

SOCIO-CULTURAL COMMUNICATION SYSTEM – A COMMUNICATION
MECHANISM FOR MULTI-MEDIA INFORMATION ACCESS SYSTEM FOR NON-
LITERATE AND LINGUISTICALLY DIVERSE USERS

A THESIS IN
Computer Science

Presented to the Faculty of the University
of Missouri-Kansas City in partial fulfillment of
the requirements for the degree

MASTER OF SCIENCE

by

VENKATA RAMA KRISHNA JAMITHIREDDY

Bachelor of Technology, Andhra University, 2005

Kansas City, Missouri

2010

© 2010

VENKATA RAMA KRISHNA JAMITHIREDDY

ALL RIGHTS RESERVED

SOCIO-CULTURAL COMMUNICATION SYSTEM – A COMMUNICATION MECHANISM
FOR MULTI-MEDIA INFORMATION ACCESS SYSTEM FOR NON-LITERATE AND
LINGUISTICALLY DIVERSE USERS

Venkata Rama Krishna Jamithireddy, Candidate for the Master of Science Degree

University of Missouri-Kansas City, 2010

ABSTRACT

Current Internet services are not optimal for the access of information for non-literate or linguistically diverse users. Almost all Internet content is available only in written form and still in a limited number of languages. However, there is a growing need to provide network services to non-literate or linguistically diverse users. The Socio-cultural communication system (SoCCS) is a communication mechanism for building an information access system that helps both literate and non-literate users to access information in various media formats. SoCCS is designed using client server architecture, isolating much of the logic from the client. This helps the client run on a basic system of a very low end configuration with an application based on a web-browser. The client will send the text/image/audio/video request to the request interpreter (RI) and it will then process the request and respond with text, image, audio, or video link information. Audio/video is streamed using a streaming server. This helps in reducing the play delay on the client side with a low bandwidth. RI will handle the request in text/image/audio using text-to-text, text-to-speech and speech-to-text translators. RI will also handle the client's preference on the response i.e., the client can choose to get the response in a text/image/audio/video format.

The SoCCS application protocol of the current design project is based on a request-response mechanism without any limitations on the natural language, client software and hardware requirements, and the type of request. SoCCS will accept a text or audio request and

respond to the user with a text or an audio message. The unique SoCCS features are providing the conversion from one language to another language, from one format to another format and consider user preferences for displaying search results. In this thesis, a prototype implementation of SoCCS is also discussed.

The undersigned, appointed by the Dean of School of Graduate Studies, have examined the thesis titled “Socio-Cultural Communication System – A Communication Mechanism for Multi-Media Information Access System for Non-literate and Linguistically Diverse Users,” presented by Venkata Rama Krishna Jamithireddy, a candidate for the Master of Science degree, and hereby certify that in their opinion it is worthy of acceptance.

Supervisory Committee

Deepankar Medhi, Ph.D., Committee Chairperson

Department of Computer Science and Electrical Engineering

Kenneth Mitchell, Ph.D.

Department of Computer Science and Electrical Engineering

Yugyung Lee, Ph.D.

Department of Computer Sciences and Electrical Engineering

CONTENTS

ABSTRACT.....	iii
LIST OF ILLUSTRATIONS.....	x
ACKNOWLEDGEMENTS.....	ix
Chapter	
1 BACKGROUND AND SIGNIFICANCE.....	1
1.1. Socio-Cultural Communication System.....	1
2 OVERVIEW.....	4
2.1. Requirements for SoCCS Design.....	4
2.1.1. Inclusive.....	4
2.1.2. Dialectic.....	5
2.1.3. Adaptive.....	5
2.1.4. Evolving.....	5
2.1.5. People Sensitive.....	5
2.3. SoCCS and e-Governance.....	5
2.4. SoCCS and e-Systems.....	6
2.4.1. Population Factor.....	7
2.4.2. Mode of Access to Information.....	7
2.4.3. Priorities and Types of Information.....	7
2.4.4. Content Type.....	7
3 ARCHITECTURAL AND DESIGN APPROACH.....	8

3.1. Architecture	8
3.1.1. User Agent (UA)	9
3.1.2. Request Interpreter	10
3.1.3. Translator.....	12
3.2. Possible Cases for Data Flow between Different Modules.....	13
3.2.1. Case1: Translator Responding for the User Request.....	13
3.2.2. Case2: Request Interpreter Handles the Translation Data.....	15
3.2.3. Case3: Request Interpreter Forwards the Data Handle to Translator	17
3.3. Design Algorithm.....	19
3.4. Communication between Modules.....	22
3.4.1. User Agent – Request Interpreter Communication	22
3.4.2. User Agent – Media Base Communication	24
3.4.3. Request Interpreter – Translator Communication	24
4 SEQUENCE DIAGRAMS FOR REQUEST HANDLING	26
4.1. No Translation for Request and Response	26
4.2. Request Translation.....	27
4.3. Response Translation	29
4.4. Request and Response Translation.....	30
4.5. Streaming Sequence Diagram	31
5 IMPLEMENTATION OF SoCCS FRAMEWORK.....	33
5.1. User Agent (UA)	34

5.1.1. Web browser	34
5.1.2. Java Sonics ListenUp Applet.....	34
5.1.3. Google Virtual Keyboard	35
5.2. Request Interpreter (RI)	35
5.2.1. Web Server	36
5.3. Translator (TR).....	36
5.3.1. Google Translate.....	36
5.3.2. eSpeak.....	37
5.3.3. Sphinx.....	37
5.3.4. SoX.....	37
5.4. Media Base (MB).....	38
5.4.1. Streaming Server	38
5.4.1.1. Darwin Streaming Server.....	38
5.4.1.2. VideoLAN Server	39
5.4.1.3. Live Stream.....	39
5.4.1.4. YouTube	39
5.4.1.5. Unreal Media Streaming Server.....	39
6 EVALUATION.....	41
7 CONCLUSION.....	47
APPENDICES	49
REFERENCE LIST	53

VITA..... 56

LIST OF ILLUSTRATIONS

Figure	Page
1. SoCCS Block Diagram.....	9
2. Translator responds to User Request.....	14
3. Request Interpreter handles the Translation data.....	16
4. Request Interpreter forwards data handles to Translator.....	18
5. SoCCS Design Work Flow Diagram.....	21
6. No Request and Response Translation.....	27
7. Request Translation.....	28
8. Response Translation.....	30
9. Request and Response Translation.....	31
10. Streaming Server.....	32

ACKNOWLEDGEMENTS

I am indebted to Dr. Deep Medhi for his most generous assistance with this research project. I highly appreciate his time and effort, especially for spending significant amount of time while performing the research and documentation. I express my sincere thanks to Dr. William J Dolla, Dr. Yugyung Lee, Prashant Sunkari, Chittibabu Ruddaraju for their assistance, time and effort. I would also like to thank Pratima Jamithireddy, my wife, for her constant inspiration and persistence. I would also like to acknowledge the support and guidance of my family and friends. Finally, I would also like to thank the University of Missouri-Kansas City for providing me with the opportunity and the resources to accomplish my master's research.

Venkata Jamithireddy

CHAPTER 1

BACKGROUND AND SIGNIFICANCE

1.1. Socio-Cultural Communication System

Most of the current information access systems are targeted toward literate users that understand at least one natural language such as English. However, a billion of world's population is non-literate, and hence, left out of the information society due to the lack of any command over natural languages. This culturally diverse and information starving community may be harnessed by developing an information access system that does not require the command of a structured language. Such an information access system targeted toward the information poor, including non-literate and linguistically diverse community, is referred to as the Socio-Cultural Communication System (SoCCS). In developed countries, the non-literate and linguistically diverse peoples have less exposure to the online services [8]. They will have less access to information on the web about employment, health, and governance processes. Design and implementation of SoCCS prototype framework that provides interactive reading aids to these potential users is discussed in this project. These SoCCS information systems provide people with access to a variety of resources, including academic, e-commerce, health and wellness, weather, civic, economic, and agricultural information. They also help the non-literate users to be proactive in taking preventive measures rather than reactive or curative steps during the times of epidemic break down.

In developed countries, the information access infrastructure for mass communication, including news media and public announcement, has already been implemented with a certain degree of success. This is due to the availability of financial resources and a relative homogeneity of the population. However, development of such mass communication systems in

highly rudimentary in developing countries due to the limited availability of financial resources, significant linguistic diversity, and low literacy rate. Therefore, the SoCCS information system of the current research project is targeted toward geographic regions of high socio-economic inequality, cultural diversity, and low literacy. Such diversity, prevalent in developing countries such as India, presents unique challenges for the development of SoCCS information systems.

Chowdhury and Medhi [1] describes eSystem as “Demographic, economic, and linguistic diversity, coupled with the lack of coordinated efforts, give rise to a situation where the population has varying degrees of understanding of health-related issues due to varying literacy levels and the associated information access facilities. This may give rise to a situation where the access and use of any Information and Communication Technologies (ICT) enablement concerning public health gets restricted only to the literate and privileged sections of the society. Therefore, our interest is in working towards building a SoCCS application that will benefit the underprivileged sections of the society”.

One of the significant efforts toward the development of an information retrieval system targeted toward the common man is the Simputer [28], which is a proprietary, low cost, and browser enabled hand-held PC. The Simputer provides information access to both literate and non-literate users through a browser using Information Markup Language (IML). IML is the variant of XML and supports in development of text free interface. However, this proprietary system limits the user experience due to its limitations in software/hardware implementations and upgrades. In addition, the Simputer lacks the ability to evolve on the highly-variable user requirements and system upgrades.

Google voice and text search for local languages is one of the major advancement in taking demographic or natural language into consideration for search criteria. These Google

applications are considering the search string a literal and they search for that string. Google applications will display the results if there are any results for that particular string. As many of the local languages do not have much of the content posted on web these searches often end up with less information or no information. SoCCS application protocol overcomes this issue by converting information from one language to another and from one format to other. SoCCS protocol will also consider the user preference for query response, i.e., users can choose to hear the query results. SoCCS framework helps the non literate users to interact with computers more easily and access the information effectively.

Therefore, the current project aims at bridging the digital divide by developing a user-friendly and highly economical information access system that can be accessed using various user interfaces through text, image, audio and video. The SoCCS implementation aims to limit the cost of buying a new device for information access while isolating the system changes and upgrades to the server side.

The SoCCS information access system of the current project was developed using client-server architecture and the concepts of E-Systems for public health [1]. The graphical user interface on the client side is based on a variety of concepts published previously [8, 13, 15]. The SoCCS application design is generic to accommodate various versions of frontend designs depending on the user community. The SoCCS application design is based on the request-response methodology not limited by the natural language, client software/hardware requirements, and the type of request.

CHAPTER 2

OVERVIEW

2.1. Requirements for SoCCS Design

A SoCCS architectural framework can be considered that would help non literate and linguistically diverse users to access information using text free interfaces. An e-System approach will deal with information access for non literate and linguistically diverse users [1]. According to Garai and Shadrach [7], the Public Health and General information should include the following:

- Websites for providing awareness about the diseases
- Online libraries which help easy access of information to common man
- Online market information to check and update the current market trends
- Online training to train NGOs and other village information officers
- Web learning tools for interactive reading and information access
- E-counseling of sexual diseases to protect the privacy of the user
- Knowledge base of latest information on medical information

By considering the key concepts for developing an information access system for a diversified country like India, the SoCCS architectural framework implementation should have the following qualities:

2.1.1. Inclusive

India has over 300 million non-literate people, who have little access to on-line information such as public health. A text-free User Interface (UI) can cater to need of this population segment. The SoCCS should cater the information about general issues to the underprivileged sections of the society.

2.1.2. Dialectic

When people have opposing ideas, dialectic debate will consider all the available ideas and arrives with an idea which is acceptable to all the people. The SoCCS needs to be developed in a diversified county like India, SoCCS design should consider different approaches and build an eSystem which will serve large domain of users [1].

2.1.3. Adaptive

SoCCS should have the ability to be adaptive to change, innovation and technology. SoCCS architecture needs provisions and addresses the variety in the environment for a diverse country like India. In Dooley's words [3], "This should be designed in the line of a complex adaptive system that can change and shape itself according to the demands of the environment, so that control and order become emergent, rather than pre-determined".

2.1.4. Evolving

SoCCS should be designed to evolve with the ever changing requirement of its users and its social conditions. It should be able to accommodate different requirement that may raise in future.

2.1.5. People Sensitive

SoCCS for a population with diversified languages and cultures will concentrate on local languages, literacy rate of the users. SoCCS user interface should be designed for easy navigation and understand of its users.

2.3. SoCCS and e-Governance

Kanungo [9] defines E-governance as "the application of information and communication technologies to transform the efficiency, effectiveness, transparency and accountability of

informational and transactional exchanges within government, between government and government agencies of national, state, municipal and local levels, citizen and businesses, and to empower citizens through access and use of information”.

The advantages of E-governance are [2]:

- Governance
 - Transparency in public work
 - Public participation in social and political processes
 - Promotion of democratic society and its diversity
- Public Services
 - Efficient governance
 - Optimized services to people
 - Public access and responsibility to information
 - Government’s accountability for services and spending
- Management
 - Managing voluminous information and data effectively
 - Simplicity, efficiency, and accountability
 - Online Information Services
 - Secure communications

2.4. SoCCS and e-Systems

According to Chowdhury and Medhi 2010, we need to consider three different dimensions for building an information access system for a diverse population like India.

2.4.1. Population Factor

We consider the primary users of SoCCS application protocol are non literates and linguistically diverse users.

2.4.2. Mode of Access to Information

Mode of access is primarily using computers with basic configurations and handheld devices. Web access is available in most of the places, including rural areas.

2.4.3. Types of Information

SoCCS framework will support different type of information in response to user's request. The types of media considered for SoCCS design are text, image, audio and video.

2.4.4. Content Type

Information is available in four formats: text, picture, audio, and video.

According to Chowdhury and Medhi [1]:

- Content: Content depending on the priority and frequency of usage
- Information type: Depending on the content we have text, image, audio and video
- Language: native languages
- Literacy: non-literate, linguistically diverse users
- User's Interface mode: text-free, audio, video and handled device

Considering e-System for public Health and the architecture, we worked towards building a working prototype for information access system which can serve the non-literate users and people with linguistic diversity.

CHAPTER 3

ARCHITECTURAL AND DESIGN APPROACH

IN THIS CHAPTER WILL DISCUSS ABOUT THE ARCHITECTURAL AND DESIGN OF SOCCS APPLICATION PROTOCOL.

3.1. Architecture

Considering a holistic systems approach, we took the conceptual architectural framework of ePH system proposed by Chowdhury and Medhi [1]. SoCCS design concentrates on building an information access system protocol that can accept requests from different user friendly text free interfaces [8, 13, 15] and respond with the best available data.

The architectural framework has four primary functional components [1] are:

- User Agent
- Request Interpreter
- Translator
- Media base

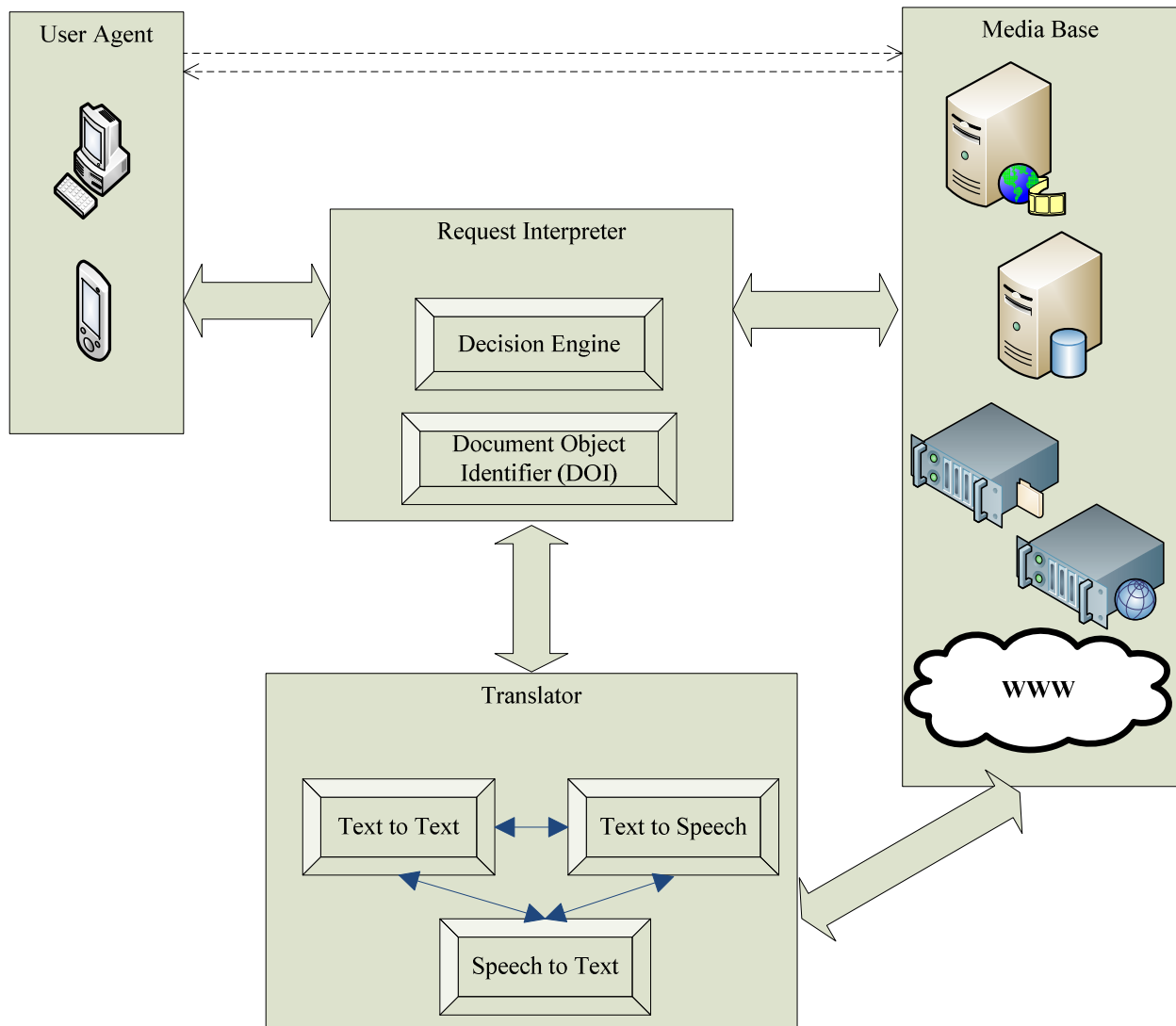


Figure 3.2. SoCCS Block Diagram

3.1.1. User Agent (UA)

The User Agent is the starting point for SoCCS system. It can be a computer connected to the web or a mobile phone. Users have the choice of entering a query (question) through a text based interface or a voice based interface. User Agent may either have the intelligence to convert a voice to text and send it to the Request Interpreter, or just blindly send a voice data to Request Interpreter. The complete operational logic is isolated from UA, which helps running UA on a

basic computer with less processing speed and memory. It also helps isolating system updates/upgrades from user agents which are installed in different geographic places.

The User Agent (UA) builds a request and forwards it to the Request Interpreter (RI). UA will then receive a response from RI and depending on the content of the response; UA will display the content or use the content to stream data from streaming server.

UA supports various MIME content types for displaying images, Unicode text etc. UA should also support and understand user profiling like native language, preferred default content type depending on user's demographic location and user's ability to read/write. UA is capable of forwarding different content types like text, image, audio and video data to RI. These audio and video data can sometimes be huge, so UA should be capable of doing chunked transfer. UA uses the timeout mechanism to wait on a request sent to RI. It supports the streaming technology for streaming audio and video data from streaming server (Media Base).It acts as a streaming client and support these streaming protocols like http streaming, RTSP, RTMP, MMS, PNM etc.

For supporting various natural languages, UA handles Unicode data. UA displays the natural language character strings received from RI as the response to a request sent from UA. UA does the terminal negotiations with RI about the format it supports.

Communication between UA-RI is through HTTP 1.1 POST message. Clients which implement HTTP Post can send data to RI and receive data from RI. The response from RI may contain Unicode text or a streaming link for the resource. UA should be implemented to support both decoding and displaying of content received from RI.

3.1.2. Request Interpreter

Request Interpreter serves as the broker for fetching the information requested by the User Agent from Media Base. User agent sends request towards Request Interpreter. The

Request Interpreter contacts the Media Base for the requested resource. A user who can read and write in his native language will send the request using his local language keyboard or using the voice option. The Request Interpreter uses Translator module to convert the user request and contacts the Media Base to provide the response to user agent. Request Interpreter is the decision engine for providing best possible response to users when the requested information is not present on the Media Base.

RI uses DOI to convert search requests to unique resource location. The Document Object Identifier (DOI) System [18] is an ISO standard for identifying content objects with the unique name assigned in the domain. DOI name will not change with time. The DOI System provides a framework for persistent identification of the resource. When a request is received from UA in text or audio format, RI will check if the request needs any translation. If request needs translation then RI will pass the data to translator and receive the translated text in ASCII format. RI will then pass the requested data to DOI for finding the data resource location. Using Resource location it will try to get media from media base.

Incase RI does not find the exact mach for the requested resource; it will make a decision depending on the availability of the requested message in other formats i.e., if a request is made for a Spanish text file and the Spanish text is not present on the Media Base, RI will look for the resource file in other language or format and try to build the Spanish version of the file using the translator module. By doing this we can serve most of the native languages as we have good resource for English language across the web.

RI will also try to build best possible response incase if a request resource is not present in any of the supporting languages or formats i.e., If a request is made for video output and the video resource is not present in the requested language, RI will try to play an audio file which is

having the same information. It helps in serving the communities where there are no much video records about the diseases in their native language. By playing a alternative resource, SoCCS can help in educating the local communities with high reliability and availability.

RI initiates the translation of data resource; RI will send the translation request with the source file location, destination file location, destination language and resource type i.e., text or audio. RI will receive the translation response from translator and build the response to UA using the newly created or translated file as the resource. Translator uses the destination location for saving the converted file.

RI will do the terminal capability negotiations with the client. If the client can only support text data RI will send the text data instead of sending audio response. This will avoid users getting no response if the response format is not supported by the client. RI will fall back to the supporting formats when the requested format is not supported by the client.

3.1.3. Translator

Translator module will accept the request(s) from request interpreter for translating data from one format to other and acknowledge the request after completing the translation. Translator is capable of converting data from one format to other or from one language to other language. Example: English text can be converted to Spanish, English audio can be converted to English text and English text can be converted to English audio. Current translators can do most of the text translations needs some more development efforts in audio to text conversion for native languages in developing and under developed countries. RI will initiate the translation by sending a request with the source file location, destination file location, destination language and resource type i.e., text or audio. Up on receiving the translate request from RI, Translator will fetch the data from media base and do the required translation. The translated data is push back

on to media base and a response is sent to RI with the updated resource location. RI will use this translated resource location to build the response towards UA.

3.1.4. Media Base

Media base is the resource center for SoCCS application protocol. Data may be any format i.e., text, pictures, audio and video. Media base can be viewed as a cluster of module which can serve the requests coming from request interpreter. Media base consists of multiple components like content provider, file system, database and www. Content providers can be categorized based on media types such as text, image, audio, video for different languages. RI will do the media lookup on the media base, which will respond with the content of the resource if it is a response type is text otherwise media base will respond with the resource location.

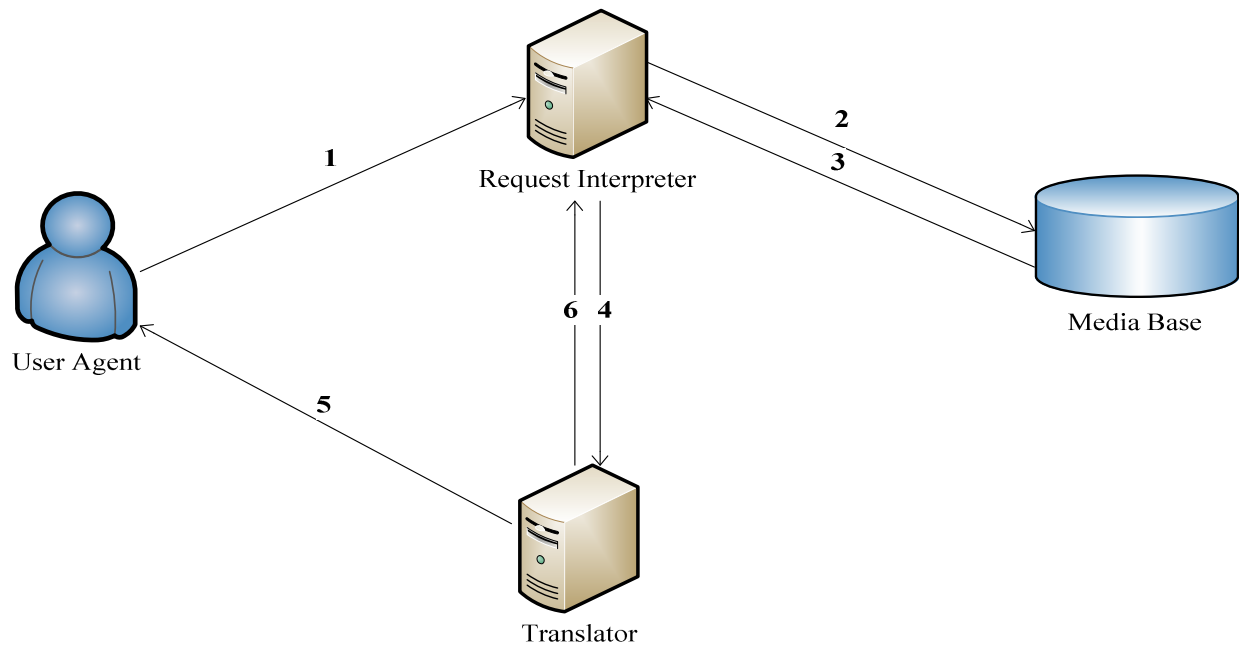
3.2. Possible Cases for Data Flow between Different Modules

We first consider possibly difference ways for data flow and discuss their advantages and disadvantages. We show UA interaction with streaming server as it is common in all the case and we discuss its working sequence separately in different sequence diagrams.

3.2.1. Case1: Translator Responding for the User Request

Assumption: User requesting for data which needs translation at remote end (Request Interpreter).

Request interpreter checks the media lookup response for any translations and if translation is required, then Request Interpreter send the data to translator for translation. Translator will do the necessary translation and sends the response directly to the User Agent as shown in fig.3.3.



1. User Request
2. Request Interpreter query towards Media Base
3. Media Base respond back with data
4. Request Interpreter forwards data for translation towards Translator
5. Translator translates the data and sends it to User Agent
6. Translator responds to the Request Interpreter about the translated data

Figure 3.3 Translator responds to User Request

Pros:

- Service response time will be less.

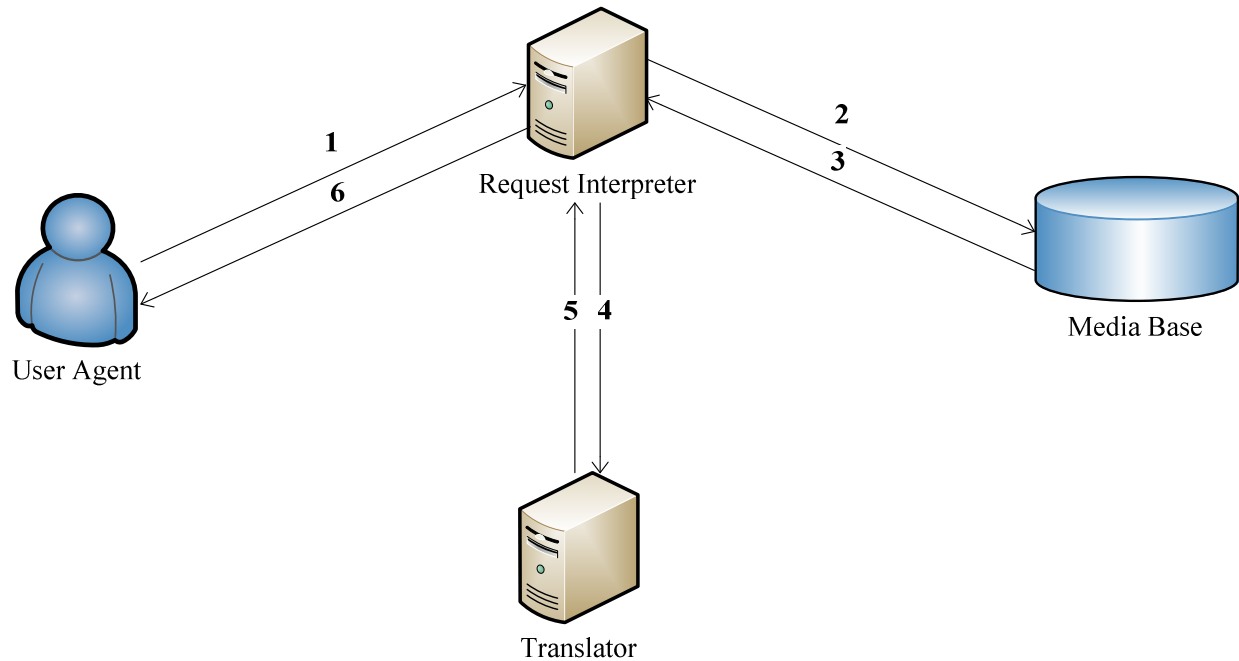
Cons:

- Difficult to maintain the state of the request and response.
- If the same requests is made again then the process has to be repeated as the translated data is not updated in the media source. For repeated requests it will consume lots of process time for doing the same operations.

- Load on the translator will be increased because of the streaming/upload to the user agent. (Translation is a tedious job).

3.2.2. Case2: Request Interpreter Handles the Translation Data

Request interpreter checks the media lookup response for any translations and if translation is required, then RI sends the data to translator for translation. Translator will handle the translation request received from RI and respond to the RI with the translated data, RI will respond to the User agent with the translated data. In this case there is no connectivity between Translator and Media Base as shown in fig. 3.4.



1. User Request
2. Request Interpreter query towards Media Base
3. Media Base respond back with data
4. Request Interpreter forwards data for translation towards Translator
5. Translator responds to the Request Interpreter with translated data
6. Request Interpreter responds to the User with data

Figure 3.4 Request Interpreter handles the Translation data

Pros:

- RI will have the complete state of the request at any instance.

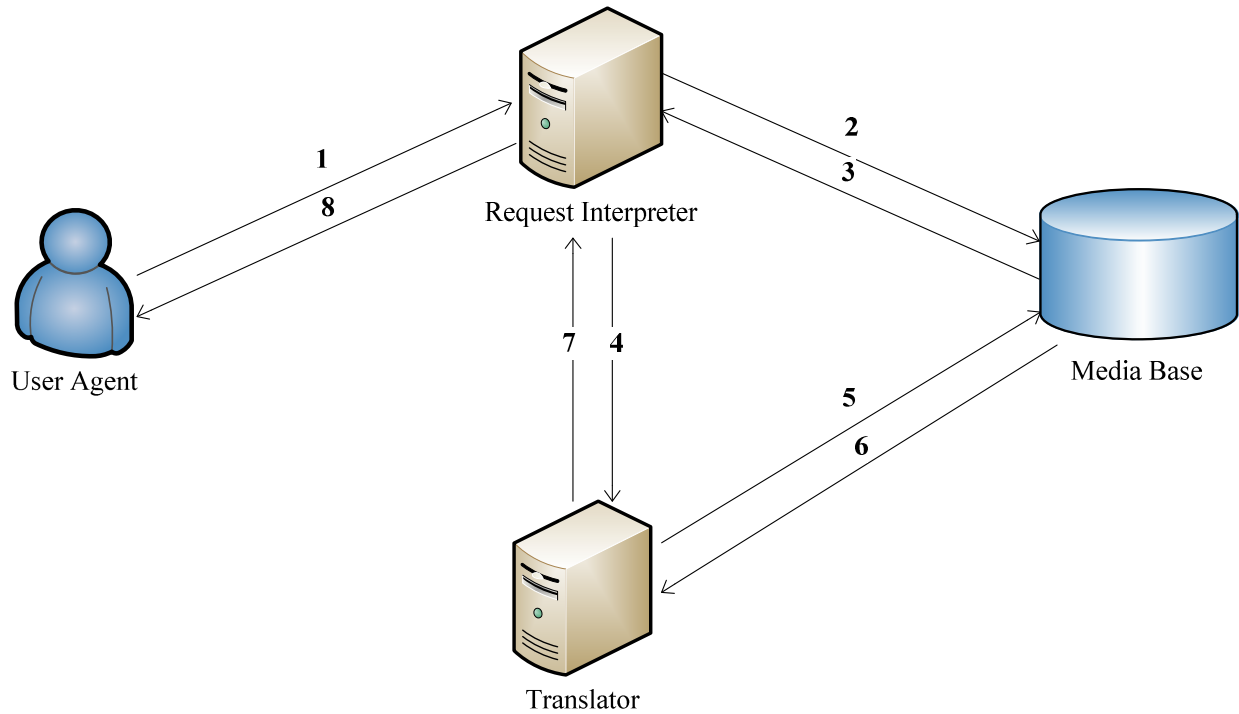
Cons:

- In case of multimedia data the response will be delayed because of the data flow at multiple stages i.e., Media Base to RI, from RI to Translator and from translator back to RI.

- Translated data is not saved on the Media Base, which leads to translation for the every query.

3.2.3. Case3: Request Interpreter Forwards the Data Handle to Translator

Request interpreter checks the media lookup response for any translations and if translation is required, then RI sends the data Info to translator for translation. Translator will connect to the Media Base and fetches the data and performs the necessary translation and writes back the translated data back to Media Base with a new DOI and informs the RI about the new handle. RI will respond to the UA with the newly created data handle. So, that UA can use this handle to stream in data from the Media Base as shown in fig. 3. 5.



1. User Request
2. Request Interpreter query towards Media Base
3. Media Base respond back with the DOI
4. Request Interpreter forwards DOI towards Translator
5. Translator connects to Media Base and fetch the data for translation
6. Translator inserts the translated data on Media Base with a new DOI
7. Translator responds to the Request Interpreter with new DOI
8. Request Interpreter responds to the User with data

Figure 3.5 Request Interpreter forwards data handle to Translator

Pros:

- State of the request is maintained by RI at any instance.
- Writing translations back to Media Base will save the response time if the requests are repeated and they need translation.
- The response time for any request will be minimized as the data flow is only in between translator and Media Base.

Cons:

- RI has to wait till the translator writes back the translated data to Media Base and respond to RI.

By considering above three cases for data flow between different modules in SoCCS framework, we choose to implement Case3, which eliminates communication between User Agent and Translator. By using Case3, we can handle repeated request efficiently. Request Interpreter will have control over the state of the machine at any instance.

3.3. Design Algorithm

We now present the overall design algorithm; its flow diagram is shown in Figure 3.6.

1. User enters request information in local language or in audio format using the user interface.

2. Request interpreter (RI) receives the request from user agent.

IF input_data_type equal to local language (Unicode)

Then convert data using translator (text-to-text)

ELSE IF input_data_type equal to Audio

Then convert data using translator (Speech-to-text)

ELSE

No Translation needed.

END IF

3. Request interpreter passes the input text to Document Object Identifier module (DOI) for resource location.
4. Check Terminal Capabilities.

IF Response_type NOTSUPPORT

 Response_type = fallback type;

ENDIF

5. RI queries Media base (MB) for the resource data.

IF Resource_Data

 EXISTS

 IF Response_type equal to TEXT || IMAGE

 Get Resource Data from Media Base

 ELSE IF Response_type equal to AUDIO || VIDEO

 Get Resource Streaming URL

 END IF

 NOT EXISTS

IF Resource_Data

 EXISTS in any other language or format

 Send Resource location and New DOI to translator (depending on translator capabilities,
 best possible response data and type is returned). GO TO 5.1

 NOT EXISTS in any other language or format

 DATA not found

ENDIF

ENDIF

6. Build response and send to User Agent

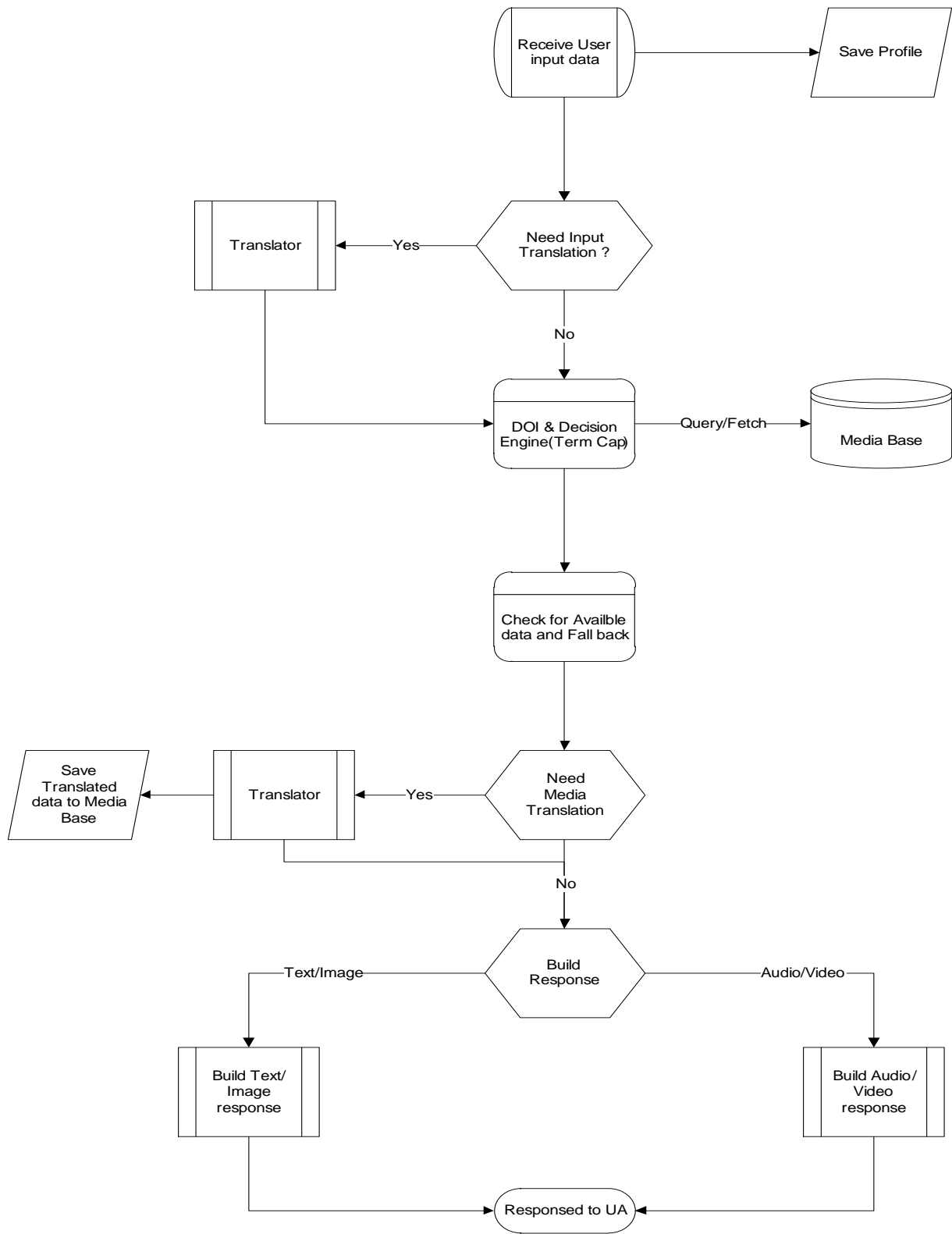


Figure 3.6 SOCCS Design Flow Diagram

3.4. Communication between Modules

In this section we will discuss about the communication between different modules in SoCCS.

Communication between UA – RI is based on HTTP/1.1 protocol. POST method is used to send the data (text/Voice/Image) from UA to RI. RI will receive the request from UA and responds back to UA with the requested data. RI is implemented as a customized Web server which can handle data requests from UA. UA can send data request either as text or voice. Once RI receives the request, RI will translate the request to text if the request is voice format. RI will query Media Base for the corresponding data and if it finds the data then it will respond back to UA with the data found in Media Base. If the requested data is not found on Media Base then RI will send the translation request to Translator. Translator will translate the data present in Media Base and assigns new DOI to the newly translated data and responds back to RI with the DOI. RI will build the response and sends back to UA. Upon receiving response UA will start display text or streaming with the streaming server using streaming protocols like RTSP.

3.4.1. User Agent – Request Interpreter Communication

We discuss about the features of HTTP 1.1 and how we are inclined towards using HTTP 1.1 for communication between User Agent and Request Interpreter in SoCCS framework.

HTTP is a stateless object-oriented and application-level protocol. The typing and negotiation feature of HTTP can be used to negotiate user agent capabilities with request interpreter. Client-server architecture is used between UA and RI.

HTTP uses the concept of channeling for multiple data streams. Each object holds its own separate channel for communication purpose and all communications are done in “session

layer”. Channeling empowers the redirection of meta-information over same connection or channel which plays major role when working with multimedia streaming technologies.

The User agent connects to the server and sends a user request on to the server. Data typing and type negotiation features of HTTP can be used in SoCCS to provide the fallback data response when the user agent does not support the response content. HTTP uses MIME Content-Types which allows all media types and are widely accepted by most of the applications.

HTTP provides a mechanism for the client to specify the server about which language and character set are being acceptable by the user. HTTP/1.1 supports:

- Server-driven negotiation. The request interpreter receives language and character set user preferences from the user agent (client) in the header part and server than matches the user preferences.
- Agent-driven negotiation. The client opts from one of the available representations that server sends. The client then resubmits the request either automatically or with user intervention.

HTTP has no limitation for the body of messages. It can carry any body length which helps us in transmitting large amounts of data during audio request. The recipient always expects to know where the message ends and only after buffering the entire message server prefers to respond in return. Sender specifies body length in Content-Length header.

With the idea of Chunked Transfer-coding, HTTP/1.1 resolved the issue with unlimited message bodies. The message body from sender is divided into chunks and each chunk is pre-pended with length of the chunk. It also appends a zero length chunk to the end and the sender uses Transfer-Encoding mechanism to explain the use of chunking. Recipient receives an alert

from the sender with the existence of the trailer in the form of Trailer header, which lists the set of headers deferred until the trailer.

To sync up with the future protocols, HTTP provides an Upgrade Request header in the message body. This provides the client an opportunity to inform a server the set of protocols that are being supported.

3.4.2. User Agent – Media Base Communication

Once the request interpreter responds with the URI, UA will start streaming the data using one of the streaming technologies i.e., RTSP, RTMP, MMS etc. Streaming technology will take of the long delays of response for large amounts of data like a video response. Streaming technology balances the goal of bandwidth limitations over the Internet which makes it more significant.

Advantages of streaming protocol are:

- Less response time
- Eliminating content piracy
- Viewer statistics
- Quality of service

User agent must have a player which supports streaming protocols like RTSP, RTMP, and RTP etc.

3.4.3. Request Interpreter – Translator Communication

The communication between RI and translator is based on RESTFUL architecture.

In Fielding's words [6], "Representational State Transfer (REST)'s client-server separation of concerns simplifies component implementation, reduces the complexity of connector semantics, improves the effectiveness of performance tuning, and increases the

scalability of pure server components. Layered system constraints allow intermediaries-proxies, gateways, and firewalls-to be introduced at various points in the communication without changing the interfaces between components, thus allowing them to assist in communication translation or improve performance via large-scale, shared caching. REST enables intermediate processing by constraining messages to be self-descriptive: interaction is stateless between requests, standard methods and media types are used to indicate semantics and exchange information, and responses explicitly indicate cache ability”.

REST architecture consists of states and functions on remote server and are considered as resource, each resource is identified by a unique URI on the Translator module. REST provides easy implementation and independence of the interface against added services such as proxies, firewalls and gateways between Request Interpreter, Translator and Media Base. REST acts as a single interface to browser and access a service, possibly distributed on multiple translation servers. REST uses standard formats like HTML, XML and JSON ensures compatibility.

RI will build a HTTP GET request to the translator’s URI. The result of a request to the Translate API is a simple JSON object.

CHAPTER 4

SEQUENCE DIAGRAMS FOR REQUEST HANDLING

In this chapter we will discuss about different sequence diagrams that shows how processes operate with one another and in what order. With the above mentioned design, we will have four different scenarios for handling requests from the User agent:

- No Translation required for Request or Response
- Translation required for the Request
- Translation required for the Response
- Translation required for Request and Response.
- Streaming media (common sequence for all the above four sequences)

4.1. No Translation for Request and Response

Client sends a request to request interpreter as a HTTP post request. RI will check whether request data needs any translation. In this scenario there is no request translation required and RI will query the DOI module for the resource location. RI will check the response type. If the response type is text/image then RI will fetch the data from Media Base and build the response. If the response type is audio/video then RI will build the response with streaming link and responds to client with the response. Figure 4.1 shows the flow of control between modules with time. In this scenario we consider the requested resource is present on the media base and it needs no translation to present it to user agent. Request Interpreter also checks for the user agent capabilities before sending response. If the client supports only text format and requested a audio file, then RI will do a fallback the format type and respond back with a user agent supporting format.

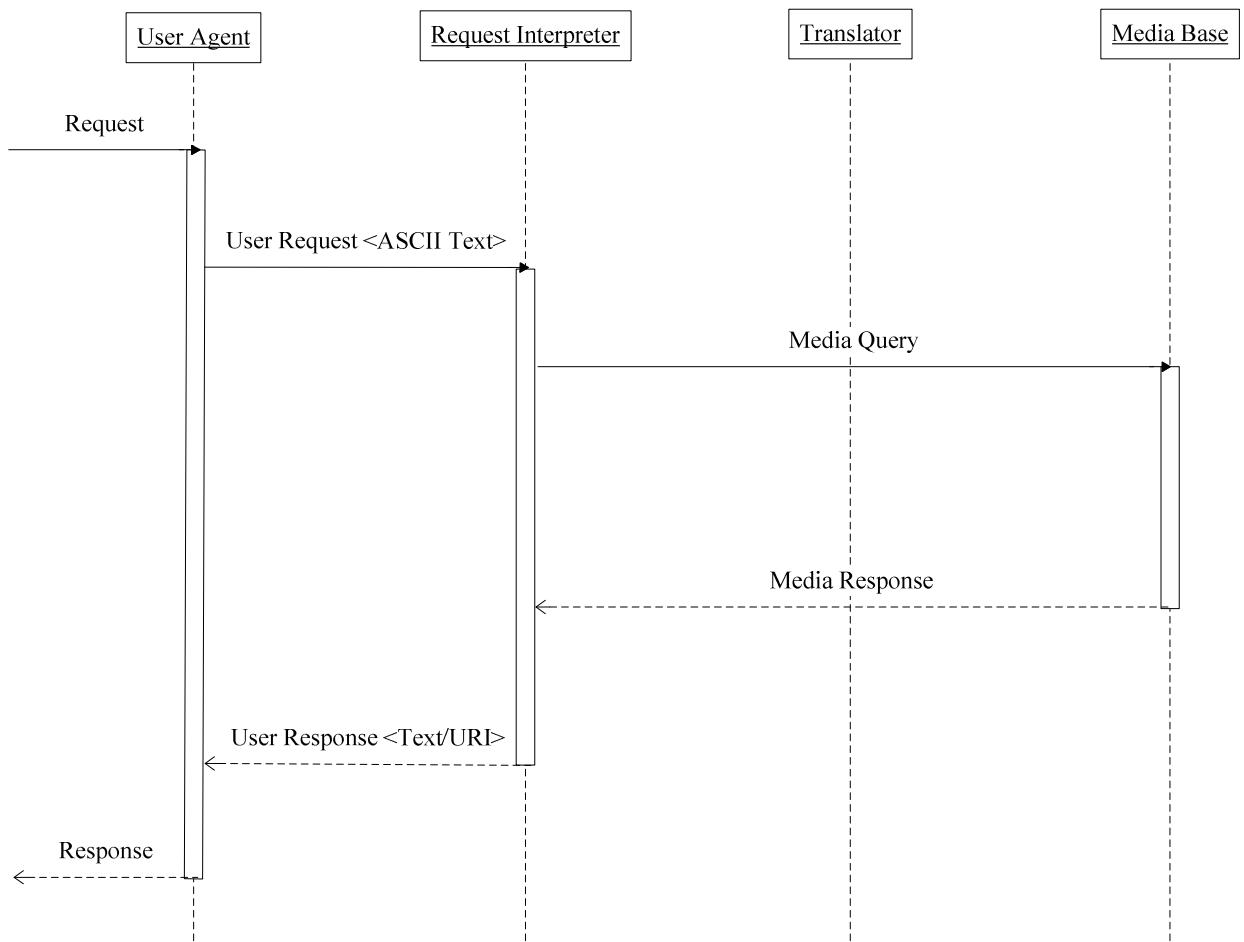


Figure 4.1. No Request and Response Translation

4.2. Request Translation

Client sends the request in audio/ Unicode text as a HTTP post Request. RI will receive the request and check the translation required. RI will forward the request data to translator for translation to text. Translator will translate the data to English text and responds to RI. RI will input the translated text to DOI and gets the data location for this request. RI will use the data

location to query data from media base and in this scenario we consider that an exact resource is found in the media base (i.e., language resource and resource format). RI will check the response format requested by client and builds the response considering user agent capabilities as shown in fig. 4.2

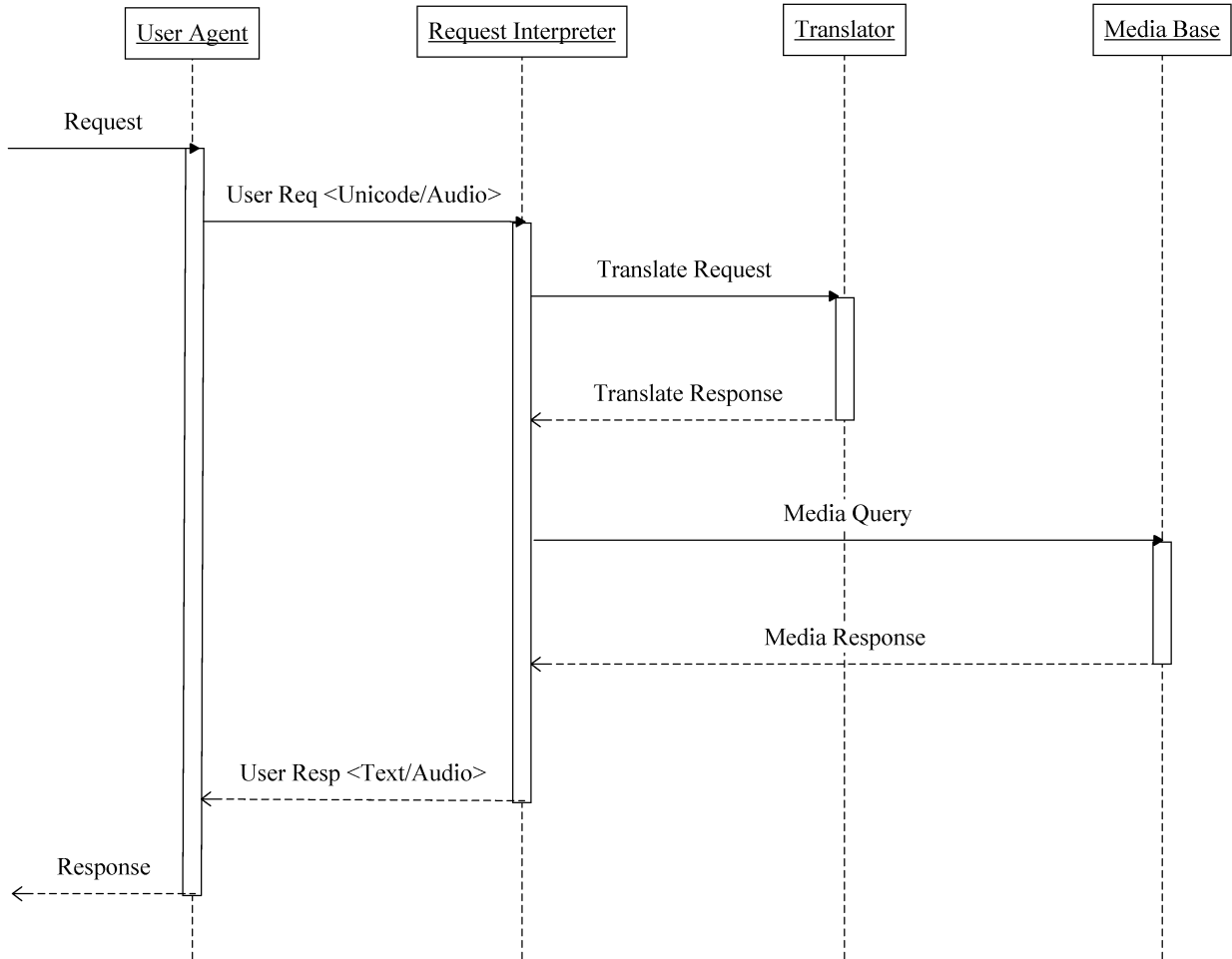


Figure 4.2. Request Translation

4.3. Response Translation

Client sends the request in ASCII text as a HTTP post request. RI will receive the request from client and forward the request data to DOI. DOI will respond with the information location. If the information needs any translation from language to language or text to speech then RI will forward data to translator. Translator will generate a new data file with the translated data and responds with the new location. RI will use this new data location to build the response considering the terminal capabilities as shown in fig. 4.3.

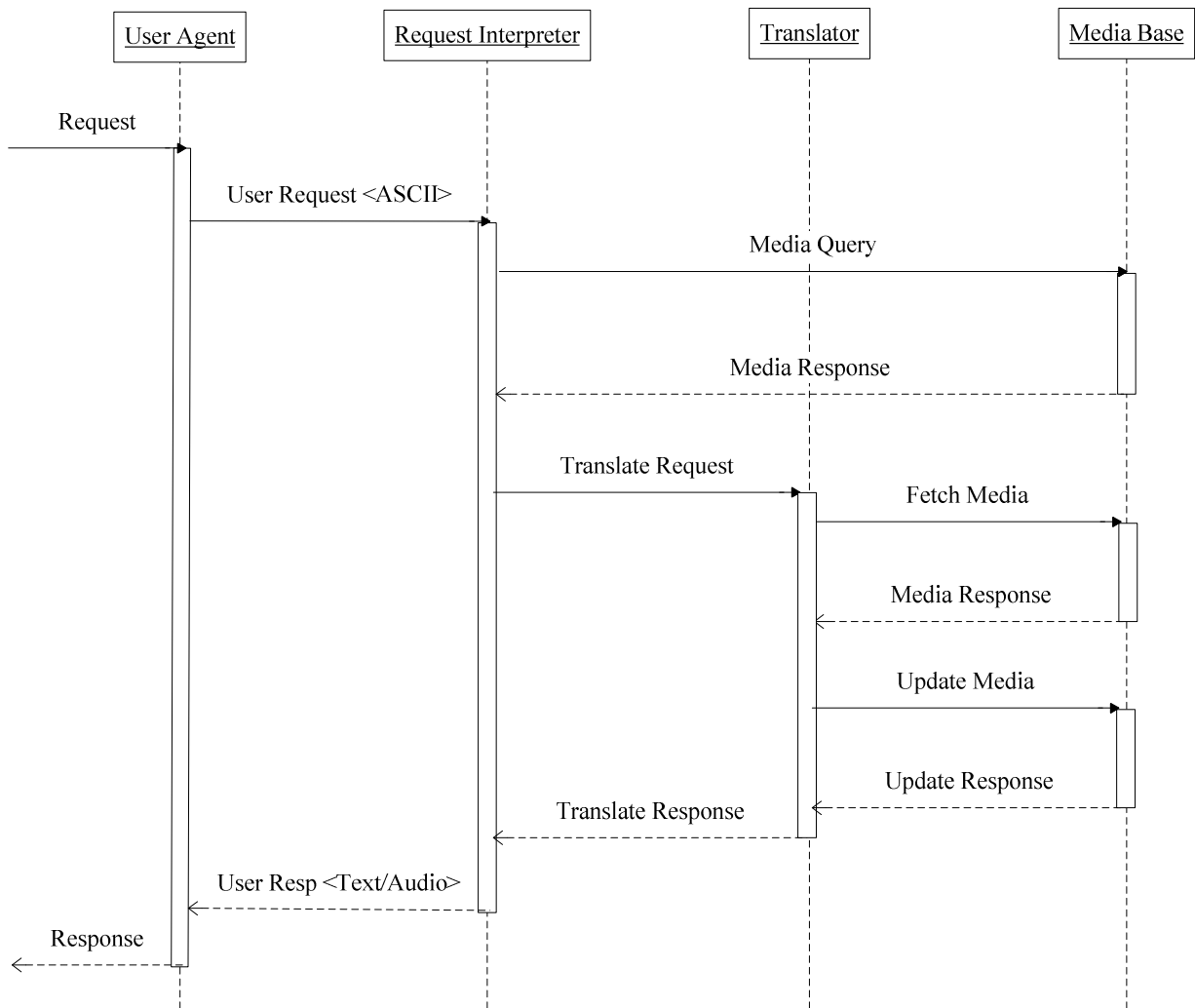


Figure 4.3. Response Translation

4.4. Request and Response Translation

Client sends the request in Unicode or in audio format as a HTTP post request. RI will receive the request from client and check the input for translation. RI will submit the input data to translator. Translator will respond back to RI with the translated data. This data is passed to DOI, DOI will respond back with the data location. If the data is not present in the requested format; RI will submit the data to translator for converting to requested format. Translator will create a new file with the translated data and respond to the RI. RI will use the new translated data for building the response depending on the terminal capabilities as shown in fig. 4.4.

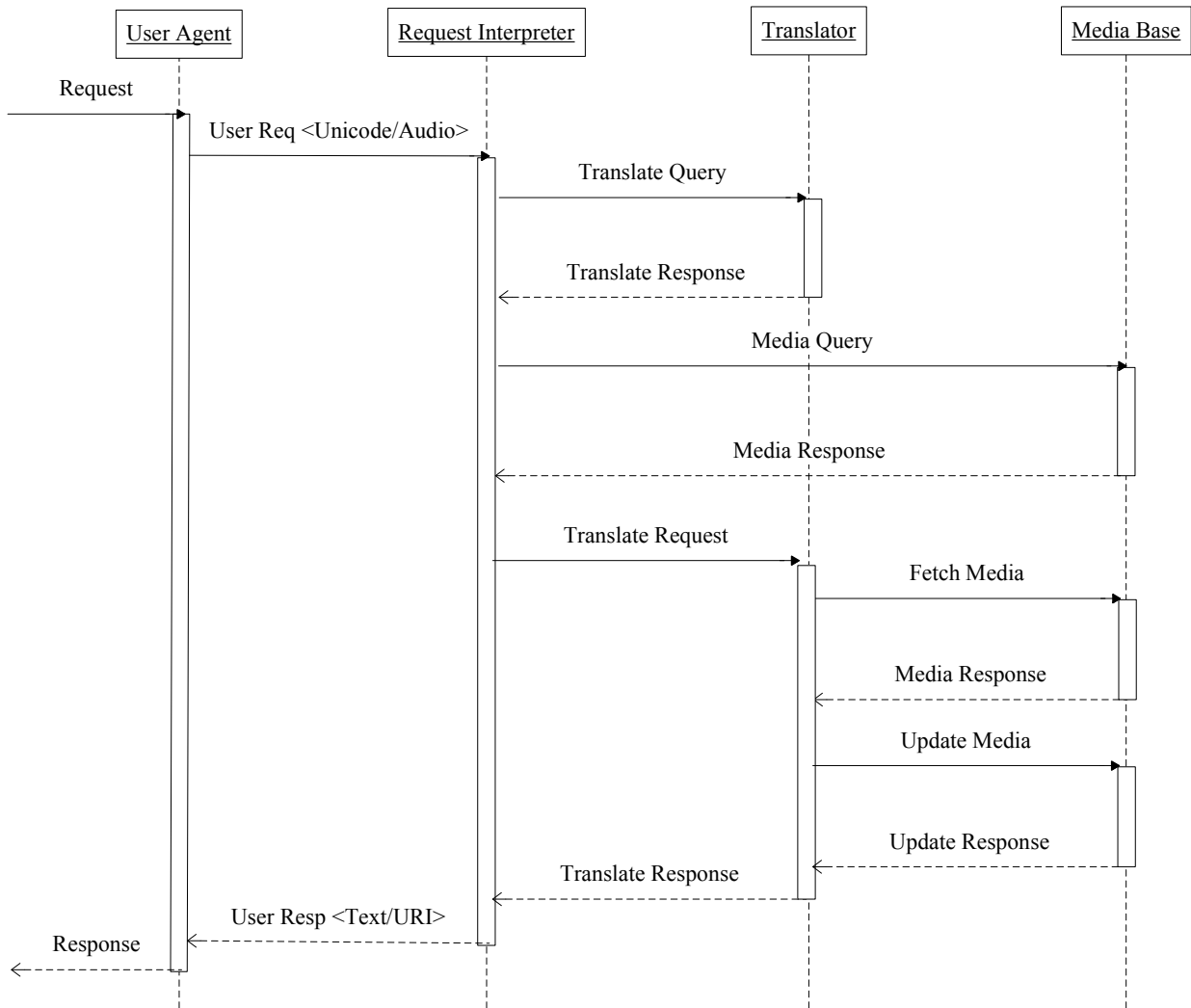


Figure 4.4. Request and Response Translation

4.5. Streaming Sequence Diagram

User agent receives a metafile about the resource location of the requested data from RI. User agent will use this resource location to initiate a streaming session with the streaming server present on the media base. User agent can use any of the streaming technologies like HTTP,

MMS, RTSP, PNM and RTMP for establishing and control the streaming session with the media server and followed by a RTP or proprietary transport protocol to receive the data stream.

We consider using Real Time Streaming Protocol (RTSP) as a default network streaming control protocol. RTSP uses TCP for communication between user agent and the request interpreter. Real-time Transport Protocol (RTP) is used for transmission of media. Figure 4.5 shows the sequence of RTSP and RTP messages.

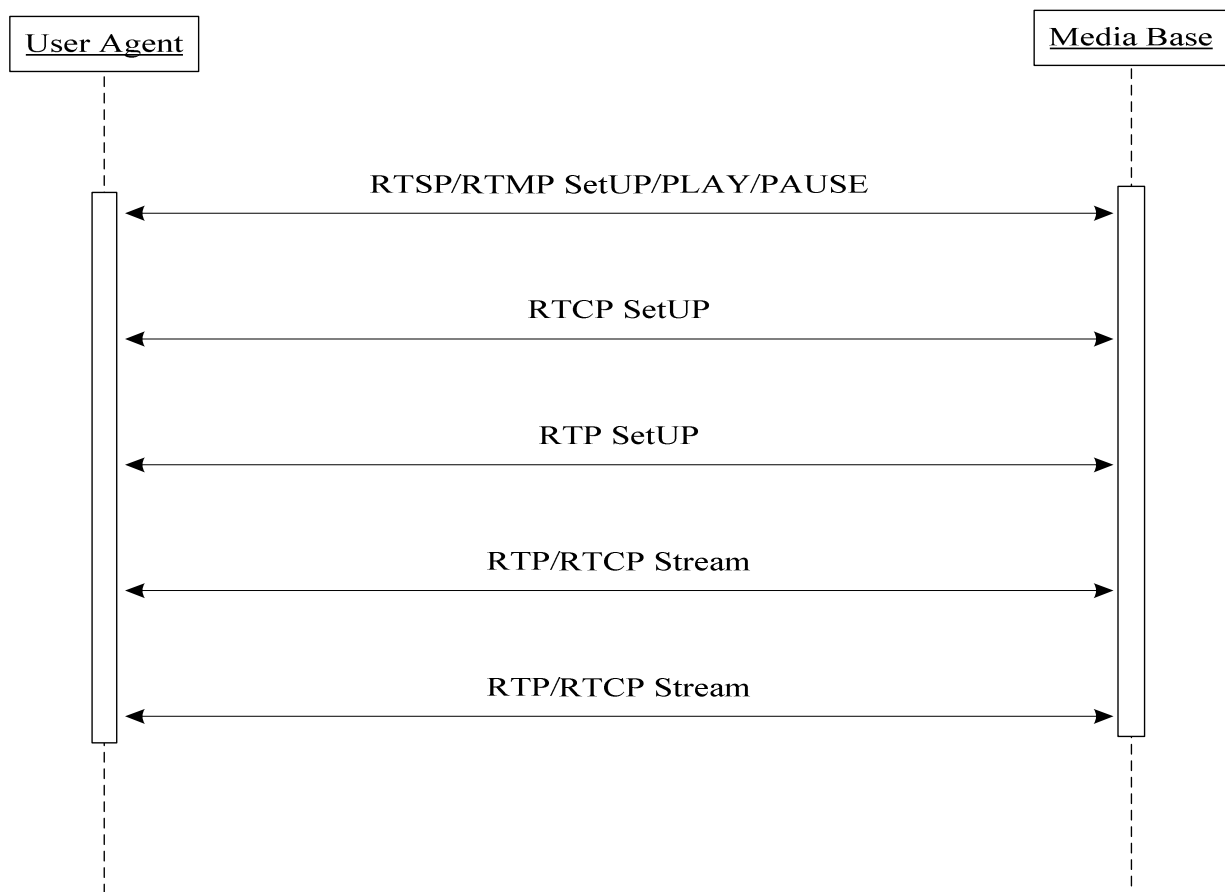


Figure 4.5. Streaming Server

CHAPTER 5

IMPLEMENTATION OF SOCCS FRAMEWORK

This chapter discusses the implementation of SoCCS application using e-Systems architectural framework. For building SoCCS application various open source applications are used in different module.

We implemented SoCCS application using the SoCCS application protocol as discussed in chapter 3 and 4. Implemented SoCCS application can be viewed as a web application which can be accessed from any computer with web browser, micro phone, sound speakers and the supporting software for handling audio and video devices. Client can access information by entering text or speaking in his native language. SoCCS server which is RI will receive the request using HTTP 1.1 and does the translation if required and pass query text to DOI for resource location. RI will use the resource location to get the information from media base. If the media resource is available in the format requested by SoCCS client, then RI will build the response use text/ URI and send it to client. Otherwise RI will forward the translate request to translator and gets the translated resource location, which is used in building the response towards SoCCS client. Translator, up on receiving request from RI will fetch data from media base using the resource location provided by RI and does the translation using the text to text, text to speech or speech to text translators. Media base can be viewed as cluster of data resources like web, content providers, data bases, SAN etc. Media is the cluster of data sources mentioned above and viewed as a one logical entity which will serve the information requests from RI and translator.

SoCCS server is implemented to read the changes to system using configuration file (socc.conf). Configuration parameters are explained in Appendix B. SoCCS supports the

languages supported by translation module. For the list of supported languages and there corresponding language codes refer appendix A.

In the following sections we will discuss in detail about the four functional modules in SoCCS application and how they were implemented using various other modules and how they handle process inputs and generate outputs.

5.1. User Agent (UA)

UA is implemented as a web application running on a web browser. UA is capable of displaying ASCII text, display Unicode text, display images, play audio and video. UA supports the java applets (javasonic ListenUP [24]) and have the media player installed for playing audio and video data. Google virtual keyboard [23] is used to input search request in local languages. By implementing UA as a web client we isolated all the functional logic from end user. User agent used the text free web interface concepts proposed by [8, 13, 15], which helps a non-literate or linguistically diverse user to use SoCCS client for information access.

Applications Used for building UA are:

5.1.1. Web browser

Web browser is used to build request and display response from the server (i.e RI). Web browser supports the HTML 5.0, java script and has plug-in(s) installed for playing audio and video content. Ex: Google Chrome, IE, Firefox

5.1.2. Java Sonics ListenUp Applet

ListenUp [24] provides java applet for voice recording on web pages. Using ListenUp, web users can record audio, upload and playback the audio files to the server for further processing. It supports various compression technologies like Speex, ADPCM and ulaw .

5.1.3. Google Virtual Keyboard

Virtual keyboards [23] helps in entering search text in local languages. It is helpful for users of non-Latin script-based languages. People are comfortable typing in their native local language and virtual keyboard needs no installation. Special characters shown on the on-screen keyboard can be entered by pressing the virtual keyboard's Alt+Ctrl and "up arrow" keys.

5.2. Request Interpreter (RI)

RI is a web server side script developed using PHP scripting language. RI is capable of receiving the request data from UA using HTTP 1.1 and responds to UA with the requested data. RI will check the received search request for any translations. If the search request needs translation then RI will send the translation request to translator and uses the translated text, else it uses the received text to forward data to DOI. DOI will read the search text and builds the unique resource location, where the requested media can be found on the media base. RI will use resource location to fetch the media. If the resource is available, then response is built using the resource, else RI will get the resource location for available formats and uses these available resource locations to generate a translated media to serve the user request. RI will forward the available resource location and where the translated media should be placed on media base. RI will build the user response depending on the user requested format (Default is text-free). If user requested for a text response then resource media is used to build the response, else resource URI is used to build the Meta data into the response. RI will also consider the user agent terminal capabilities for building the response data. If a user requests for video data and has no video plug-in installed on client, to decode video data, then RI builds the response as audio or image.

Application used for building RI is:

5.2.1. Web Server

Web Server handles requests from web browsers and responds with the requested data. Web servers will be integrated with the PHP server and can generate the HTML code upon execution of the PHP server script. Web server will log all the requests coming to the server in a access log file and errors into apache_error log file. Ex: Apache web server

5.3. Translator (TR)

TR is a module which can translate data from one language to another or from one format to another format. TR includes modules for,

- Text to Text (Google Translate), which converts text from one language to another
- Text to Speech (eSpeak), which converts text to voice
- Speech to Text (Sphinx), which converts voice to text and
- Sox to convert audio files from one format to another format.

Applications used for building TR are:

5.3.1. Google Translate

Google Translate [22] is a tool that automatically translates text from one language to another language (e.g., Hindi to English). Google Translate, can translate text in web pages or applications. The response from the Translate API is a JSON object. You can translate text from one language to another language by sending an HTTP GET request to an URI on server. The server responds with a 200 OK HTTP status code and the JSON object data.

5.3.2. eSpeak

eSpeak [20] is a speech synthesizer for various languages. It uses a "formant synthesis" method, which helps in building small size language source files. eSpeak can accept input from a file or from command prompt. It provides a shared library to make calls from other programs.

eSpeak can produce WAV file as a speech output, supports Speech Synthesis Markup Language (SSML), and also HTML. It will translate text into phoneme codes and helps in producing and tuning phoneme data.

5.3.3. Sphinx

Sphinx [17] is a speech recognizer, which is capable of recognizing both discrete and continuous speech.

“Sphinx includes pluggable implementations of pre-emphasis, Hamming window, FFT, Mel frequency filter bank, discrete cosine transform, cepstral mean normalization, and feature extraction of cepstra, delta cepstra, double delta cepstra features. It also includes pluggable language model support for ASCII and binary versions of unigram, bigram, trigram, Java Speech API Grammar Format (JSGF), and ARPA-format FST grammars”.

Sphinx provides utilities like confidence scores, generating lattices and embedding ECMA Script into JSGF tags.

Appendix C shows how to build a sample grammar using JSGF.

5.3.4. SoX

SoX [27] is an application which can convert audio files from one format to other format.

5.4. Media Base (MB)

MB is the storage module for SoCCS application. Media Base can be viewed as a multiple servers performing different functionalities on each server. We implemented as a file system which contents media categorized on language bases. Streaming server (Unreal media server) which supports RTMP, MMS, HTTP, TCP/IP and RTP, is installed to support streaming requests from user agent.

Applications Used for building MB are:

5.4.1. Streaming Server

Streaming server is used for streaming data onto UA. For the purpose of implementing a streaming server in house, we tried different streaming servers and listed there advantages and disadvantages.

- Darwin streaming server
- Video LAN server
- Live stream
- YouTube
- Unreal media streaming server

5.4.1.1. Darwin Streaming Server

Darwin Streaming Server [16], allows streaming media towards clients across the network using the RTP and RTSP protocols. Darwin Streaming Server supports customization to provide better response time in different networks.

5.4.1.2. VideoLAN Server

VLS (VideoLAN Server) and VLC (Video LAN Client) are the streaming server and client applications which can stream MPEG-1/2/4 files, DVDs, satellite/television channels and live videos on the network.

5.4.1.3. Live Stream

The "LIVE555 Media Server" is a streaming server which uses RTSP for streaming data to clients. It supports media file formats like MPEG, MPEG-1 or 2, MPEG-4, DV, WAV, AMR and AAC. The server supports multicast streaming, which enables clients to stream media concurrently. The Live stream server will transmit its streams as RTP/UDP or RTP/TCP(and RTCP) packets.

VLC media player, QuickTime, Amino set-top box and openRTSP are the RTSP/RTP media clients available.

5.4.1.4. YouTube

YouTube is an online video streaming service. Users can broadcast their videos to all the users in YouTube. YouTube can be easily embedded to the websites.

5.4.1.5. Unreal Media Streaming Server

Unreal Media Server [29] is a media streaming server. Unreal Media Server uses TCP/IP for streaming media towards streaming client. The server supports various media file formats. Flash player uses RTMP protocol to stream data from media server and also supports TCP-unicast streaming, RTP Multicast streaming and HTTP(S) unicast streaming.

With multiple data sources available at media base, an effective catalog service helps in identifying and fetching the data from the resource. These catalog services act as a directory for

the resource mapping. By using a sophisticated catalog user requests can be serviced with most relevant data available at different media servers available.

CHAPTER 6

EVALUATION

Screen shots of SoCCS application protocol implementation are shown below.

User preference Page:




The screenshot shows a blue background with the following text and controls:

- SOCCS Application Demo**
- User Preference Settings**
- Choose your Language:** ENGLISH ▼
- Choose your response format:** Audio ▼
- Submit**

Search Options:

Select a Search Type




Text Search:

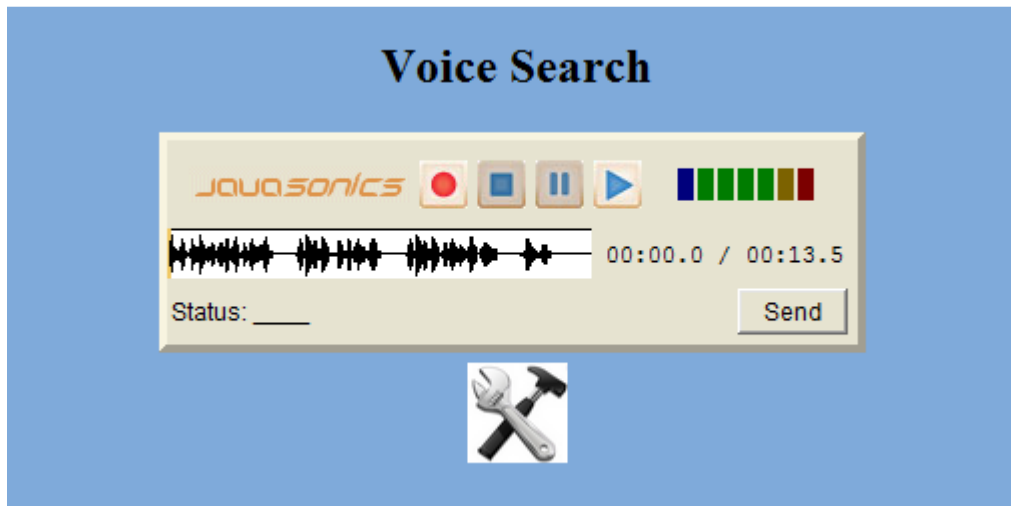
SOCCS Application Demo

Text Search

You can click the buttons on the onscreen keyboard to type. You can also type with your keyboard. please click the [-] button to minimize the keyboard.



Voice Search:



Text Response:

Text Response

A fever 100F (37.8C) is any body temperature elevation.

(36.1C) 97F and 100F between a healthy person's body temperature fluctuates (37.8C) averaged 98.6F (37C) with.

Within this range by balancing the heat produced by metabolism with the heat lost to the environment body maintains stability.

Thermostat that controls the process hypothalamus, located in a little deeper structure within the brain.

Nervous system constantly thermostat, various physical responses to Nda or heated to activate the body;

temperature relay information.

A fever occurs when the thermostat at a higher temperature in response to an infection mainly resets.

Overcome the infection or aspirin or acetaminophen (Tylenol) as such drugs are taken

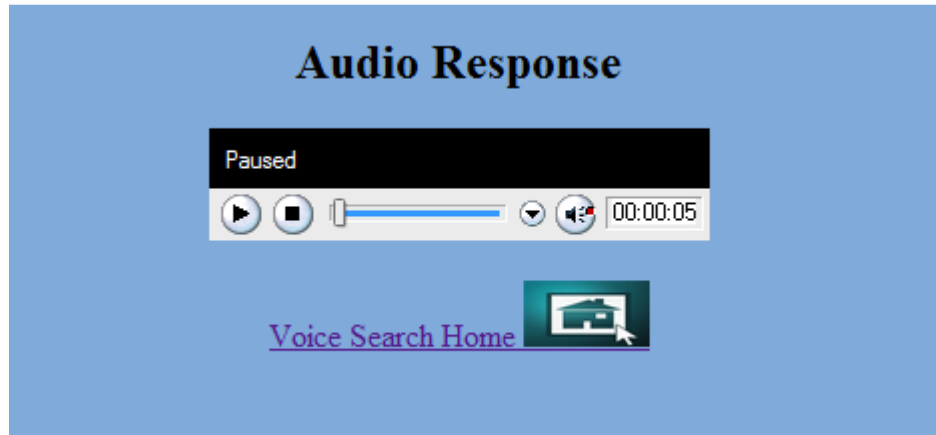
To normal thermostat reset switch on the cooling system of the body: surface and sweat blood to run.

Nothing prevents the growth of bacteria,

Themselves up chemical reactions that help the body's cells, the rapid repair, and the arrival of white blood cells to sites of infection speed.



Audio Response:



Video Response:

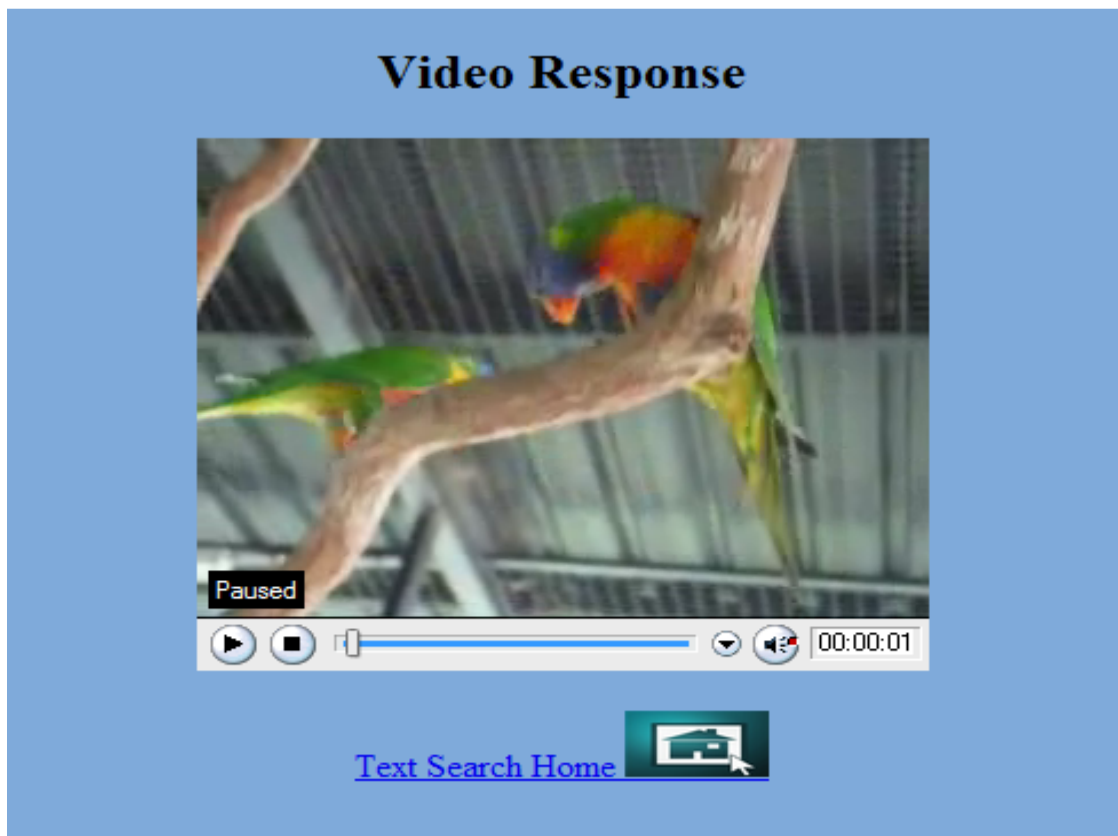


Image Response:



SOCCS Application Performance Metrics:

SOCCS application is tested using both text and voice request interfaces against different media formats. The average resource response time for each case is listed in Table 1 and the number of interactions for each case is 50.

Table 1. -- SOCCS application performance for different types of requests.

Request Interface	Test Case	Process	Average Response Time (Sec)
Local Language Text Interface (Unicode)	Requesting for resource which is available on the media base in the requested format	Input search string conversion + Resource search on Media base + Resource fetching	1.285
Local Language Text Interface (Unicode)	Requesting for resource which is NOT available on the media base in the requested format	Input search string conversion + Resource search on media base + Available resource location + Media language or format conversion + Resource fetch	15.201
Local Language Voice Interface	Requesting for resource which is available on the media base in the requested format	Voice to Text conversion + Search string Identification + search string conversion + Media search + Resource fetching	6.611
Local Language Voice Interface	Requesting for resource which is NOT available on the media base in the requested format	Voice to Text conversion + Search string Identification + Search String conversion + Media search on Media base + Available resource location + Media language or format conversion + Resource fetch	21.322

CHAPTER 7

CONCLUSION

Socio Cultural Communication System is an architectural design for information access system that helps non-literates and linguistically diverse users to access information. SoCCS application protocol is designed and implemented using e-systems architectural design framework proposed by Chowdhury and Medhi [1].

Clients can be programmed as web applications or standalone windows applications. Clients communicate with server using HTTP 1.1 and use POST to transfer information from client to server. Server will receive the query data from client and process the query. Server will accept both text and audio requests. Server supports all MIME content types. After receiving data request from the client, server will process the input query data using translators. Translator module is used to convert data from one format to the other and from one language to the other language. By converting from one format to the other, i.e., from text to audio or audio to text we can server both literate and non-literate users. By converting text from one language to other helps in processing the data request from a local demographic user by using the generic data (Media Base) with the translations. Server will respond to the client or user in any of the format i.e., text or audio. SoCCS default setting is for the non-literate users. Therefore, responses are presented in audio format when output format is not specified. Server uses data handler module to generate unique file address or location for a request. Using this unique address, the SoCCS application will find the corresponding data and provides a response in the format desired by the user. Multiple word grammar to match complex requests can be written using the advanced data handler systems proposed in the design. Server uses this unique data address information and finds the location of the resource and uses this data/data location and responds to the user request

using text messages. Additional audio or video responses will be sent using streaming technology at user's request. Streaming link and protocol will be embedded in the response to stream audio or video data. Client will start streaming the data using any of the default media players using the streaming protocol specified in the response and the location of the streaming data. Streaming server is used to serve the streaming requests from the client browser. Streaming servers can handle multiple streaming requests at a time and allow clients to stream data from the streaming servers. Streaming server can support different streaming protocols and serve the client streaming request using the protocol requested by the client. SoCCS server architecture is designed for future scale up using complex information retrieval algorithms for information specific applications. Therefore, SoCCS application protocol developed by accomplishing this research project will help literates, non-literates and linguistically diverse users to access useful information that enables them to make informed choices.

Some of the challenges for designing SOCCS project

- Designing the system for a user who do not have any prior knowledge on accessing information through computers or handheld devices.
- Communication protocols between modules should be capable of transporting large amounts of data.
- Building a prototype with the above discussed SOCCS design using available translator components.

APPENDIX A

List of all supported languages for SoCCS application

Language	Language-Code
Albanian	sq
Catalan	ca
Croatian	hr
Czech	cs
Dutch	nl
English	en
Finnish	fi
French	fr
German	de
Greek	el
Hindi	hi
Hungarian	hu
Icelandic	is

Indonesian	id
Italian	it
Latvian	lv
Macedonian	mk
Norwegian	no
Polish	pl
Portuguese	pt
Romanian	ro
Russian	ru
Slovak	sk
Spanish	es
Swedish	sv
Turkish	tr
Vietnamese	vi

APPENDIX B

SoCCS Configuration File (soccs.conf)

soccs.conf is the configuration file for SoCCS application protocol implementation. soccs.conf can be used to retrieve language specific information by SoCCS application. Lines which starts with ‘;;’ represents commented lines. Each section start with a language tag embedded in [] i.e., [<language tag>]. All the available language tags is listed in Appendix A.

“Desc” parameter describes the language tag.

“Path” parameter describes the resource location for the language.

Example:

```
[en]
Desc = ENGLISH
Path = “Resource Location”
```

To add a language supported by SoCCS application, add the above three lines in configuration file (soccs.conf) with the corresponding language tag and description values listed in Appendix A.

Search section helps in configuring supported voice commands and their mapping with content.

Example:

```
[search]
one = fever
two = malaria
eight = flu
```

APPENDIX C

Sphinx Grammar

Sphinx-4 uses the Java Speech API Grammar Format (JSGF) to perform speech recognition using a BNF-style grammar. It is used by the Java Speech API (JSAPI)

Example 1: "Hello World" grammar in JSGF

```
#JSGF V1.0  
  
public <helloWorld> = Hello World;
```

Example 2: General Command request Grammar in JSGF

Grammar for commands like "malaria symptoms", "information about malaria" and "malaria information".

```
#JSGF V1.0  
  
public <diseaseInfo> = <startTags> <disease> <endTags>;  
  
<disease> = malaria | cholera;  
  
<startTags> = (symptoms | information) [about | of];  
  
<endTags> = [symptoms | information];
```

REFERENCE LIST

1. Chowdhury, R and Medhi, D.,. E-System for Public Health in India: Towards an Architectural Framework Incorporating Illiteracy and Linguistic Diversity, in Systems Thinking and e-Participation: ICT in the Governance of Society, edited by J. Cordoba-Pachon and A. Ochoa-Arias, IGI Global, pp. 69--91, 2010.
2. Das, S.R. and Chandrashekhar, R. (2006). Capacity Building for E-Governance in India. UNDP Asia-Pacific Development Information Programme. Available from <http://www.apdip.net/projects/e-government/capblg/casestudies/India-Chandrashekhar.pdf>; accessed 2 August 2009
3. Dooley, K. (1997). A complex adaptive systems model of organizational change. In Nonlinear Dynamics, Psychology and Life Science (pp.69-97). Vol.1, No.1.
4. Eemeren, F. H. v. (2003). Anyone who has a view: theoretical contributions to the study of argumentation. Argumentation library, v. 8. Dordrecht: Kluwer Academic. Page 92.
5. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and Berners-Lee, T. Hypertext Transfer Protocol -- HTTP/1.1. Available from <http://www.w3.org/Protocols/rfc2616/rfc2616.html> (1999); accessed 5 August 2009.
6. Fielding, R. T. Architectural Styles and the Design of Network-based Software Architectures. Available from <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.91.9164&rep=rep1&type=pdf> (2000); accessed 20 June 2010.
7. Garai, A. and Shadrach, B. (2006). Processes and Appropriation of ICT in Human Development. In Rural India: Bridging the Research and Practice Gaps. Available from

- http://www.dgroups.org/groups/oneworld/OneWorldSA/docs/TICTEIV_pdf.pdf; accessed August 25th 2009.
8. Goetze, M., Strothotte, T. (2001) An Approach to Help Functionally Illiterate People with Graphical Reading Aids. Available from <http://www.dfki.de/~krueger/sg2001/schedule/goetze.pdf>; accessed September 14th 2009.
 9. Kanungo, V. (2004). Citizen Centric e-Governance in India: Strategies for Today, Vision for Future. New Delhi: Society for Promotion of E-Governance. Languages of India. Available from http://en.wikipedia.org/wiki/Languages_of_India; accessed 25 August 2009.
 10. Krishnamurthy, B., Mogul, C.J., and Kristol., M.D. Key Differences between HTTP/1.0 and HTTP/1.1. Available from <http://www8.org/w8-papers/5c-protocols/key/key.html> (1999; accessed 10 January 2010)
 11. McTaggart, J. M. E. (1964). A commentary on Hegel's logic. New York: Russell and Russell.
 12. Medhi, I., Pitti B., and Toyama K. (2005). Text-Free UI for Employment Search. In Proceedings of Asian Applied Computing Conference. Nepal
 13. Medhi, I. and R. Kuriyan, R. (2007). Text-Free UI: Prospects for Social Inclusion. In Proceedings of International Conference on Social Implications of Computers in Developing countries. Brazil.
 14. Stump, E. (1989). Dialectic and its place in the development of medieval logic. Ithaca, N.Y.: Cornell University Press.
 15. Sunkari, P. INFOKIOSK: An Information Kiosk with Text-Free User Interface, MS Thesis, University of Missouri-Kansas City, December 2010.
 16. Advantages of Streaming Technology, Available from <http://www.videodesk.net/Streaming.aspx>; accessed 20 November 2009.

17. CMU Sphinx - speech synthesizer, Available from <http://cmusphinx.sourceforge.net/>
18. [DOI] The Document Object Identifier System, <http://www.doi.org/>
19. Dhvani Indian language text to speech Engine, Available from <http://fci.wikia.com/wiki/Dhvani>; accessed 20 August 2010
20. e-Speak, Available from <http://espeak.sourceforge.net/>
21. [Handle] Handle System Architecture, Corporation for National Research Initiatives, <http://www.handle.net/overviews/architecture.html>
22. Google Translator, Available from <http://translate.google.com/#>
23. Google Virtual Keyboard, Available from <http://www.google.com/ig/directory?hl=en&num=24&url=www.gate2home.com/gate2home.xml>
24. [ListenUP] Javasonic, URL: <http://www.javasonics.com/>
25. LumenVox – Speech Recognition Software, Available from <http://www.lumenvox.com/>
26. [PHP] Wamp server. Available from <http://www.wampserver.com/en/>; accessed 3 November 2009.
27. Sox – Sound eXchange, Available from <http://sox.sourceforge.net/>
28. Simputer, Available from <http://www.simputer.org/>; accessed 20 August 2010
29. Unreal media server and player, Available from <http://www.umediaserver.net/>
30. [Unicode] Unicode Consortium, <http://www.unicode.org/>

VITA

Venkata Rama Krishna Jamithireddy was born on August 1, 1984, in Hyderabad, India. He was graduated from Andhra University in 2005, with a Bachelors of Technology degree in Computer Sciences.

After working as a Software Developer for Bharti Telesoft, India (2005 to 2008), he began his Master's program in University of Missouri, Kansas City. He was awarded Dean's International Computing and Engineering Award in his Master's program. With Open Methods scholarship he worked as a Software Intern during his Master's program.

He is now working as a Systems Engineer for Charter Communications, St. Louis. His experience with software design and implementation inspired him to develop SoCCS framework.