is proposed to make use of RXP – a validating XML parser written in C to parse SIP message bodies and include a module on top of it that ensures that validation of content type in XML messages against a schema is performed and support for namespaces is included. This is because RXP inherently performs validations only against a DTD and a limitation of DTD is that it does not support multiple data types i.e does not provide a check on the validity of content type of elements (i.e restrictions on data) and does not support namespaces which are required for qualifying element and attribute names used in Extensible Markup Language documents by associating them with namespaces identified by URI references. So it leads to building an additional module as RFC's that needs to be implemented need content validation and support for namespaces. Schema to DTD conversion is one time activity and can be done manually or using a tool (like xmlspy).

Also, as part of this work it is planned to integrate RXP parser (with the newly included module described above) into a SIP framework so as to enable the SIP entity perform protocol specific behavior specified by the request for comments RFC 3858, RFC 3994, RFC 4661 and RFC 4662. In order to display the behavior mentioned in the above listed RFCs, it is important that the SIP entity have XML parsing capabilities, for which purpose the RXP parser is to be integrated. Precisely, this work plans to use RXP to parse XML content in SIP messages and provide the inputs to a SIP entity for further processing. XML message bodies need to be validated, parsed and the inputs provided to the SIP entity by means of filling the relevant data structures.

CHAPTER 4
DESIGN AND IMPLEMENTATION

The XML parsing done is based on an open source named RXP validating C parser. So, to use and to implement extra features, there is a need to first understand the architecture of RXP and then go for its further integration.

4.1 Design RXP Architecture

The following diagram shows the architecture of the RXP parser. It shows the important modules that the parser has and how these modules are interconnected with each other. So first thing that needs to be done is understand the functionality of RXP parser and its architecture to understand how the various modules of RXP interact with each other.

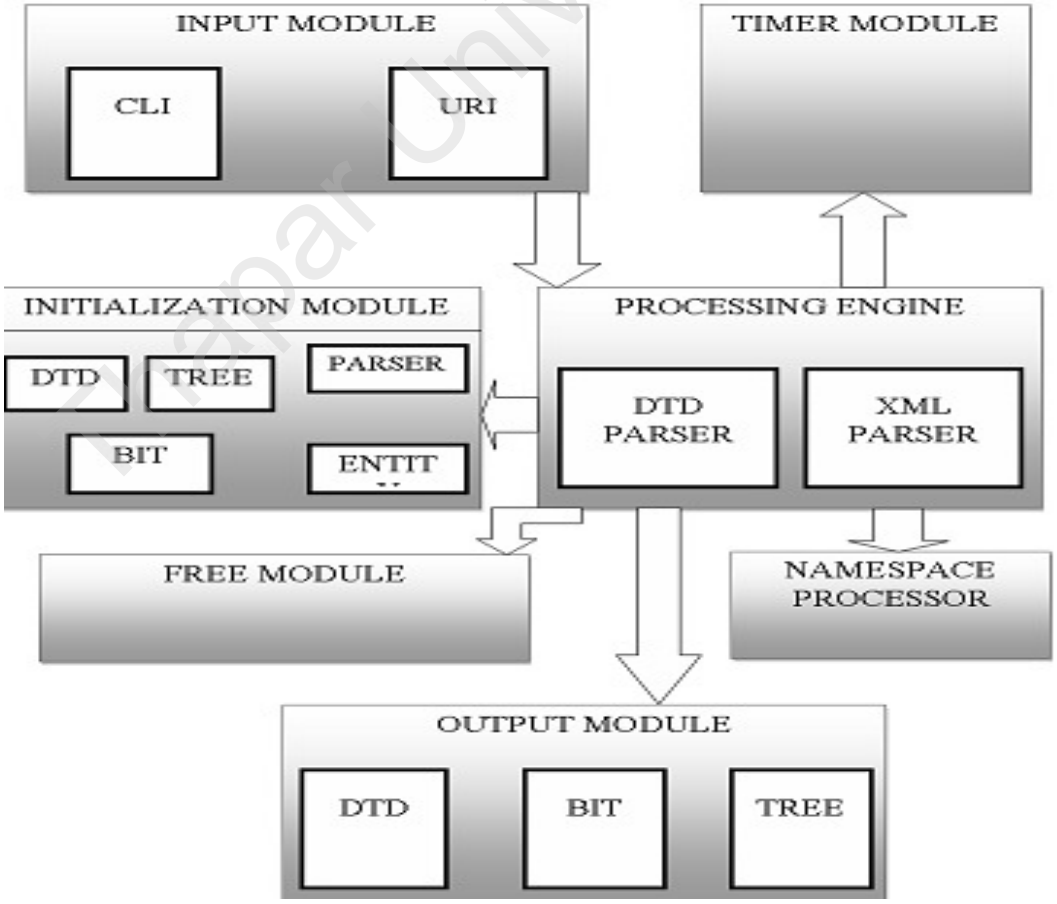

Figure 4.1: RXP Architecture

The different modules present in RXP are Input, Timer, Initialization, Processing, Free and Namespace module. The functionality of these modules is explained below:

Input module: This module is used to accept input by RXP. Input can be obtained in two ways Command Line Input or in the form of Uniform Resource Locator. It can create an input source referring to a given entity and stream. It is only intended for direct use by the user if the parser's entity opener has been set. It can be used to get a source when the input is not an entity but an existing open stream. A fake entity is created with the description as its system ID. If the description contains a slash character, it will be used as the entity's base URL, so if we know where the stream came from, we can pass in its URL as the description; otherwise use something like "stdin" so that the user gets reasonable error messages. It can also be used to return a FILE16 connected to the specified URL obtained by using EntityURL which returns the URL of an entity if it has one or more URL obtained by merging its system ID with the URL of any parent entity by calling function url_merge(). It uses function EntityBaseURL () to get the base URL for an entity. It can be used to merge a URL with a base URL. The merged URL is returned. If base, scheme, host, port and path are non-null, the parts of the merged URL are returned in them.

Timer module: It is used to check whether the CPU time has been exceeded. It checks if the time limit is exceeded and prints a message.

Initialization module: This module implements all initialization functions. It creates some internal entities. It initializes Charset, namespaces, input/output and url. It prints the rxp version. It can initialize the character set and also makes arrays which help in conversion of ISO and Unicode encodings. It checks for legal characters. It checks for the Legal ASCII characters, letters, digits, whitespaces, and Unicode characters. But the character size must be 16-bit type. It can initialize the pointers for the input, output and error streams. It is used to pass the mode, encoding and stream to some functions. It checks whether proxy needs to be used. It checks the protocol, domain and also the port number from the URI mentioned. It can initialize the namespace structure. It lists URI of the namespace, and lists of the element types and global attributes in the namespace. Attribute definitions in a namespace are represented by

NSAttributeDefinition structures. Element types in a namespace are represented by NSElementDefinition structures. It creates a new internal entity by filling the Entity structure. It fills various members of the Entity structure such as type, name, parent, next, systemid and encoding. It creates a new external entity by calling function. It creates a new external entity by filling various members of the Entity structure such as type, name, notation, parent, next, systemid and encoding. It creates a new instance of parser by dynamically allocating memory to it and then initializes all its members like state, seen_validity_error, document entity, standalone, source etc. It creates a new dtd and allocates memory to it dynamically and initializes all its members like name, internal_part, entities used and notations.

Processing module: Processing engine is the main module that parses the message body received. It provides support for parsing and encoding. It pushes an input source onto the parser's input stack. It checks encoding even if we have already determined it for this entity. It returns true if the encoding has 8-bit input units and is the same as ASCII for characters <= 127. It processes entities representing the DOCTYPE declaration, created when an XBIT_dtd is returned. The entities are opened. This function allows you to intercept entity opening with a callback. It looks up an encoding by name. It understands various aliases (ISO-Latin-1 for ISO-8859-1 for example). It returns CE_unknown if the name is not recognized. It reads a whole tree, i.e., if the next bit is a start bit, all the following bits are read until the end bit is encountered. The "nchildren" field of the returned bit contains the number of children of the node, and they are stored in the children field as bit->children [0] ... bit->children [bit->nchildren-1], and so on recursively. It reads the next bit from a document. It is used to parse the XML file. It calls various functions used to parse the character references, pcdata and comments. It can read the next bit without consuming it. It can compare the names of two notation definitions and also names of two entities.

Free module: This module implements all freeing functions which free the memory associated with Bits, Tree and Dtd. It is used to free the source. It frees the memory associated with an XBit structure. It frees a tree of XBits. It frees the DTD which is created dynamically. It frees the Entity created. It frees an element definition and its attribute definitions by freeing the element name, attributes and content model. It is

used to free the notation name, id's and the definition. It frees parser structure that was created dynamically. It is used to delete a checker object. It is used to free the hash table created. It also frees the memory when the library is no longer required.

Output module: This module prints the output onto the device. It is used to print the attributes of an element. It searches for total number of attributes for a particular ElementDefinition and then prints the array which holds those attributes. It is used to print the attributes which are present in the namespaces. It is used to print the namespaces used by the XML file. It is used to print the characters inside the text. It is used to print substitutions for characters like <, > and &. Then it prints the parser instance in a tree format by calling itself recursively till an end bit is received. It takes the parser and bit as input and checks for the output format. If it is "o_bits", then it checks for the type of bit and prints it by calling appropriate functions.

4.2 Schema to DTD Conversion

In this step DTD is prepared for all RFC's (3858, 3994, 4661, and 4662). These DTD's must be prepared based on the schema mentioned in each particular RFC. The conversion of schema to DTD is required as RXP validates xml messages against a DTD when input comes to SIP entity. So for all four RFC's DTD needs to be prepared from schema given in the RFC's.

4.2.1 RFC 3858 [An Extensible Markup Language (XML) Based Format for Watcher Information] [38]

Watchers are defined as entities that request (i.e., subscribe to) information about a resource, using the SIP event framework. There is fairly complex state associated with these subscriptions. This state includes the identity of the subscriber, the state of the subscription, and so on. The union of the state for all subscriptions to a particular resource is called the watcher information for that resource. This state is dynamic, changing as subscribers come and go. As a result, it is possible, and indeed useful, to subscribe to the watcher information for a particular resource. In order to enable this, a format is needed to describe the state of watchers on a resource. This specification

describes an Extensible Markup Language (XML) document format for such state. An important application of this is the ability for a user to find out the set of subscribers to their presentity. This would allow the user to provide an authorization decision for the subscription. Watcher information is an XML document that MUST be well-formed and SHOULD be valid. Watcher information documents MUST be based on XML 1.0 and MUST be encoded using UTF-8. This specification makes use of XML namespaces for identifying watcher info documents and document fragments. The namespace URI for elements defined by this specification is a URN, using the namespace identifier 'ietf' defined by and extended by. This URN is:Urn: ietf: params: xml: ns: watcherinfo.

DTD obtained for watcher info document:-

<! ELEMENT watcherinfo (watcher-list*)>
<! ATTLIST watcherinfo version CDATA #REQUIRED>
<! ATTLIST watcherinfo state (full|partial) #REQUIRED>
<! ELEMENT watcher-list (watcher*)>
<! ATTLIST watcher-list resource CDATA #REQUIRED>
<! ATTLIST watcher-list package CDATA #REQUIRED>
<! ELEMENT watcher (#PCDATA)>
<! ATTLIST watcher id CDATA #REQUIRED>
<! ATTLIST watcher status (pending|active|waiting|terminated) #REQUIRED>
<!ATTLIST watcher event (subscribe|approved|decativated|probation|rejected|timeout |giveup|noresource) #REQUIRED>
<! ATTLIST watcher display-name CDATA #IMPLIED>
<! ATTLIST watcher expiration CDATA #IMPLIED>
<! ATTLIST watcher duration-subscribed CDATA #IMPLIED>

4.2.2 RFC 3994 [Indication of Message Composition for Instant Messaging] [39]

In Instant Messaging (IM) systems, it is useful to know during an IM conversation whether the other party is composing a message; e.g., typing or recording an audio message. A user composes a message by typing, speaking, or recording a video clip. This message is then sent to one or more recipients. Unlike email, instant messaging

is often conversational, so the other party is waiting for a response. If no response is forthcoming, a participant in an Instant Messaging conversation may erroneously assume either the communication partner has left or it is her turn to type again, leading to two messages "crossing on the wire". To avoid this uncertainty, a number of commercial instant messaging systems feature an "is-typing" indication sent as soon as one party starts typing a message. In this document, a generalized version of this indication is described; called the isComposing status message. A status message is delivered to the instant message recipient in the same manner as are the messages themselves. The isComposing status messages can announce the composition of any media type, not just text. The status messages are carried as XML, as instances of the XML schema and labeled as an application/im-iscomposing+xml content type.

DTD obtained for isComposing document:

<! ELEMENT isComposing (state, lastactive? ,contenttype?, refresh?)>
<! ELEMENT state (#PCDATA) >
<! ELEMENT lastactive (#PCDATA)>
<! ELEMENT contenttype (#PCDATA)>
<! ELEMENT refresh (#PCDATA)>

4.2.3. RFC 4661 [An Extensible Markup Language (XML) Based Format for Event Notification Filtering] [40]

The SIP event notification framework describes the usage of the Session Initiation Protocol (SIP) for subscriptions and notifications of changes to a state of a resource. The document does not describe a mechanism whereby filtering of event notification information can be achieved. Filtering is a mechanism for defining the preferred notification information, referred to as content, to be delivered and for specifying the rules for when that information should be delivered. In order to enable this, a format is needed to enable the subscriber to describe the state changes of a resource that cause notifications to be sent to it and what those notifications are to contain. This document presents a format to describe filter criteria in the form of an XML document. The XML document contains the user's preference as to when notifications are to be sent to it and what they are to contain. The scope of the "when" part is

triggering. The triggering is defined as enabling a subscriber to specify triggering rules for notifications where the criteria are based on changes of the event package specific state information. The filtering mechanism is expected to be particularly valuable and primarily applicable to users of mobile wireless access devices. The characteristics of the devices typically include high latency, low bandwidth, low data processing capabilities, small display, and limited battery power. Such devices can benefit from the ability to filter the amount of information generated at the source of the event notification.

DTD obtained for simple-filter document:-

```
<! ELEMENT filter-set (ns-bindings?, filter+)>
<! ATTLIST filter-set package CDATA #IMPLIED >
<! ELEMENT ns-bindings (ns-binding+)>
<! ELEMENT ns-binding (#PCDATA)>
<! ATTLIST ns-binding prefix CDATA #REQUIRED>
<! ATTLIST ns-binding urn CDATA #REQUIRED>
<! ELEMENT filter (what?, trigger*)>
<! ATTLIST filter id CDATA #REQUIRED>
<! ATTLIST filter uri CDATA #IMPLIED>
<! ATTLIST filter domain CDATA #IMPLIED>
<! ATTLIST filter remove (true|false) "false" >
<! ATTLIST filter enabled (true|false) "true" >
<! ELEMENT what (include*, exclude*)>
<! ELEMENT include (#PCDATA)>
<! ELEMENT exclude (#PCDATA)>
<! ATTLIST include type (xpath|namespace) "xpath" >
<! ATTLIST exclude type (xpath|namespace) "xpath" >
<! ELEMENT trigger (changed*, added*, removed*)>
<! ELEMENT changed (#PCDATA)>
<! ELEMENT added (#PCDATA)>
<! ELEMENT removed (#PCDATA)>
<! ATTLIST changed from CDATA #IMPLIED>
<! ATTLIST changed to CDATA #IMPLIED>
```

<! ATTLIST changed by CDATA #IMPLIED>

4.2.4. RFC 4662 [SIP Event notification extension for resource lists] [41]

The SIP-specific event notification mechanism allows a user (the subscriber) to request to be notified of changes in the state of a particular resource. This is accomplished by the subscriber generating a SUBSCRIBE request for the resource, which is processed by a notifier that represents the resource. In many cases, a subscriber has a list of resources they are interested in. Without some aggregating mechanism, this will require the subscriber to generate a SUBSCRIBE request for each resource about which they want information.

When users wish to subscribe to the resource of a list of resources, the first step is the creation of a resource list. This resource list is represented by a SIP URI. The list contains a set of Uri's, each of which represents a resource for which the subscriber wants to receive information. The resource list can exist in any domain. The list could be manipulated through a web page, through a voice response system, or through some other protocol. To learn the resource state of the set of elements on the list, the user sends a single SUBSCRIBE request targeted to the URI of the list. This will be routed to an RLS resource list server for that URI. The RLS acts as a notifier, authenticates the subscriber, and accepts the subscription. The RLS may have direct information about some or all of the resources specified by the list. If it does not, it could subscribe to any non-local resources specified by the list resource. As the state of resources in the list change, the RLS generates notifications to the list subscribers. The RLS can, at its discretion, buffer notifications of resource changes and send the resource information to the subscriber in batches, rather than individually. This allows the RLS to provide rate limiting for the subscriber. The list notifications contain a body of type multipart/related. The root section of the multipart/related content is an XML document that provides meta-information about each resource present in the list. The remaining sections contain the actual state information for each resource.

DTD obtained for RLMI (Resource List Meta-Information) document:-

<! ELEMENT list (name*, resource*)>
<! ATTLIST list uri CDATA #REQUIRED>

<! ATTLIST list version CDATA #REQUIRED>

<! ATTLIST list fullState (true|false) #REQUIRED>

<! ATTLIST list cid CDATA #IMPLIED>

<! ELEMENT resource (name*, instance*)>

<! ELEMENT name (#PCDATA)>

<! ELEMENT instance EMPTY>

<! ATTLIST resource uri CDATA #REQUIRED>

<! ATTLIST instance id CDATA #REQUIRED>

<! ATTLIST instance state (active|pending|terminated) #REQUIRED>

<! ATTLIST instance reason CDATA #IMPLIED>

<! ATTLIST instance cid CDATA #IMPLIED>

4.3 Input to RXP (XML Message + DTD)

The XML file and the DTD can be provided to the RXP parser for checking the syntax and also for validation against a DTD. The xml file is taken as input in the form of command line argument or as an uri. Then, this is passed to the initialization module where the entity and parser structures are initialized. There is also a timer module to check whether the execution exceeds time limit or not. Then processing is done where it has an XML and a DTD parser where both the DTD and XML are parsed and validation is done. The namespace processor is used to handle the namespaces used in the XML file. Then, the output is passed to the output module where it can be displayed. There is a module to free the structures used such as entity and parser structures. The RXP does the syntax check and validation, stores in a bit structure. This bit structure has a parent-child relationship and it has all the data relevant to a particular element of the xml like number of children, name of the element, location where the child element is stored and information about attributes like name and its value. This bit structure is very important for further processing.

4.4 Defining Data Structures

The next step is to define data structures for all four RFC's defined above so that these data structures get filled and can be used by the application. The elements and

attributes in the data structures get populated and can be returned to the application. The application can then choose how it wants to use the data.

RFC 3858

Watcher is a singly linked list as there can be unbounded number of watcher elements inside each watcherlist element. The attributes are Display Name, Status, Event, Expiration, Id and Duration Subscribed. All these attributes are represented by char* because all the above attributes have string as their data types. It does not have any subelements.

```
struct watcher
{
        char *DisplayName;
        char *Status;
        char *Event;
        char *Expiration;
        char *Id;
        char *DurationSubscribed;
        char *Value;
        struct watcher *next;
};
typedef struct watcher Watcher;
```

WatcherList is a singly linked list as there can be unbounded number of watcher list elements in each watcherinfo structure. This list has attributes resource and package of string type and so they have been declared as char*. It has unbounded number of watcher sub elements.

```
struct watcherlist
{
        char *Resource;
        char *Package;
        struct watcherlist *next;
```

```
        Watcher *wat;
};
typedef struct watcherlist WatcherList;
```

Watcher Info is the main structure and it has an unbounded number of watcher list sub elements. This structure has attributes xmlversion, encoding, version and state of string type and so they have been declared as char*.

```
typedef struct watcherinfo
{
        char *xmlversion;
        char *encoding;
        char *version;
        char *state;
        WatcherList *watlist;
} WatcherInfo;
```

RFC 3994

The att structure is a structure for storing attributes of all elements. This structure has name and value fields of string type represented as char* and these hold the attribute name and value of all elements

```
struct att
{
        char *name;
        char *value;
};
```

The common is a structure which is a structure for all elements present in the xml document. It has fields for storing the element name and content. It also has a child field to store the elements in the form of a linked list.

```
struct common
{
        char *name;
        char *value;
        struct att *attribute;
        struct common *child;
};
```

RFC3994 is the main structure and it has state, lastactive, contenttype and refresh elements. This structure has fields like version and encoding of string type to store the version and encoding of the xml document.

```
struct rfc3994
{
        char *version;
        char *encoding;
        struct common *state;
        struct common *lastactive;
        struct common *contenttype;
        struct common *refresh;
};
struct rfc3994 *RFC3994;
```

RFC 4661

Include is a singly linked list as there can be unbounded number of include elements inside what structure. This list has attribute type of string type and so they have been declared as char*.

```
struct include
{
        char *Type;
        char *Value;
        struct include *next;
```

```
};
typedef struct include Include;
```

Exclude is singly linked lists as there can be unbounded number of exclude elements inside what structure. This list has attribute type of string type and so they have been declared as char*.

```
struct exclude
{
        char *Type;
        char *Value;
        struct exclude *next;
};
typedef struct exclude Exclude;
```

What element is defined as a structure. This structure has unbounded number of include and exclude elements inside it and it does not have any attributes

```
struct what
{
        Include *inc;
        Exclude *exc;
};
typedef struct what What;
```

Changed is a singly linked list as there can be unbounded number of changed elements inside Trigger structure. This list has attributes from, to and by of string type and so they have been declared as char*.

```
struct changed
{
        char *From;
        char *To;
        char *By;
```

```
        char *Value;
        struct changed *next;
};
typedef struct changed Changed;
```

Added is a singly linked list as there can be unbounded number of added elements inside Trigger structure. This list does not have any attributes. It only has a content which is of string type and represented as char* in the list.

```
struct added
{
        char *Value;
        struct added *next;
};
typedef struct added Added;
```

Removed is a singly linked list as there can be unbounded number of removed elements inside Trigger structure. This list does not have any attributes. It only has a content which is of string type and represented as char* in the list

```
struct removed
{
        char *Value;
        struct removed *next;
};
typedef struct removed Removed;
```

Trigger is a singly linked list as there can be unbounded number of trigger elements inside Filter structure. This list does not have any attributes. It consists of unbounded number of changed, added and removed elements. It also has a content which is of string type and represented as char* in the list.

```
struct trigger
{
```

```
        Changed *chan;
        Added *add;
        Removed *rem;
        struct trigger *next;
};
typedef struct trigger Trigger;
```

Filter is a singly linked list as there can be unbounded number of trigger elements inside Filterset structure. This list has id, uri, domain, remove and enabled attributes which are of string type and so are represented as char* in the list. It also has what element and unbounded number of trigger elements

```
struct filter
{
        char *Id;
        char *Uri;
        char *Domain;
        char *Remove;
        char *Enabled;
        struct filter *next;
        What *wat;
        Trigger *trig;
};
typedef struct filter Filter;
```

Nsbinding is a singly linked list as there can be unbounded number of nsbinding elements inside Nsbindings structure. This list has prefix and urn attributes which are of string type and so are represented as char* in the list.

```
struct nsbinding
{
        char *Prefix;
        char *Urn;
        char *Value;
```

```
        struct nsbinding *next;
};
typedef struct nsbinding NsBinding;
```

Nsbinding is a structure and this is made up of an unbounded number of nsbinding elements.

```
struct nsbindings
{
        NsBinding *nsbind;
};
typedef struct nsbindings NsBindings;
```

FilterSet is the main structure and it has an unbounded number of nsbindings and filter sub elements. This structure has attributes xmlversion, encoding and package of string type and so they have been declared as char*.

```
typedef struct filterset
{
        char *Encoding;
        char *xmlversion;
        char *Package;
        NsBindings *nsbinds;
        Filter *filt;
} Filterset;
```

RFC 4662

Name is a singly linked list as there can be unbounded number of name elements inside instance or resource structure. This list does not have any attributes. It only has a content which is of string type and represented as char* in the list.

```
struct name
{
        char *Value;
        struct name *next;
};
typedef struct name Name;
```

Instance is a singly linked list as there can be unbounded number of instance elements inside Resource structure. This list has id, state, reason and cid attributes which are of string type and so are represented as char* in the list.

```
struct instance
{
        char *Id;
        char *State;
        char *Reason;
        char *Cid;
        struct instance *next;
};
typedef struct instance Instance;
```

Resource is a singly linked list as there can be unbounded number of instance elements inside Resource structure. This list has uri attribute which is of string type and so is represented as char* in the list. It has an unbounded number of name and instance elements.

```
struct resource
{
        char *Uri;
        struct resource *next;
        Name *nam;
        Instance *ins;
};
typedef struct resource Resource;
```

List is the main structure and it has an unbounded number of name and resource sub elements. This structure has attributes version, encoding, uri, cid and fullstate of string type and so they have been declared as char*.

```
typedef struct list
{
        char *XmlVersion;
        char *Encoding;
        char *Uri;
        char *Version;
        char *FullState;
        char *Cid;
        Name *nam;
        Resource *res;
} List;
```

4.5 Adding Main Module : Validate and Fill

The next step is to add a module to RXP for doing validation of content type (as RXP validates XML document against a DTD which does not support multiple data types) and to fill the data structures as mentioned in section 5.4.

In this phase, the bit structure obtained is taken for further processing. The get_tree function is used to get the bit structure from RXP parser .The bit structure is then traversed from root element to the leaf element since the structure will be arranged in the form of a tree. Each and every bit of the tree is read and every element is having a type like start tag, end tag or pcdata. Based on this information, the bit is passed to the handlers which handle each and every type of xml element. The handlers have different functionality. If the element encountered has a start type, then the element definition name should be inserted into the data structure. If they have the attributes, the proper function should be called which takes this element and fill up the data structures with the attribute name and value. If the element has a pcdata type, the handler should validate the content of the element and then fill the data structures. If

the element encountered has a pi type, the xml version and encoding must be validated and then filled into the proper data structures. During this traversal, each and every element information is obtained and the values like element name, attribute name, attribute value, content of that element are all inserted into the structures or the linked lists which are specified. During this process, another level of validation is done to overcome the limitations of DTD like checking the data type of the content of the element and attributes. This is a very important step because it is very necessary to check the content of the element matches the type which is specified in the schema or not. So the function Sdf_fn_ValidateFill is used to do second level validation of the XML file and then it is used to fill the structures and linked lists with the element names and values. The function Fillatt is used to get the attribute names and values ,and map into corresponding elements, and fill the structures and linked lists with those values.

4.6 Integration

To invoke APIs provided by RXP for validation and parsing, an application needs to integrate RXP into its application. To integrate RXP into the application the following needs to be done.

1. Include the RXP headers directory in the application include path.
2. Include the RXP library directory in the application library path.
3. Application needs to link librxp.a and libcatalog.a while building the application. If the requirement be there then the application should also link libsocket.a

After a successful build the application binary can be used for the testing.

CHAPTER 5
TESTING AND RESULTS

5.1 Testing RXP with Inputs of All four RFC's

The code is tested before integration against a number of test cases. These test cases include a number of valid and invalid XML files which have a reference to the external DTD. External DTD is referenced within the XML file. When this file is given as an input and the application is run, the parsing and validation of the XML file is done against a DTD. If there are no errors, data structures are filled and output is displayed. If any errors are found, the appropriate error message is displayed.

5.1.1 RFC 3858 [An Extensible Markup Language (XML) Based Format for Watcher Information]

The DTD "watcher.dtd" against which XML SIP message body of watcherinfo document is validated, which is given as an input to the RXP parser along with the input XML message body to be parsed is shown below:

<! ELEMENT watcherinfo (watcher-list*)>
<! ATTLIST watcherinfo version CDATA #REQUIRED>
<! ATTLIST watcherinfo state (full|partial) #REQUIRED>
<! ELEMENT watcher-list (watcher*)>
<! ATTLIST watcher-list resource CDATA #REQUIRED>
<! ATTLIST watcher-list package CDATA #REQUIRED>
<! ELEMENT watcher (#PCDATA)>
<! ATTLIST watcher id CDATA #REQUIRED>
<! ATTLIST watcher status (pending|active|waiting|terminated) #REQUIRED>
<! ATTLIST watcher event (subscribe|approved|decativated|probation|rejected |timeout|giveup|noresource) #REQUIRED>
<! ATTLIST watcher display-name CDATA #IMPLIED>
<! ATTLIST watcher expiration CDATA #IMPLIED>
<! ATTLIST watcher duration-subscribed CDATA #IMPLIED>

Valid Input

A sample valid watcher info message watcherinfo.xml given as input to the RXP parser by the SIP entity is given below:

```
<? xml version="1.0" encoding="UTF-8"?>
<! DOCTYPE watcherinfo SYSTEM "watcher.dtd">
<watcherinfo state="full" version="1">
<watcher-list resource="sip:professor@example.net" package="presence">
<watcher status="active" id="8ajksjda7s" duration-subscribed="509"
event="approved"> sip:userA@example.net </watcher>
<watcher status="active" id="948442dsd" event="subscribe"></watcher>
<watcher status="active" id="dju90382" event="subscribe"></watcher>
</watcher-list>
<watcher-list resource="sip:student@example.net" package="presence">
<watcher status="active" id="abcd123" event="subscribe"></watcher>
<watcher status="active" id="xyz321" event="subscribe"></watcher>
</watcher-list>
</watcherinfo>
```

As the input watcherinfo.xml is a valid xml message so RXP parses it and fill the data structures containing members like watcherlist, displayname, status, watcher etc to be used by the application. A screenshot of output obtained from RXP parser after parsing and filling structures by giving command rxp watcherinfo.xml is shown below:

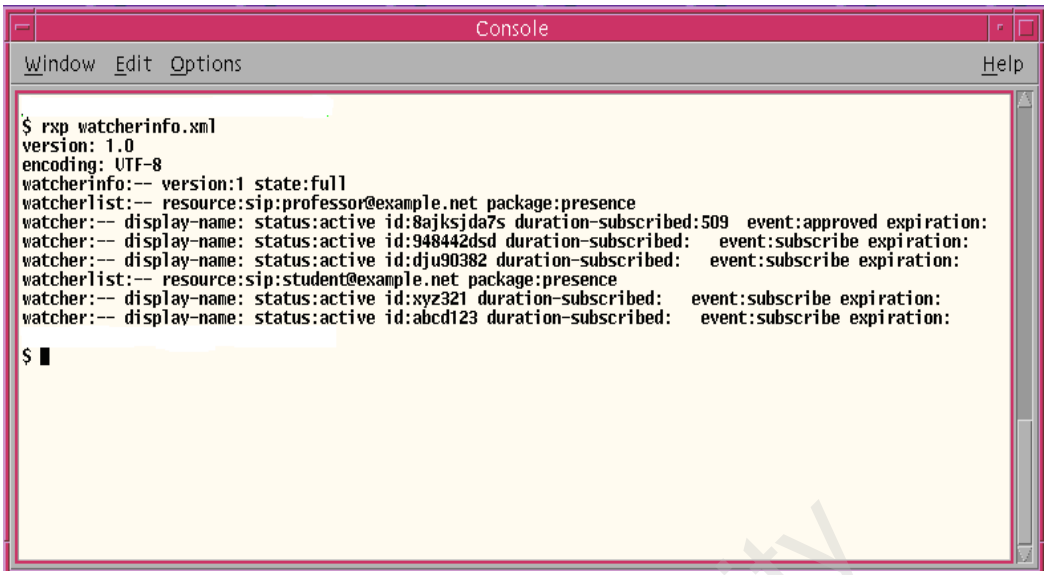

Figure 5.1 Output of RFC 3858 with Valid Input

Invalid Input

A sample invalid watcher message watcherinfo1.xml given as input to the RXP parser by the SIP entity is shown below:

<? xml version="1.0" encoding="UTF-8"?>
<! DOCTYPE watcherinfo SYSTEM "watcher.dtd">
<watcherinfo state="xyz" version="1">
<watcher-list resource="sip:professor@example.net" package="presence">
<watcher status="active" id="8ajksjda7s" duration-subscribed="509"
event="approved"> sip:userA@example.net </watcher>
<watcher status="active" id="948442dsd" event="subscribe"></watcher>
<watcher status="active" id="dju90382" event="subscribe"></watcher>
</watcher-list>
</watcherinfo>

Though the input watcherinfo1.xml is well-formed (syntactically correct) but the above input contains the attribute “state” with value “xyz” which is not as per specifications provided by the DTD. As per DTD “watcher.dtd” state attribute can have only “full” or “partial” as its values. So RXP being a validating parser which in addition to check for well-formedness also checks for validity and thus gives an error

pointing out “xyz” is not one of the allowed values for state. A screenshot of output showing an error obtained from RXP parser by giving command rxp watcherinfo1.xml is shown below:

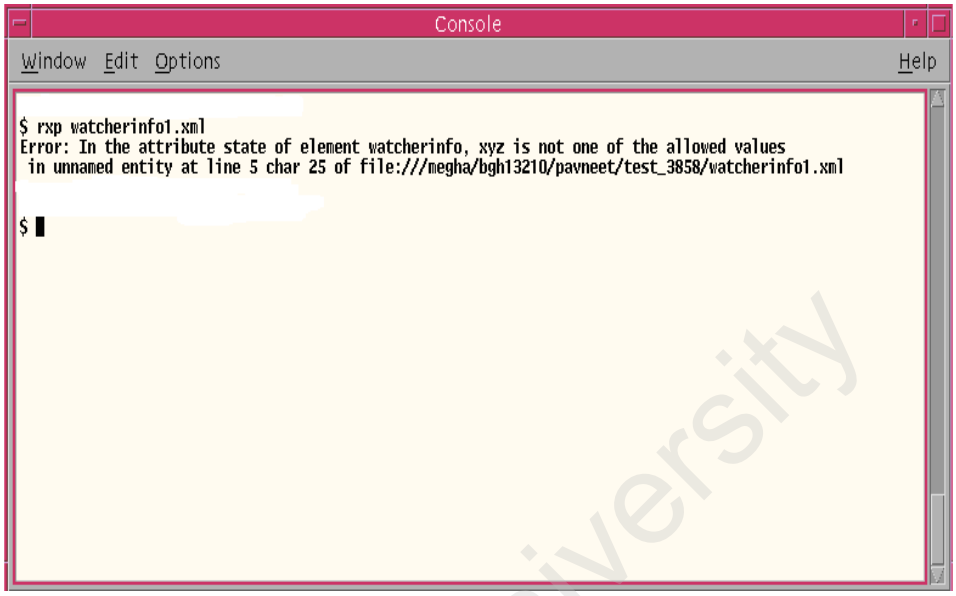

Figure 5.2 Output of RFC 3858 with Invalid Input

5.1.2 RFC 3994 [Indication of Message Composition for Instant Messaging]

The DTD “compose.dtd” against which XML SIP message body of is composing document is validated , which is given as an input to the RXP parser along with the input XML message body to be parsed is given below:

<! ELEMENT isComposing (state, lastactive? ,contenttype?, refresh?)>
<! ELEMENT state (#PCDATA) >
<! ELEMENT lastactive (#PCDATA)>
<! ELEMENT contenttype (#PCDATA)>
<! ELEMENT refresh (#PCDATA)>

Valid Input

A sample valid is-composing message message.xml given as input to the RXP parser by the SIP entity is given below:

<? xml version="1.0" encoding="UTF-8"?>

<! DOCTYPE isComposing SYSTEM "compose.dtd">

<isComposing>

<state>idle</state>

<lastactive>2003-01-27T10:43:00Z</lastactive>

<contenttype>audio</contenttype>

<refresh>90</refresh>

</isComposing

As the input message.xml is a valid xml message so RXP parses it and fill the data structures containing members like version, encoding, state etc to be used by the application. A screenshot of output obtained from RXP parser after parsing and filling structures by giving command rxp message.xml shown below:

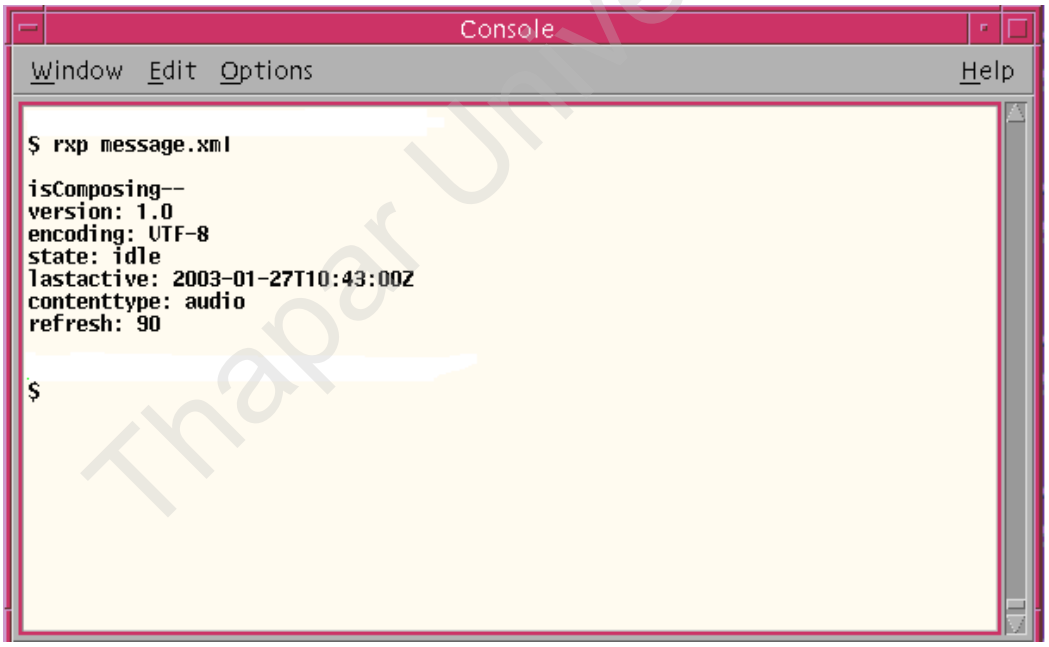

Figure 5.3 Output of RFC 3994 with Valid Input

Invalid Input

A sample invalid is-composing message message1.xml given as input to the RXP parser by the SIP entity is shown below:

<? xml version="1.0" encoding="UTF-8"?>

<! DOCTYPE isComposing SYSTEM "compose.dtd">

<isComposing>
<state>idle
<lastactive>2003-01-27T10:43:00Z</lastactive>
</state>
<contenttype>audio</contenttype>
<refresh>90</refresh>
</isComposing>

Though the input message1.xml is well-formed (syntactically correct) but the above input contains the element “lastactive” inside “state” which is not as per specifications provided by the DTD .As per DTD “compose.dtd” element “lastactive” is child of “isComposing” instead of “state”. So RXP being a validating parser checks for validity and thus gives an error pointing out that “lastactive” is not allowed at this position. A screenshot of output showing an error obtained from RXP parser by giving command message1.xml is shown below:

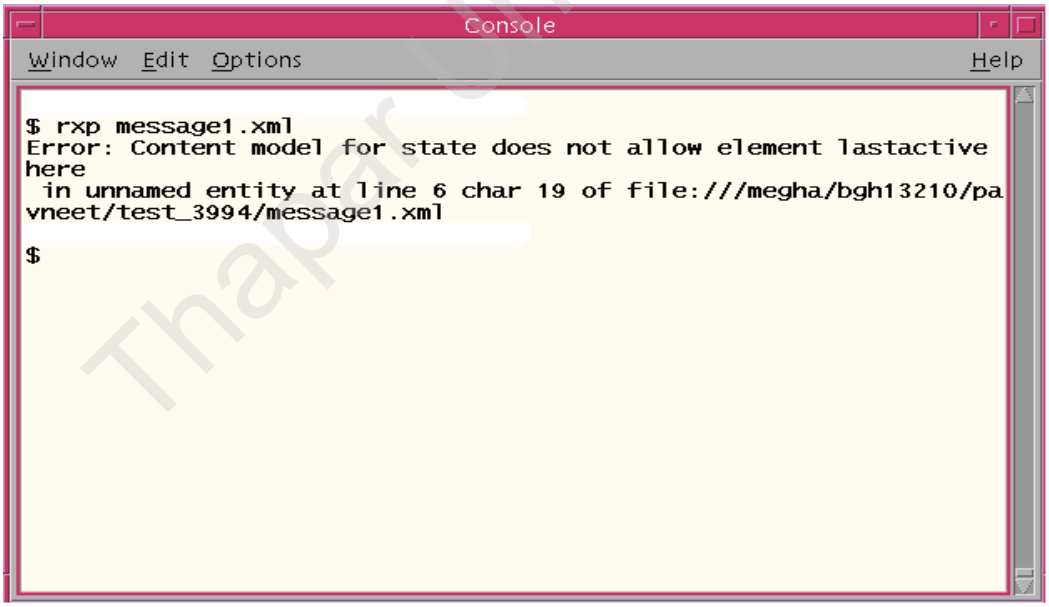

Figure 5.4 Output of RFC 3994 with Invalid Input

5.1.3 RFC 4661 [An Extensible Markup Language (XML) Based Format for Event Notification Filtering]

The DTD “eventnot.dtd” against which XML SIP message body of event notification

document is validated, which is given as an input to the RXP parser along with the input XML message body to be parsed is given below:

<! ELEMENT filter-set (ns-bindings?, filter+)>
<! ATTLIST filter-set package CDATA #IMPLIED>
<! ELEMENT ns-bindings (ns-binding+)>
<! ELEMENT ns-binding (#PCDATA)>
<! ATTLIST ns-binding prefix CDATA #REQUIRED>
<! ATTLIST ns-binding urn CDATA #REQUIRED>
<! ELEMENT filter (what?, trigger*)>
<! ATTLIST filter id CDATA #REQUIRED>
<! ATTLIST filter uri CDATA #IMPLIED>
<! ATTLIST filter domain CDATA #IMPLIED>
<! ATTLIST filter remove (true|false) "false">
<! ATTLIST filter enabled (true|false) "true">
<! ELEMENT what (include*, exclude*)>
<! ELEMENT include (#PCDATA)>
<! ELEMENT exclude (#PCDATA)>
<! ATTLIST include type (xpath|namespace) "xpath">
<! ATTLIST exclude type (xpath|namespace) "xpath">
<! ELEMENT trigger (changed*, added*, removed*)>
<! ELEMENT changed (#PCDATA)>
<! ELEMENT added (#PCDATA)>
<! ELEMENT removed (#PCDATA)>
<! ATTLIST changed from CDATA #IMPLIED>
<! ATTLIST changed by CDATA #IMPLIED>
<! ATTLIST changed to CDATA #IMPLIED

Valid Input

A sample valid event notification message test1_4661.xml given as input to the RXP parser by the SIP entity is given below:

<? xml version="1.0" encoding="UTF-8"?>

```
<! DOCTYPE filter-set SYSTEM "eventnot.dtd">
<filter-set package="hellloo">
<ns-bindings>
<ns-binding prefix="pidf" urn="urn:ietf:params:xml:ns:game-ext">abbb
</ns-binding>
<ns-binding prefix="rpid" urn="urn:ietf:params:xml:ns:game-ext">bbccc
</ns-binding>
</ns-bindings>
<filter id="8439" uri="sip:buddies@example.com" domain="abc" remove="true"
enabled="false"> <what>
<include type="xpath">firstinclude</include>
<include type="xpath">secondinclude</include>
<exclude type="namespace">firstexclude</exclude>
<exclude type="namespace">secondexclude</exclude></what>
<trigger>
<changed to="pending" from="CLOSED" by="ccc">firstchang</changed>
<changed to="waiting" from="CLOSED" by="ccc1">secondchang</changed>
<added>firstaddd</added>
<added>secondaa</added>
<removed>firstremoov</removed>
<removed>secondremmov</removed>
</trigger></filter>
<filter  id="123" uri="sip:bob@example.com" remove="true" enabled="false">
<what><include type="xpath">first1inc</include>
<include type="xpath">sec1inc</include>
<exclude type="namespace">first1exclude</exclude>
<exclude type="namespace">second1exclude</exclude></what>
<trigger>
<changed to="open" from="closed" by="ccc">firstchang111</changed>
<changed to="pending" from="closed" by="ccc1">secondchang1111</changed>
<added>firstaddd1111</added>
<added>secondaa1111</added></trigger></filter>
</filter-set>
```

As the input test1_4661.xml is a valid xml message so RXP parses it and fill the data structures containing members like nsbind, filter, id, tigger etc to be used by the application. A screenshot of output obtained from RXP parser after parsing and filling structures by giving command rxp test1_4661.xml is shown below:

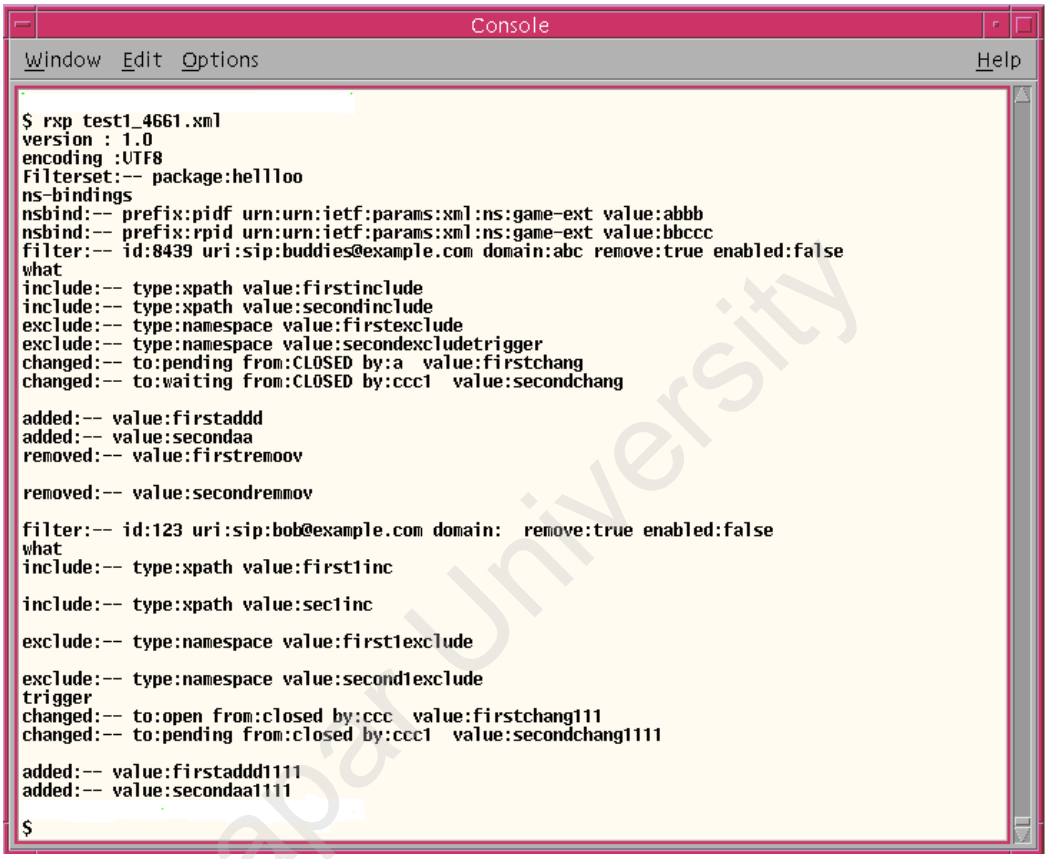

Figure 5.5 Output of RFC 4661 with Valid Input

Invalid Input

A sample invalid event notification message test2_4661.xml given as input to the RXP parser by the SIP entity is shown below:

<? xml version="1.0" encoding="UTF-8"?>

<! DOCTYPE filter-set SYSTEM "eventnot.dtd">

<filter-set package="hellloo">

<ns-bindings>

<ns-binding urn="urn:ietf:params:xml:ns:game-ext">abbb </ns-binding>

<ns-binding prefix="rpid" urn="urn:ietf:params:xml:ns:game-ext">bbccc

```
</ns-binding>
</ns-bindings>
<filter id="8439" uri="sip:buddies@example.com" domain="abc" remove="true"
enabled="false"> <what>
<include type="xpath">firstinclude</include>
<include type="xpath">secondinclude</include>
<exclude type="namespace">firstexclude</exclude>
<exclude type="namespace">secondexclude</exclude></what>
</filter></filter-set>
```

Though the input test2_4661.xml is well-formed (syntactically correct) but in the above input attribute “prefix” for element “nsbinding” is missing which as per specifications provided by the DTD “eventnot.dtd” is a REQUIRED attribute. So RXP gives an error pointing out that required attribute “prefix” for element “nsbinding” is missing. A screenshot of output showing an error obtained from RXP parser by giving command rxp test2_4661.xml is shown below:

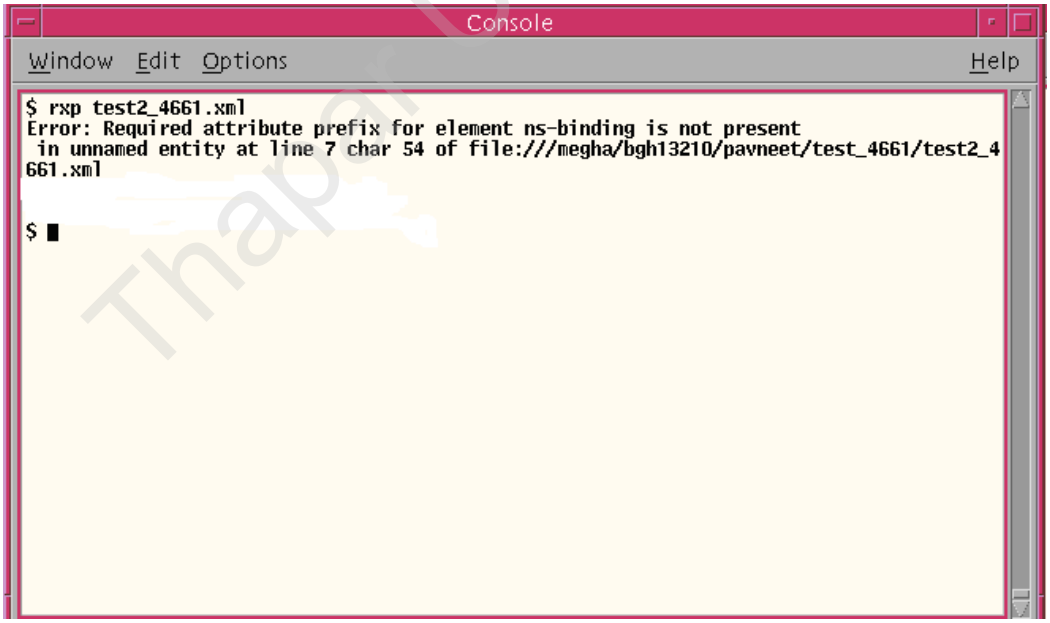

Figure 5.6 Output of RFC 4661 with Invalid Input

5.1.4 RFC 4662 [SIP Event notification extension for resource lists]

The DTD "rfc_4662.dtd" against which XML SIP message body of RLMI document is validated, which is given as an input to the RXP parser along with the input XML message body to be parsed is given below:

<! ELEMENT list (name*, resource*)>
<! ATTLIST list uri CDATA #REQUIRED>
<! ATTLIST list version CDATA #REQUIRED>
<! ATTLIST list fullState (true|false) #REQUIRED>
<! ATTLIST list cid CDATA #IMPLIED>
<! ELEMENT resource (name*, instance*)>
<! ELEMENT name (#PCDATA)>
<! ELEMENT instance (#PCDATA)>
<! ATTLIST resource uri CDATA #REQUIRED>
<! ATTLIST instance id CDATA #REQUIRED>
<! ATTLIST instance state (active|pending|terminated) #REQUIRED>
<! ATTLIST instance reason CDATA #IMPLIED>
<! ATTLIST instance cid CDATA #IMPLIED>

Valid Input

A sample valid event notification message test1_4662.xml given as input to the RXP parser by the SIP entity is given below:

9
<? xml version="1.0" encoding="UTF-8"?>
<! DOCTYPE list SYSTEM "rfc_4662.dtd">
<list uri="sip:adam-friends@lists.vancouver.example.com"
version="7"fullState="true">
<name>Buddy List</name>
<name>Liste d'amis</name>
<resource uri="sip:bob@vancouver.example.com">
<name>Bob Smith</name><name>Ed</name>
<instance id="juwigmtboe" state="active"
cid="12345.aaa@vancouver.example.com"/>

```
</resource>
<resource uri="sip:dave@vancouver.example.com">
<name>Dave Jones</name>
<instance id="hqzsuxtfyq" state="active"
cid="12345.aab@vancouver.example.com"/> </resource>
<resource uri="sip:jim@vancouver.example.com">
<name>Jim</name>
<instance id="oflzxqzuvg" state="terminated" reason="rejected" />
<instance id="ofzxqzuvg" state="terminated" reason="rejected" /></resource>
<resource uri="sip:ed@vancouver.example.com">
<name>Ed</name>
<instance id="grqhzsppxb" state="pending"/></resource>
</list>
```

As the input test1_4662.xml is a valid xml message so RXP parses it and fill the data structures containing members like name, resource, instance etc to be used by the application. A screenshot of output obtained from RXP parser after parsing and filling structures by giving command rxp test1_4662.xml is shown below:

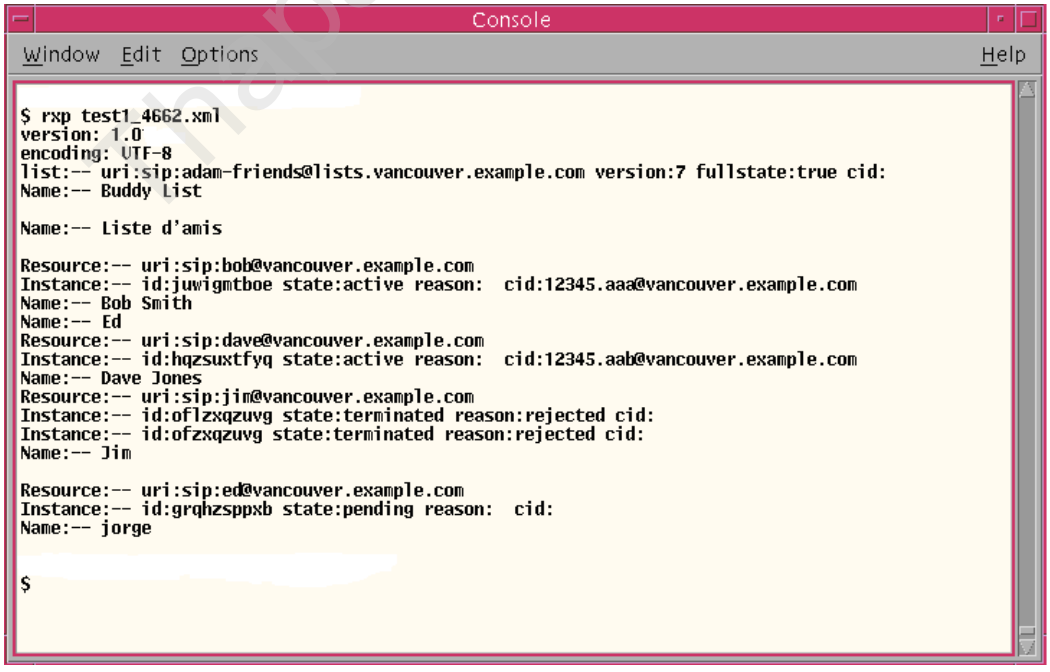

Figure 5.7 Output of RFC 4662 with Valid Input

Invalid Input

A sample invalid event notification message test2_4662.xml given as input to the RXP parser by the SIP entity is shown below:

```
<? xml version="1.0" encoding="UTF-8"?>
<! DOCTYPE list SYSTEM "rfc_4662.dtd">
<list uri="sip:adam-friends@lists.vancouver.example.com"
version="7"fullState="true">
<name>Buddy List</name>
<name>Liste d'amis</name>
<resource uri="sip:bob@vancouver.example.com">
<name>Bob Smith</name><name>Ed</name>
<instance id="juwigmtboe" state="active"
cid="12345.aaa@vancouver.example.com"/>
</resource>
</list>
```

Though the input test2_4662.xml is well-formed (syntactically correct) but in the above input attribute “version” for element “list” is missing which as per specifications provided by the DTD “rfc_4662.dtd” is a REQUIRED attribute.So RXP being a validating parser gives an error pointing out that required attribute “version” for element “list” is not present. A screenshot of output showing an error obtained from RXP parser by giving command rxp test2_4662.xml is shown below:

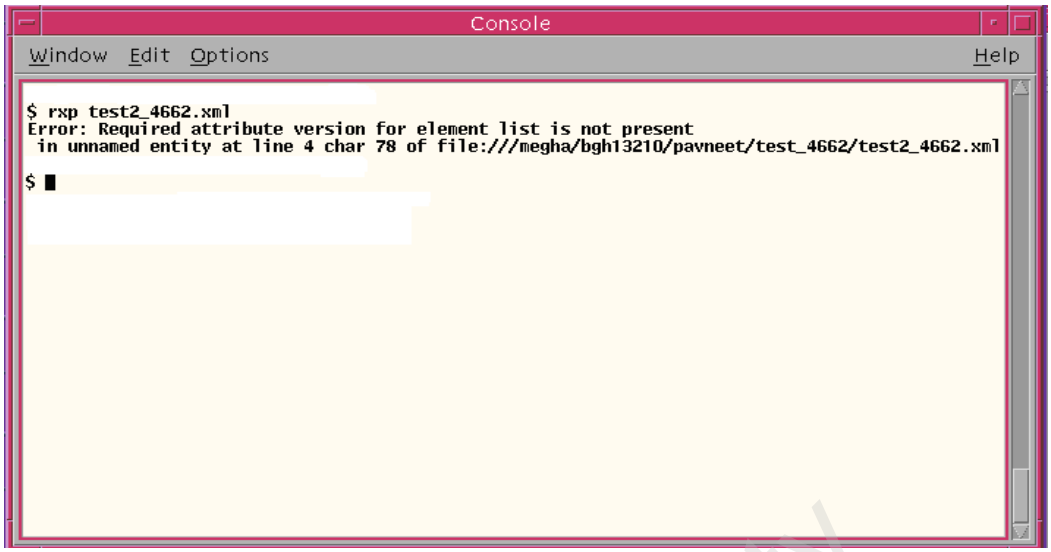

Figure 5.8 Output of RFC 4662 with Invalid Input

5.2 Performance Analysis of RXP

The performance of RXP parser is to be determined exactly to identify whether RXP is efficient and less time consuming. So, to analyze the performance of the parser a graph has been plotted which shows the time utilized by the parser for execution of different sized XML files. If the time taken for parsing an XML file varies linearly as the size of the parser increases, it suggests that the parsing is linear in nature and also it indicates that the parser is efficient. This discussion is demonstrated by the example below:

Firstly a table which gives the number of lines that XML file has and the time taken for parsing all these files is taken using clock () function. clock() function returns the number of clock ticks elapsed since the program was launched. Two variables t1 and t2 are declared of type clock_t. clock_t is a type defined in <ctime> to some type capable of representing clock tick counts and support arithmetical operations (generally a long integer). Time t1 is calculated by calling clock() function before calling ReadXTree() function of RXP Parser and another time t2 is calculated after ReadXtree() function. Then time is calculated then as, time= (t2-t1)/CLOCKS_PER_SEC. The macro constant expression CLOCKS_PER_SEC

specifies the relation between a clock tick and a second (clock ticks per second). Dividing a count of clock ticks by this expression yields the number of seconds.

Lines in XML	Time taken (Seconds)
0	0
100	0
1000	0.02
10000	0.20
25000	0.61
50000	1.25
75000	1.64
100000	2.40

Table 5.1 Table showing number of XML lines and time taken to parse

Then, based on the above mentioned details, a line graph has been plotted. This graph has the number of lines as X-axis and the time taken for parsing as Y-axis. The graph is plotted by considering main points and this results in almost a linear graph. So, this suggests that the time taken by the RXP Parser to parse the XML files varies almost linearly to the size of the XML file. The linearity is one of the key factors to determine whether the parser is efficient or not. The plotted graph is shown in the diagram below:

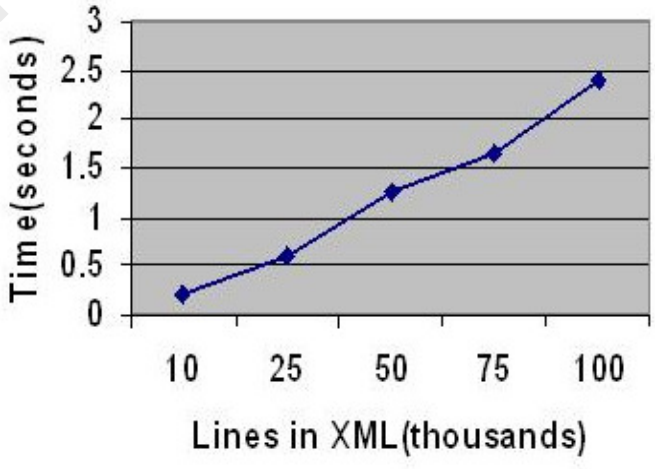

Figure 5.9 Graph showing the Time taken vs Size of XML

CHAPTER 6
CONCLUSION AND FUTURE SCOPE

6.1 Conclusion

This work concludes that a validating parser removes the overhead of doing the XML validation against any schema or DTD, hence reducing the overall code size. Also effort needed for the inclusion of any new parsing functionality will be significantly less as compared to using a non-validating parser which requires the validation to be done separately by writing our own source code, thereby increasing the code size and extending greatly the time taken to implement any new feature enhancement. By using a validating parser, these validations will be done by the XML parser itself and hence the SIP entity (which is the system under development) need not perform the validations by itself. This will ensure that support for many XML data formats can be included quickly. Thus, validating XML parsers help business by aiding the development of useful features within a shorter time frame.

By making use of a validating parser it is just needed to maintain structures for storing the parsed XML data. The incoming XML message with its schema or DTD can be passed as input to the parser which validates the input message against the input schema / DTD and provides as output parsed data in the form of a tree. This output can then be used to fill the data structures maintained and given to the application that is responsible for feature implementation.

6.2 Future Scope of Work

Technology is changing the face of world. VoIP is one such technology. The future exentions in this technology heavily depend of parsers. Validating parsers, like RXP, thus have a major role to play in the coming years. RXP further needs to be enhanced to meet such future requirements. As RXP Parser is restricted to do validation of an XML message body against a DTD and not against schema so in future its functionality can be enhanced by including support for validation against schema which will eliminate the need of doing extra validation of content type and support for namespaces. Support for encryption and digital signature is also a desired feature. It

can be enhanced to have Parse-on-demand feature. The Parse-on-demand technology allows the handling of very large (100's of MBs) XML files that are memory-mapped and parsed on demand (i.e. as you navigate through the DOM, the document is parsed as needed). It can further be enhanced to have Parse from Compressed feature. This new concept of Parse from compressed XML hopes to use parse-on-demand technology on top of a decompress-on-demand technology. As RXP Parser gives efficient results and reduces the code size and the time taken to implement any new features. So support for schemas defined in some more RFC's can be easily included. These enhancements will help develop more robust and efficient VoIP application. Needless to say the future for XML parsers for SIP message bodies is bright and very challenging.

REFERENCES

[1] Jan Eqil Refsnes "Extensible markup language"
http://www.xmlfiles.com/xml/xml_usedfor.asp

[2] Adobe Developer Center- "XML Overview"
http://www.adobe.com/devnet/dreamweaver/articles/xml_overview_03.html

[3] Francisco J. Mora and Raimo Kantola **"**HELSINKI UNIVERSITY OF TECHNOLOGY**,** Laboratory of Telecommunications Technology" Espoo, December 1999
www.cs.columbia.edu/**sip**/articles/Espi9912_**Implementation**.pdf

[4] From Wikipedia, the free encyclopedia "Voice over Internet Protocol"
http://en.wikipedia.org/wiki/VoIP

[5]Federal Communications Commission "VOIP"
http://www.fcc.gov/voip/

[6] Robert Valdes and Dave Roos "How VoIP Works"
http://communication.howstuffworks.com/ip-telephony.html

[7] SIP Center "SIP Basics"
http://www.sipcenter.com/sip.nsf/html/What+Is+SIP+Introduction

[8] Network Working Group J. Rosenberg Request for Comments: 3261 Category: Standards Track, June 2002 "SIP"
http://www.ietf.org/rfc/rfc3261.txt?number=3261

[9] Wikipedia "Session Initiation Protocol**"**
http://en.wikipedia.org/wiki/Session_Initiation_Protocol

[10] Artech House Publishers by Alan B. Johnston "Sip: Understanding the SIP"
www.ebookee.com/Sip-Understanding-the-Session-Initiation-Protocol

[11] "SIP Features " http://www.voip-info.org/wiki-SIP

[12] Wikipedia "XML Definition"
http://en.wikipedia.org/wiki/XML

[13] "XML Definition" www.rustybrick.com/definitions.php

[14] "XML Basics"http://www.softwareag.com/xml/about/starters.htm

[15] "Correctness of XML Documents"http://www.experiencefestival.com/a/XML_-_Correctness_in_an_XML_document/id/1964503

[16]Advantages of XML
www.cs.columbia.edu/sip/articles/Espi9912_Implementation.pdf

[17] Jennifer Kyrnin "Document Type Definition "
http://webdesign.about.com/od/dtds/a/aa101700a.htm

[18] Jan Eqil Refsnes "Introduction to DTD"
http://www.xmlfiles.com/dtd/dtd_intro.asp

[19] Wikipedia "DTD"
http://en.wikipedia.org/wiki/Document_Type_Definition

[20]"XML Schema Definition"
http://searchsoa.techtarget.com/sDefinition/0,,sid26_gci831325,00.html

[21] "Why use XML Schemas" http://www.w3schools.com/Schema/schema_why.asp

[22] "Parsing" Edited by Peter Flynn Version 4.56 (8 August 2007)
http://xml.silmaril.ie/authors/parsers/

[23] "What is Parser" by NorKen Technologies
http://www.programmar.com/parser.htm

[24] Webopedia "Parsing"http://www.webopedia.com/TERM/p/parse.html

[25] "Parser in computer technology"
http://searchsoa.techtarget.com/sDefinition/0,,sid26_gci212749,00.html

[26]"XML Parser Definition"
http://www.pcmag.com/encyclopedia_term/0,2542,t=XML+parser&i=55052,00.asp

[27] XML Glossary, December 1999 "XML Parser "
http://www.microsoft.com/presspass/features/2000/01-03xmlglossary.mspx

[28] XML Software Guide: XML Parsers Ken Sall July 5th 1998, Last Modified: July 29, 2000
http://www.wdvl.com/Software/XML/parsers.html

[29] "XML Parsers" by Jani Utriainen 41836V
http://mia.ece.uic.edu/~papers/WWW/MultimediaStandards/Parsers.pdf

[30] "Event Based Parsing" by Kenneth B.Sall ,May 31 ,2002
http://www.informit.com/articles/article.aspx?p=27006&seqNum=7

[31] http://www.saxproject.org/event.html

[32] SYBASE, June 2002 "SAX & DOM parsing"
http://www.theserverside.com/tt/articles/article.tss?l=JAXP

[33] XML and Perl By Mark Riehl, Ilya Sterin "XML Processing"
http://books.google.co.in/books?id=dn-37TE8bAoC&dq=isbn:0735712891

[34]"Expat tutorial"http://www.vivtek.com/expat.html

[35]Wikipedia "Expat" http://en.wikipedia.org/wiki/Expat_(XML)

[36] XML: LibXML by Kip Hampton, November 14, 2001
http://www.xml.com/pub/a/2001/11/14/xml-libxml.html

[37]"RXP" version: 43312(open source 09/10/07)
http://www.linuxcertif.com/man/1/rxp

[38]J. Rosenberg, August 2004 "RFC 3858"
http://www.ietf.org/rfc/rfc3858.txt

[39] H. Schulzrinne and Columbia U.,January 2005
http://www.ietf.org/rfc/rfc3994.txt

[40] H. Khartabil, Telio, E. Leppanen, M. LonnforsJ, Costa-Requena September 2006
http://www.ietf.org/rfc/rfc4661.txt

[41 A. B. Roach, Estacado Systems Cisco Systems, 2006
http://www.ietf.org/rfc/rfc4662.txt