TEXT BASED DIALOG SYSTEM

Nabila Naushin
ID: 05201002

Zaki Mohammad Abdul Kareem
ID: 05201007

Rasel Chowdhury
ID: 05201006

Department of Computer Science and Engineering
December 2009

BRAC University, Dhaka, Bangladesh

DECLARATION

We hereby declare that this thesis is based on the results found by ourselves. Materials of work found by other researcher are mentioned by reference. This Thesis, neither in whole nor in part, has been previously submitted for any degree.

Signature of Supervisor

Dr. Mumit Khan

Signature of Author

Nabila Naushin

Zaki Mohammad Abdul Kareem

Rasel Chowdhury

ACKNOWLEDGMENTS

Firstly we would like to thank our advisor Dr. Mumit Khan for giving us the golden opportunity to work on this project and also for providing his invaluable support and guidance throughout the time. The brief discussions that we had with him were very enlightening and for that we shall be forever grateful.

Secondly we thank our lecturer Mr. Sarwar Alam for sharing his radiant ideas and for suggesting realistic approaches to tackle our problems.

Thirdly we would like to thank both our junior lecturer Mr. Annajiat Alim Rasel and friend E. M. Y. Tahin Rahman: the former for the unconditional technical assistance we received to run the system and the latter for providing his work on Bangla transliteration.

Lastly we would like to take the opportunity to thank the founders of the Olympus architecture as well as the contributors to this end. Their endeavors did not go in vain as many researchers are using their free tools to study and develop dialog systems. We sincerely hope that in the near future more work will be done on developing, documenting and proposing their systems to the greater community for a better tomorrow.

ABSTRACT

Amidst the growing popularity of complex dialog systems it has become quite difficult to grasp the implementation of such systems. The intention of this document is to introduce the designers and researchers new to this field to tools and design techniques which may be used to develop dialog systems.

We looked into systems such as OpenEphyra, Trindikit and Ariadne. We then shifted our gaze to MyBus, a text based dialog system which makes use of Olympus architecture and concluded that it would stand very nicely for our cause.

.

.**Keywords:** Dialog System, Olympus Architecture, MyBus, Bangla Transliteration

TABLE OF CONTENT

LIST OF FIGURES

CHAPTER I: INTRODUCTION

This era has seen a lot of technological improvements throughout the world in all kinds of area that we can conceive of. The technology though improved in leaps and bounds is still in its rudimentary stage in areas of natural language processing for our native language - Bangla.

It is our desire to facilitate matters for our country folk and it invariably leads us to think of the communication issues the common people face in their everyday life. It is worth noting that over 80% of the population hardly understand English and therefore cannot make use of the technology that is at their disposal.

That majority of the public is not competent in English is a reason good enough for us to develop a dialog system that will enable communication via our mother language. This system can be extended to a dialogue system for any given field. For clarity let us assume that the dialogue system is for a shop. Customers can query the system for answers related to their needs [price, model, etc]. The system can use Bangla text or speech as a means of reply.

The study of dialog systems is commonly considered a branch of human-computer interaction, although its origins are generally rooted in the automatic speech recognition community. Current trends are putting more research emphasis on aspects of psychology and linguistics [1]. Dialog system can of various categories. It can be both spoken and text-based.

Our intention was to develop a dialog system which converses with the user in Bangla. The input as well as the output were to be in speech form. We modified MyBus, a text-based dialogue system which requires Olympus architecture. This system oversees bus scheduling. This modified version enables the user to input text in Bangla transliterated form to interact with the machine. It replies back to the client as both text and speech. The output text is

in Bangla transliterated form and the speech is made as close as possible to Bangla using Microsoft SAPI.

In the following chapters we show how we managed to implement such a system.

1.1 Background Study

To develop AmarBus (Bangla Version of MyBus) we had to do a lot of research on the fields of dialog systems and natural language processing. For that purpose we studied many books, papers and websites. We will mention a few of them in this section.

Speech and Language Processing by Daniel Jurafsky has introduced us with the preliminary ideas of Natural Language processing and Dialog Systems [2]. This book gave us the basic concepts of how a dialog system works. We also went through manuals and websites of some of the available dialog system frameworks such as Trindikit, OpenEphyra and Ariadne [3, 4, 5, 6].

Then we came across Olympus (Bohus 2007), an open source architecture for spoken dialog systems. It is used to Implement and test conversational agents on full systems, without having to build them from scratch [7,8,9].

We also went through papers of Let's Go! telephone based dialog system that works using the Olympus Architecture. MyBus is the simpler version of this system. This system is mainly designed mainly for elderly and non-speaking people [11, 12]. It is also analyzed (Raux 2006) how Let's Go! performed as a practical system [10].

MyBus tutorial [13] gave us the idea of how MyBus is running. Manuals of Olympus also helped in that manner.

For Bangla transliteration resources of Center for Research on Bangla Language Processing helped us a lot [17].

CHAPTER II: DIALOG SYSTEMS

"Imagine a machine trying to emulate the human pattern of thinking and conversing with a human being on some designated topic" – such a system is a dialog system.

2.1 Definition

A dialog system can be interpreted as a computer system that can converse with a human using a coherent structure. Dialog systems can make use of text, speech, graphics, haptics, gestures and other types for communication on both the input and output channel. [1].

Dialog systems have become quite popular in this age. Classic adventure games like Escape from Monkey Island and Adventures of Sherlock Holmes are examples of systems which make exhaustive use of dialog foreplay.

2.2 Desired characteristics of Dialog System

Not all dialog systems make it big. Hence it's necessary to identify what makes a dialog system a satisfactory system. Below we discuss some of the desirable traits of a dialog system framework [7].

Open: Open systems enable researchers and designers to study the codes of each every component and then tune them according to their needs. The other benefit of having an open system is that the community on using and modifying the codes will engender overall growth of the system as more and more individuals will get involved for the development.

Transparent / Analytic: Open source codes are a must but that alone will not make sure that the system is transparent. It will only promote transparency. It is important that researchers are able to analyze and interpret the behavior of various components. On top of that researches should be given tools for inspection of data within the components.

Flexible: The framework should be flexible and have the capacity to put up with a wide range of programs and research elements.

Modular / Reusable: It is imperative that the system addresses the concept of reusability. This is because in this age when developing a system one needs to think of the future implications. A well developed application-independent system with flexible components and rich interfaces is a must.

Scalable: The designer has to keep in mind that flexibility or transparency or both may compromise scalability. A framework which follows simple well established approaches will most likely promote scalability – development of large-scale systems. But then again one will not know the implications of flexibility and transparency from a toy system. To understand its impact one has to move from toy to large-scale applications.

We have chosen Olympus architecture as it addresses the aforementioned characteristics.

CHAPTER III: OLYMPUS

3.1 Introduction to Olympus

The Olympus architecture is an architecture built for supporting the creation and deployment of spoken dialog systems. It was created at Carnegie Mellon University [CMU]. Building a dialog system from scratch can be very tedious and time consuming. That is why the Olympus architecture took birth. Its main purpose is to serve the researchers and students and not build such systems from scratch [7, 8].

Olympus is a free framework designed for the study of dialog systems. Previously we have noted that this system complies with all the major requirements of a good dialog system. It is open, transparent, flexible, modular and scalable. This means the system is well suited for facilitating the development of large-scale and real-world systems. It can aid in the growth of both text based as well as spoken dialog systems.

Academia pursues long-term scientific research goals, which are not related to immediate economic returns or customer populations. As a result, academic groups are free to explore a larger variety of research questions, even with a high risk of failure or a lack of immediate payoff. These groups also engage in a more open exchange of ideas and results. However, building spoken language interfaces requires significant investments that are sometimes beyond the scope of academic researchers. As a consequence, research in academia is mostly limited to conducting research with toy systems. In turn, this raises questions on the validity of the results and lowers the research impact [7].

In an effort to address this problem Olympus was developed. It is a freely available framework for developing and deploying conversational spoken language interfaces. Olympus is an open, transparent, flexible, modular, and scalable architecture.

Many spoken language systems having different domains and interaction modes have been generated and deployed using this architecture. Currently researches on diverse aspects of spoken language interaction are being carried out.

To bridge the gap between industry and academia Olympus and other similar frameworks will be needed and should be promoted. The advantage of having such frameworks is that it lowers the cost of entry for research on pragmatic conversing agents. They also promote technology transfer through the reuse of components, and support direct comparisons between systems and technologies [7].

3.2 Olympus Architecture

In Olympus the input from the user is captured by a speech recognizer and passed to a module responsible for dissecting the meaning out of the input. Phoenix is the parser and is responsible for this. the dialog manager, in this case RavenClaw gets to decide the appropriate action or response for the user. Olympus may have more than just one backend agent. Therefore its possible for the dialog manager to communicate with more than one backend agent. Once the appropriate action or reply is determined, the output from RavenClaw is sent to a natural language generator [NLG] which is responsible for converting actions into text. This is then passed on to an audio system via a synthesis component. Olympus is a classical example of pipelined spoken dialog architecture.

The image below provides a detailed view of the various system components.

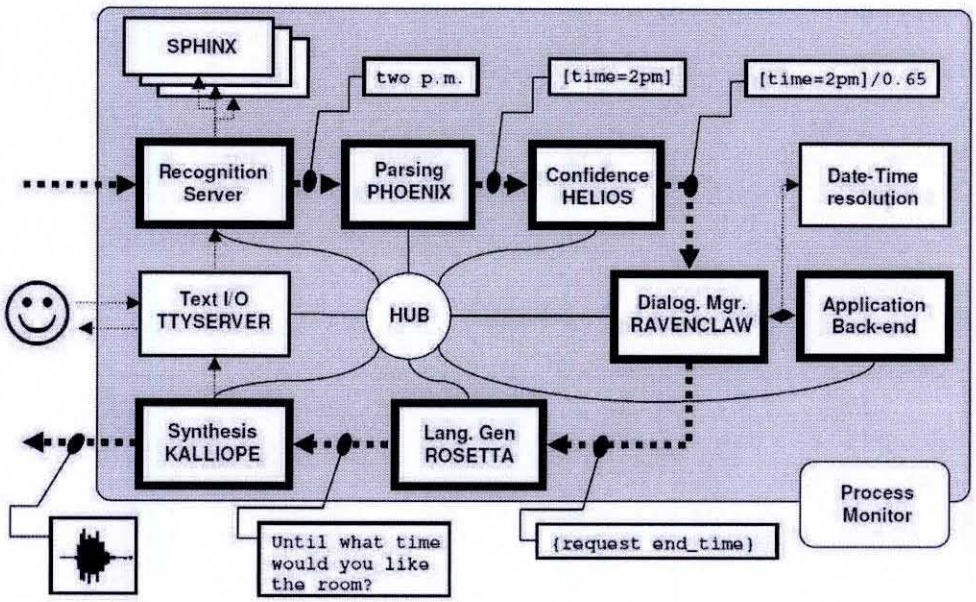


Figure1. The Olympus dialog system architecture

3.2.1 Overview of the components involved

The main tasks of Olympus components involve [9]:

Recognition

The audio input stream is captured by the soundcard, which is then forwarded to multiple engines of Sphinx-II. The multiple decoder engines of Sphinx can be configured to work in parallel. Each of them forwards the results to the Phoenix parser the module responsible for language understanding. Sphinx-III has been developed but due to real-world constraints is not being used.

Language Understanding

Phoenix is the parser which means its responsible for finding the language understanding. It draws the semantic meanings of the input and forwards it to the dialog manager via Helios. The grammars are manually constructed.

Dialog Manager

The dialog manager is responsible for selecting the appropriate action or response. In this case the dialog manager is RavenCalw. It receives semantic inputs and after processing and getting the answer it sends the reply to natural language generation module. Dialog manager has the capacity to interact with multiple back-ends.

Domain Reasoning

Dialog manager has the capacity to interact with multiple back-ends such as databases. Databases hold domain information.

Language Generation

In this case Rosetta is the language generator. It takes semantic output from the dialog manager and produces the corresponding natural text based on templates.

Synthesis

Clients for several synthesis engines (e.g. Festival, Theta, Swift) are available and can be used in the Olympus dialog management framework.

Text I/O

For debugging an I/O text component for the system is also provided.

These various components are interconnected by the Galaxy message-passing communication infrastructure. Its also simply known as the hub. Galaxy

uses a central hub and a set of rules for relaying messages from one component to the other.

3.2.2 Systems

As mentioned before Olympus has been used on many occasions to build spoken dialog systems of different domains and interaction modes. It can aid in deployment of both text based as well as spoken dialog systems. We will discuss about two example systems, which used this infrastructure: Let's Go! and MyBus.

3.2.2.1 Let's Go!

Let's Go was built mainly to aid the elderly and non-natives. It is a telephone based bus-scheduling system and is still used by the Pittsburgh population since March 2005. It was created as a test bed for spoken dialog agent with extreme user population. [11, 12].

.

3.2.2.2 MyBus

MyBus an Olympus example system is a simplified version of Let's Go and is the one we will use to demonstrate the designing tools and methods. [13] MyBus is a simple bus schedule information system just like Let's Go and uses Olympus architecture. The input to MyBus is text-based and output is both text-based and spoken. It retrieves bus route information according to user queries.

Below is the sample conversation of MyBus:

S: Welcome to MyBus. Where are you leaving from? U: DOWNTOWN

S: Where are you going?

U: THE AIRPORT

S: Let me check that for you.

There is a 28X leaving DOWNTOWN at 4:20 p.m. It will arrive at THE AIRPORT at 4:56 p.m.

You can say, when is the next bus, when is the previous bus, start a new query, or goodbye.

U: GOODBYE

S: Thank you for using MyBus. Goodbye!

Figure 2: Sample Conversation of MyBus

CHAPTER IV: MYBUS

4.1 Methodology

In this chapter we briefly discuss the steps followed to develop MyBus. We start with decomposition of the system as this will simplify the problem.

4.1.1 Decomposition of the System

For simplicity to achieve the task the system is decomposed into subtasks as shown below [13]:

1. Greeting the user
2. Responding to the user's queries
3. Wishing goodbye to the user

The middle task of "responding to the user's queries" can be further divided into three subtasks [13]:

1. Getting the user's request/query
2. Getting the information pertaining to the query
3. Displaying the result (if any) to the user

Each of the subtasks can be further decomposed into smaller tasks. It will be easier to visualize the structure if seen as a tree. This sort of tree is called the "Task Tree". Each node of the tree represents a subtask. From the parent nodes, we can get child nodes. This process is recursive. The designer can keep on dividing the tasks to smaller tasks to simply the system. The nodes of the trees are know as "agents" as they are responsible for performing a certain sub-dialogue. Below we show the Dialog Task Tree of MyBus:

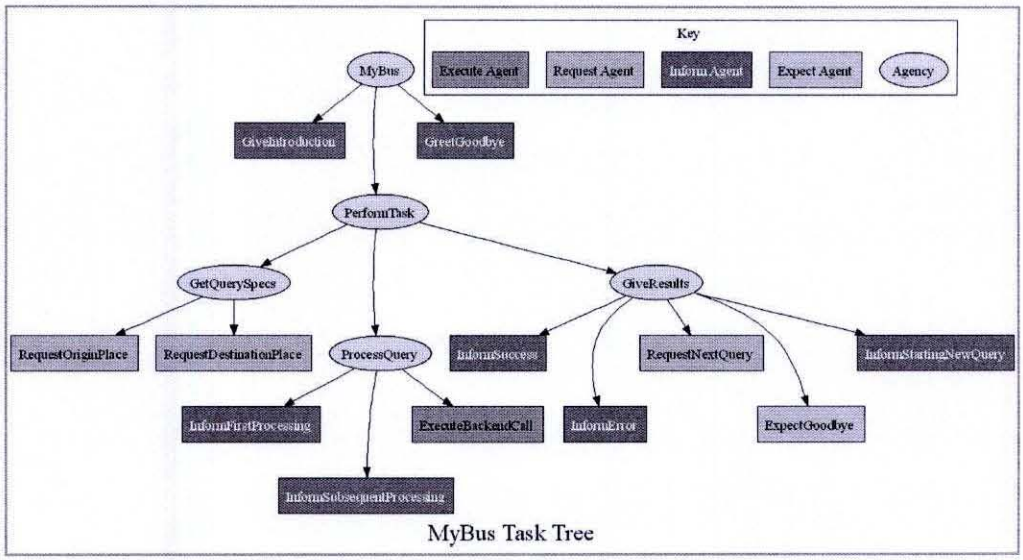


Figure 3: Task tree for the MyBus system

As seen from the tree we have five types of nodes or agents. One type of internal node and four types of leaf nodes [13]:

Agencies

They are the internal nodes and represent subtasks, which can be further subdivided into smaller tasks.

Inform Agents

As the name indicates, these agents are responsible for informing the user or provide data related to his queries.

Request Agents

They represent the atomic behavior of asking questions to the user and understanding the answers.

Execute Agents

They are not involved in the conversation that takes place between the user and machine. They simply execute actions or queries demanded by the user.

Expect Agents

Unlike the execute agent, expect agents do not perform any action but rather capture specific concepts that the user might say or mean.

4.1.2 Traversing the Tree

For traversing depth-first, left-right traversal is used. This means that, when traversing the tree [13]:

1. Agencies immediately traverse towards their child agents
2. The children agents are traversed from left to right

The first thing the system is going to do is execute GiveIntroduction:

S: Welcome to MyBus.

Next, it will traverse PerformTask to GetQuerySpecs to RequestDeparturePlace:

S: Where are you leaving from? U: DOWNTOWN

Once the answer is obtained from the user, the system will move onto RequestArrivalPlace:

S: Where are you going? U: THE AIRPORT

Since all the children of GetQuerySpecs have been traversed, the system will go to ProcessQuery, which will lead to InformFirstProcessing:

S: Let me get that for you.

The role of this message is to make the user wait while the system is searching the database to retrieve the results. If we went on like this, it would go to InformSubsequentProcessing. We do not want the system to execute InformFirstProcessing and InformSubsequentProcessing sequentially. Instead, for the first user query, we would like the system to execute InformFirstProcessing, skip InformSubsequentProcessing, and execute ExecuteBackendCall. For subsequent user queries, the system should skip InformFirstProcessing, execute InformSubsequentProcessing and execute ExecuteBackendCall. This is therefore breaking the basic "depth-first left-right" traversal rule. Fortunately, this can be done in RavenClaw using preconditions. Agent preconditions define when a given agent should be executed and when it should be skipped. So assuming we have completed the backend call, following our top-down rule, the next agent to be executed is InformSuccess:

S: There is a 28X leaving DOWNTOWN at 4:20 p.m. It will arrive at THE AIRPORT at 4:56 p.m.

Top-down traversal tells us that the system now executes InformError. Again both InformSuccess and InformError for a single database query will not be executed. Either the system was able to retrieve results for the user or for some reason the system could not get an appropriate answer to the query. Once either of the two Inform Agents has been executed, traversal continues to RequestNextQuery:

S: You can say, when is the next bus, when is the previous bus, start a new query, or goodbye.
U: WHEN IS THE NEXT BUS

The next agent to be executed is ExpectGoodbye. Then, according to the left-right rule, the system should execute InformStartingNewQuery, and go to the next terminal agent on the right, which is GreetGoodbye. However, this is not the intended behavior. Instead, the desire is to go back and look at the database for the time of the next and inform the user of the new results. Then execute InformSubsequentProcessing:

S: Okay.

The new result is obtained from the backend, and InformSuccess is executed:

S: There is a 28X leaving DOWNTOWN at 7:03 p.m. It will arrive at THE AIRPORT at 7:37 p.m.

InformError is again skipped because of preconditions and re-execute RequestNextQuery:

S: You can say, when is the next bus, when is the previous bus, start a new query, or goodbye.
U: GOODBYE

As we do not consider exiting as a next query, it is not understood by RequestNextQuery. Instead, "GOOODBYE" is understood by the ExpectGoodbye attached just next to RequestNextQuery. Accordingly, the GiveResults agency completes its execution (note that the precondition for

InformStartingNewQuery being still false), so does PerformTask (normally, agencies complete their execution once all their children have been traversed and have not been reopened). We continue our left-right traversal to execute GreetGoodbye:

```
S: Thank you for using MyBus. Goodbye!
```

4.1.3 Grammar

The role of the grammar is to take input from the user and dissect the real meaning from it. Words, which are not useful for understanding the semantics of the query, are discarded. The user input does not have to be grammatically correct in the linguistic sense. This module [Phoenix parser] is responsible for matching words found in the grammar database. The grammar helps to eliminate ambiguity and redundancy. [13].

The role of the grammar is to enlist all the possible sentences the system can understand. This is not done by listing all the sentences one by one. It simply directs the combination of words that the system might expect from the user. If the words match, then the user query is classified as a valid query. Therefore, grammar formalisms such as the one used by Phoenix, offer a way of representing thousands or millions of sentences in a compact, readable way. [7]. The grammar for MyBus system is found in the file "Resources/Grammar/MyBusTask.gra":

```
[Place]
  (carnegie mellon university)
  (downtown)
  (robinson towne center)
  (the airport)
```

```
(south hills junction)
(mount oliver)
(the south side)
(oakland)
(bloomfield)
(polish hill)
(the strip district)
(the north side)
;
```

Figure 4: List of places

The following file describes the grammar in the forms definition file: "Resources/Grammar/MyBusTask.forms". Basically, this file lists all the top level slots and groups them by function.

```
[NextBus]
   (*WHEN_IS *the next *BUS)
   (*WHEN_IS *the BUS after that *BUS)

 WHEN_IS
   (when is)
   (when's)

 BUS
   (bus)
   (one)
 ;
```

Figure 5: Grammar for defining

4.1.4 Task Specification Language

Olympus does not have a GUI for creating the tree structure. Hence the inclusion of RavenClaw Task Specification Language [RCTSL]. It's a language which is explicitly used to define the agents and the relations between them.

RCTSL is a set of C++ macros, that once compiled, generate an executable that will be the dialogue manager (DM). The big advantage of using C or C++ code in parts of the task specification, gives the full power of a complete programming language when the need arises, which inevitably happens once we start building realistically complex systems.

A couple of examples of using the language specification language is provided in Appendix C and if the reader wishes to see all of the codes along with explanation then he/she can visit Tutorial 1 of Olympus Dialog System Framework Website [14].

CHAPTER V: IMPLEMENTATION

Before we discuss about the implementation we briefly outline the requirements necessary to design and run the system [15]:

- Visual Studio 2005 (SP1)
- Microsoft Speech SDK 5.1 [SAPI]
- ActivePerl
- Subversion client (TortoiseSVN, SmartSVN, etc.)

A download of Olympus architecture is available in the main site. After installation of Olympus the reader can download any of the example systems provided in there.

5.1 Methodology of AmarBus

To interact with the System in Bangla we had to modify some files included in MyBus. The files are listed below and the modification of these files are shown in Appendix B.

- Inform.pm: as the name implies is responsible of notifying the user or providing the user with information pertaining to his queries.

- Request.pm: asks the user questions related to his queries or wants.

- MyBusTask.gra: this is where the grammar is specified.

- MyBus.schedules.txt: holds the route information

CHAPTER VI: DISCUSSION

6.1 Problems Faced

The complexity of the system was enormous. It was painstaking to sort out which files were called and when. We could not understand the complete interaction between the system components and the files involved in processing were far too many. More time will be required to study and understand the importance of each and every file used by the system.

JavaTTY although provided does not connect to the session manger. The JavaTTY.jar file does not run when called directly or indirectly. Its highly possible that certain files were missing and the solution could not be identified. We also noted that JavaTTY was not called by the Process Monitor.

The example system Roomline although provided in the site does not run when installed. Also Roomline.zip file uploaded in the site cannot be downloaded due to some internal problem. In the site it was recommended to download. Olympus from [http://trac.speech.cs.cmu.edu/repos/olympus/branches/2.2] as it's the stable version. But we discovered that Roomline calls files which were not available in that version of Olympus. On checking Olympus in the Trunk Folder [http://trac.speech.cs.cmu.edu/repos/olympus/trunk] it was noted that this unstable version had the functions Roomline required. On installing these files to the stable Olympus we were still unable to run RoomLine.

The Unicode representation of Bangla in MyBus console is not supported. Hence we were not able to display the Bangla characters. To counter this problem we tried using JNI. It calls java funtions from C++ modules but the implementation became too tedious and procedure becomes too clumsy. The

data was to be passed to and fro between java and C++ which we do not desire in the long run.

For people who are interested to study the problems - we strongly recommend that they should go through the startlist which is called by the Process Monitor. This should give them a head-start to identify the processes involved. In addition, they need to see SystemRun.bat and SystemRunTTY.bat along with the configuration files. Best would be to go through all the files that come with Olympus and the example system.

6.2 Future Plans

- Intention of using JavaTTY provided in Olympus as this interface will provide an easier means for Bangla Unicode character representation.

- Try to make the backend more flexible as in the case of RoomLine. The backend of MyBus has zero flexibility and therefore not recommended. For this reason it might be necessary to initially find a way to make the RoomLine project work. From there we will be able to explore the potentials of using the backend.

- Lastly we plan to use Bangla phones for speech output. This we desire because currently the speech output generates utterances not proper for Bangla pronunciation. Bangla phones will help us to emulate the Bangla accent.

CHAPTER VII: CONCLUSION

We chose Olympus as it has all the desirable traits of a good dialog system framework. It is open, transparent, flexible, modular and supports scalability. We concluded that Olympus is an excellent framework for developing and deploying systems which could be text based or spoken dialog sytem or both.

Our goal was to develop a dialog system which would converse with the user in Bangla. Unfortunately due to lack of time to study and improve our system we ended up with a prototype which could take input in Bangla transliterated form and could answer both in text and speech. The output text was in Bangla transliterated form and the speech as close as possible to Bangla with the aid of Microsoft SAPI.

The main problem we faced was in presenting Bangla characters. The interface was made in C++ and did not support Unicode character representation. More time will be needed to understand the components of Olympus and their respective functionalities.

In future we plan to deploy the system using Bangla phones so as to emulate the Bangla accent. We also would encourage the future developers to make use of JavaTTY a free component provided in Olympus. It would enable them to represent Unicode characters more freely. The backend of MyBus has zero flexibility. It would be desirable to make use of a more flexible backend such as the one provided in RoomLine, an example system of Olympus.

We sincerely hope that in the near future more work will be done on developing, documenting and proposing such systems to the greater community for a better tomorrow.

REFERENCES

1. http://en.wikipedia.org/wiki/Dialog_system
2. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, D. Jurafsky and J. Martin
3. Trindikit Website: http://www.ling.gu.se/projekt/trindi/trindikit/
4. OpenEphyra Website: http://mu.lti.cs.cmu.edu/trac/Ephyra/wiki/OpenEphyra
5. Ariadne Website: http://www.opendialog.org/
6. Matthias Denecke, *Rapid Prototyping For Spoken Dialogue Systems*-The 19th International Conference on Computational Linguistics 2002, COLING 2002, Taipei, China, 13. December 2009
7. Bohus, D., Raux, A., Harris, T., Eskenazi, M., and Rudnicky, A. (2007) - *Olympus: an open-source framework for conversational spoken language interface research*, to appear in HLT-NAACL 2007 workshop on Bridging the Gap: Academic and Industrial Research in Dialog Technology, Rochester, NY
8. Olympus Website: http://wiki.speech.cs.cmu.edu/olympus/index.php/Olympus
9. RavenClaw Website: http://www.cs.cmu.edu/~dbohus/ravenclaw-olympus/what_is_olympus.html
10. Raux, A., Bohus, D., Langner, B., Black, A., and Eskenazi, M. (2006) *Doing Research on a Deployed Spoken Dialogue System: One Year of Let's Go! Experience*, Interspeech 2006 - ICSLP, Pittsburgh, PA.
11. Raux, A., Langner, B., Black, A., Eskenazi, M. (2005) *Let's Go Public! Taking a Spoken Dialog System to the Real World*, Interspeech 2005 (Eurospeech), Lisbon, Portugal.
12. Antoine Raux; Brian Langner; Alan W. Black; Maxine Eskenazi, (2008), *Tutorial: Building Practical Spoken Dialog Systems, 46th Annual Meeting of the Association for Computational Linguistics*: Human Language Teshnologies Columbus, OH.

13. http://wiki.speech.cs.cmu.edu/olympus/index.php/Tutorial_1

14. http://wiki.speech.cs.cmu.edu/olympus/index.php/Tutorial_1#Writing_the_task_specification

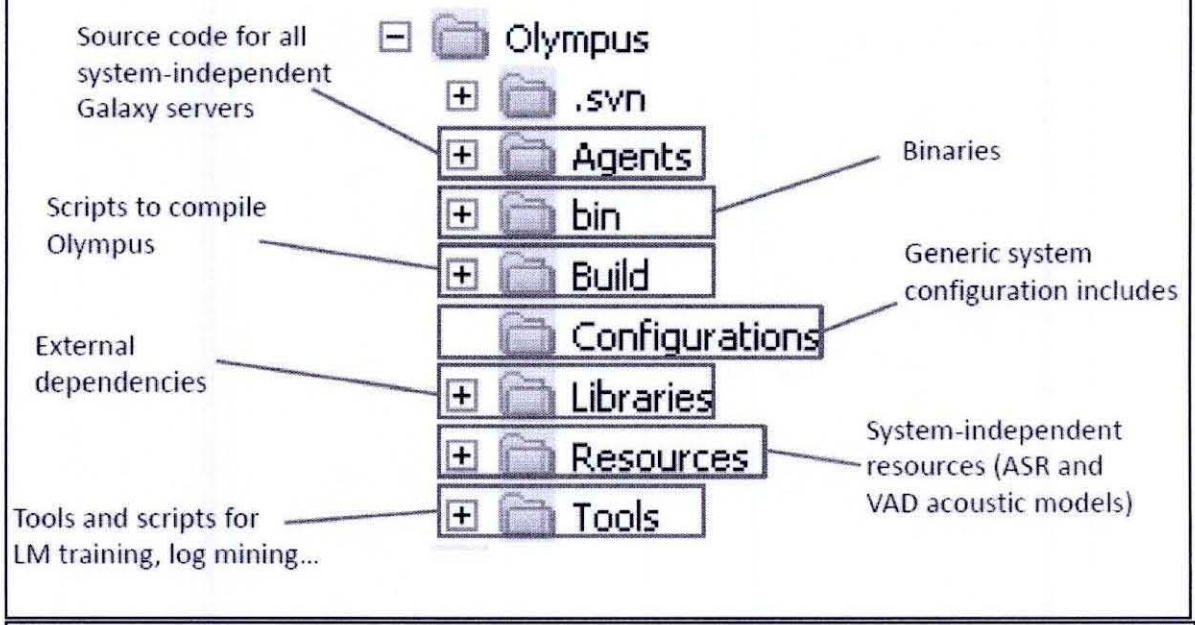
15. Spoken Dialog Systems-Advanced Concepts in Dialog: http://www.speech.cs.cmu.edu/15-492/slides/sds_advanced.pdf

16. Naushad UzZaman, Arnab Zaheen and Mumit Khan, *A Comprehensive Roman (English) to Bangla Transliteration Scheme*, International Conference on Computer Processing on Bangla (ICCPB-2006), Dhaka, Bangladesh, 17 February, 2006.

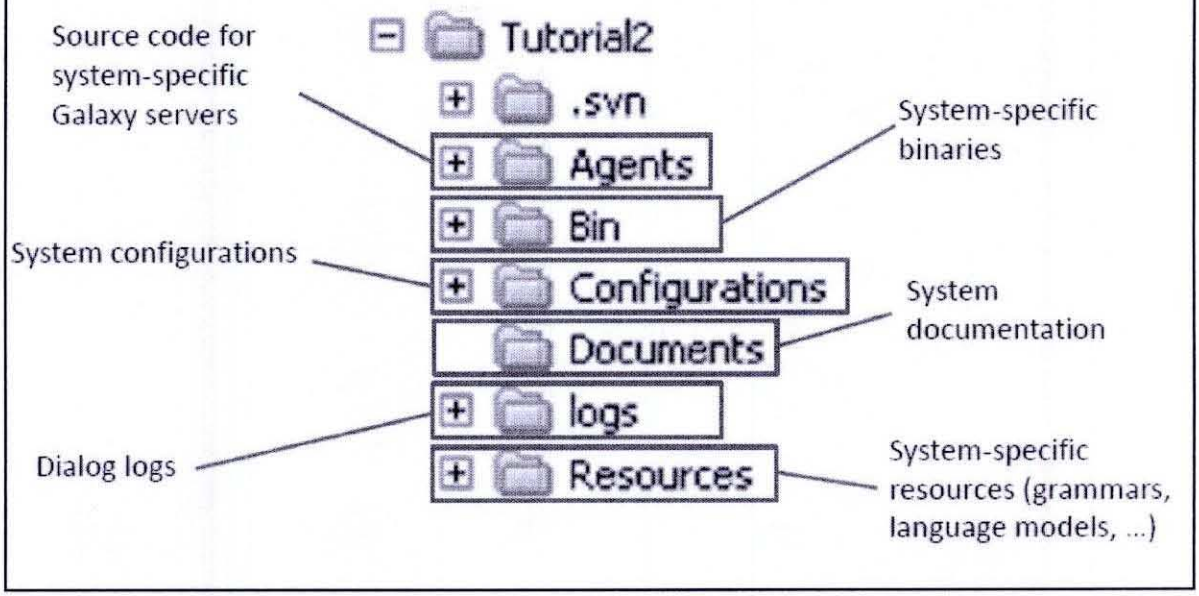
APPENDIX A

Olympus and MyBus Directory Structure

Olympus Core Directory Structure



System Directory Structure



APPENDIX B

Code Snippets of AmarBus

File name: MyBus\Agents\Rosetta\Templates\Inform.pm

```
These are the INFORM acts for MyBus

  ##################################################################
  # Greetings
  ##################################################################

  # welcome to the system
  "welcome" => "Aamar Bus ey aapnaakey shagotom.",

  # quitting
  "goodbye" =>
   "Aamar Bus babohaar kaurar jonno apnake dhonnobaad.",

  # looking up database (to announce a delay)
  "looking_up_database_first" => "Ki-chu-khon opekkha korun. ami dekhchi.",
  "looking_up_database_subsequent" => "Ki-chu-khon opekkha korun.",

  # inform the user we're starting a new query
  "starting_new_query" => "Ashun prothom theke shuru kori.",

  ##################################################################
  # Query results
  ##################################################################

  "result" => sub {

          my %args = @_;

                  my $dep_time = &convertTime($args{"result.departure_time"});
                  my $arr_time = &convertTime($args{"result.arrival_time"});
                  return "Route nong <result.route> bus <origin_place> thekey
<result.departure_time $dep_time> e chere jaabey. ".
                   "Sheita <destination_place> e pouchaabey <result.arrival_time $arr_time> e.";
            },
```

File name: MyBus\Agents\Rosetta\Templates\Request.pm

```
These are the REQUEST acts for MyBus
#########################################################################
#
# Task-specific requests
#
#########################################################################
#

 # request nothing (just wait for the user to say something)
 "nothing" => "...",

 # ask if the user wants to make another query
 "next_query" => "Apni jigesh kortaey paaren, porer bus kaukhon, aager bus kaukhon ".
                 ", prothom theikey shuru koro, othoba goodbye.",

 # request the departure place
 "origin_place" => "Aapni kothha theikey uuthhben?",

 # request the arrival place
 "destination_place" => "Aapni kothhay jeitey chaachchen?",

};
```

File name: MyBus\Resources\Grammar\GRAMMAR\ MyBusTask.gra

```
#############################################################
#
#
#      TASK-SPECIFIC  GRAMMAR
#
```

```
#
##################################################################

##################################################################
# TASK-DEPENDENT TOP-LEVEL SLOTS
##################################################################

[Place]
        (carnegie mellon *university)
        (cmu)
        (downtown)
        (robinson towne center)
        (*the airport)
        (south hills junction)
        (mount oliver)
        (*the south side)
        (oakland)
        (bloomfield)
        (polish hill)
        (*the strip *district)
        (*the north side)
;

[NextBus]
        (PORER BUS *KOKHON *ashbe)

PORER
        (porer)
        (erporer)
KOKHON
        (kokhon)
        (konshomoy)
        (koitai)

BUS
        (bus)
```

```
        (ta)
;

[PreviousBus]
        (AAGER BUS *KOKHON *AASHBEY)

AAGER
        (aager)
        (ager)

KOKHON
        (kokhon)
        (konshomoy)
        (koitai)
        (kaukhon)

AASHBEY
        (aashbey)
        (ashbe)
        (aashbe)
        (chilo)
        (chilo)
        (silo)
BUS
        (bus)
        (ta)
;

##################################################################
# START-OVER GRAMMAR
##################################################################

[StartOver]
        (*ABAR *PROTHHOM *THHEIKEY *SHURU *KORO)

ABAR
```

```
        (abar)
        (aabar)
        (aabaar)
        (abaar)
        (abaro)
        (aabaro)
        (aabaaro)

PROTHHOM
        (prothom)
        (prothhom)
        (notun)
        (nutun)

THHEIKEI
        (thheikei)
        (theke)
        (theikey)
        (kore)

SHURU
        (shuru)
        (arombho)

KORO
        (koro)
        (kauro)
        (korun)
;
```

APPENDIX C

RavenClaw Task Specification Language

Agent Specification

```
// /MyBus
 DEFINE_AGENCY( CMyBus,

  IS_MAIN_TOPIC()

  DEFINE_SUBAGENTS(
   SUBAGENT(GiveIntroduction, CGiveIntroduction, "")
   SUBAGENT(PerformTask, CPerformTask, "")
   SUBAGENT(GreetGoodbye, CGreetGoodbye, "")
  )

 )
```

GiveIntroduction:

```
// /MyBus/GiveIntroduction
 DEFINE_INFORM_AGENT( CGiveIntroduction,
  PROMPT("inform welcome")
 )
```

Agent:

```
// /MyBus/PerformTask
 DEFINE_AGENCY( CPerformTask,

  DEFINE_CONCEPTS(
   INT_USER_CONCEPT(query_type, "")
   STRING_USER_CONCEPT(origin_place, "")
   STRING_USER_CONCEPT(destination_place, "")

   CUSTOM_SYSTEM_CONCEPT(result, CResultConcept)
   CUSTOM_SYSTEM_CONCEPT(new_result, CResultConcept)
  )
```

```
  DEFINE_SUBAGENTS(
    SUBAGENT(GetQuerySpecs, CGetQuerySpecs, "")
    SUBAGENT(ProcessQuery, CProcessQuery, "")
    SUBAGENT(GiveResults, CGiveResults, "")
  )

)
```

Concept Type Specification

```
DEFINE_FRAME_CONCEPT_TYPE( CResultConcept,
  ITEMS(
    INT_ITEM(failed)
    STRING_ITEM(route)
    INT_ITEM(departure_time)
    INT_ITEM(arrival_time)
  )
)
```

GetQuerySpecs

```
// /MyBus/PerformTask/GetQuerySpecs/RequestOriginPlace
 DEFINE_REQUEST_AGENT( CRequestOriginPlace,

  PROMPT("request origin_place")
  REQUEST_CONCEPT(origin_place)
  GRAMMAR_MAPPING("![Place]")

)
```

ProcessQuery

```
// /MyBus/PerformTask/ProcessQuery
DEFINE_AGENCY( CProcessQuery,

  DEFINE_SUBAGENTS(
    SUBAGENT(InformFirstProcessing, CInformFirstProcessing, "")
    SUBAGENT(InformSubsequentProcessing, CInformSubsequentProcessing, "")
    SUBAGENT(ExecuteBackendCall, CExecuteBackendCall, "")
  )

  SUCCEEDS_WHEN(
    SUCCEEDED(ExecuteBackendCall)
  )
)
```

Inform Agents

```
// /MyBus/PerformTask/ProcessQuery/InformFirstProcessing
DEFINE_INFORM_AGENT( CInformFirstProcessing,
  PRECONDITION(!AVAILABLE(result))
  PROMPT("inform looking_up_database_first")
)

// /MyBus/PerformTask/ProcessQuery/InformSubsequentProcessing
DEFINE_INFORM_AGENT( CInformSubsequentProcessing,
  PRECONDITION(AVAILABLE(result))
  PROMPT("inform looking_up_database_subsequent")
)
```

Execute Agent

```
// /MyBus/PerformTask/ProcessQuery/ExecuteBackendCall
 DEFINE_EXECUTE_AGENT( CExecuteBackendCall,
  EXECUTE(
   if (!AVAILABLE(query_type)) {
    C("query_type") = NQ_BUS_AFTER_THAT;
   }

   // call on the galaxy stub agent to execute that particular call
   pTrafficManager->Call(this, "gal_be.launch_query <query_type "
                         "<origin_place <destination_place "
                         "<result >new_result");

   C("result") = C("new_result");
  )
 )
```

GiveResults agency

```
/ /MyBus/PerformTask/GiveResults
 DEFINE_AGENCY( CGiveResults,

  DEFINE_CONCEPTS(
   INT_USER_CONCEPT( next_query, "")
   BOOL_USER_CONCEPT( goodbye, "")
  )
  DEFINE_SUBAGENTS(
   SUBAGENT(InformSuccess, CInformSuccess, "")
   SUBAGENT(InformError, CInformError, "")
   SUBAGENT(RequestNextQuery, CRequestNextQuery, "")
   SUBAGENT(ExpectGoodbye, CExpectGoodbye, "")
   SUBAGENT(InformStartingNewQuery, CInformStartingNewQuery, "")
  )
  SUCCEEDS_WHEN(
   ((int)C("next_query") == NQ_BUS_AFTER_THAT) ||
   ((int)C("next_query") == NQ_BUS_BEFORE_THAT) ||
```

```
  SUCCEEDED(InformStartingNewQuery) ||
  IS_TRUE(goodbye)
 )
 ON_COMPLETION(
  if (((int)C("next_query") == NQ_BUS_AFTER_THAT) ||
     ((int)C("next_query") == NQ_BUS_BEFORE_THAT)) {
   A("..").ReOpenTopic();
   C("query_type") = (int)C("next_query");
   C("next_query").Clear();
  } else if ((int)C("next_query") == NQ_NEW_REQUEST) {
   A("/MyBus/PerformTask").Reset();
  }
 ))
```

RequestNextQuery

```
// /MyBus/PerformTask/GiveResults/RequestNextQuery
 DEFINE_REQUEST_AGENT( CRequestNextQuery,
   REQUEST_CONCEPT(next_query)
   PROMPT("request next_query")
   GRAMMAR_MAPPING("![StartOver]>1, "
                   "![NextBus]>2, "
                   "![PreviousBus]>3")
 )
```

Expect Agent:

```
// /MyBus/PerformTask/GiveResults/ExpectGoodbye
 DEFINE_EXPECT_AGENT( CExpectGoodbye,
   EXPECT_CONCEPT( goodbye)
   GRAMMAR_MAPPING("@(../RequestNextQuery)[Quit]>true")
 )
```

Inform agent:

```
DECLARE_AGENTS(
   DECLARE_AGENT(CMyBus)
     DECLARE_AGENT(CGiveIntroduction)
     DECLARE_AGENT(CPerformTask)
       DECLARE_AGENT(CGetQuerySpecs)
         DECLARE_AGENT(CRequestOriginPlace)
         DECLARE_AGENT(CRequestDestinationPlace)
       DECLARE_AGENT(CProcessQuery)
         DECLARE_AGENT(CInformFirstProcessing)
         DECLARE_AGENT(CInformSubsequentProcessing)
         DECLARE_AGENT(CExecuteBackendCall)
       DECLARE_AGENT(CGiveResults)
         DECLARE_AGENT(CInformSuccess)
         DECLARE_AGENT(CInformError)
         DECLARE_AGENT(CRequestNextQuery)
         DECLARE_AGENT(CExpectGoodbye)
         DECLARE_AGENT(CInformStartingNewQuery)
       DECLARE_AGENT(CGreetGoodbye)
 )
```

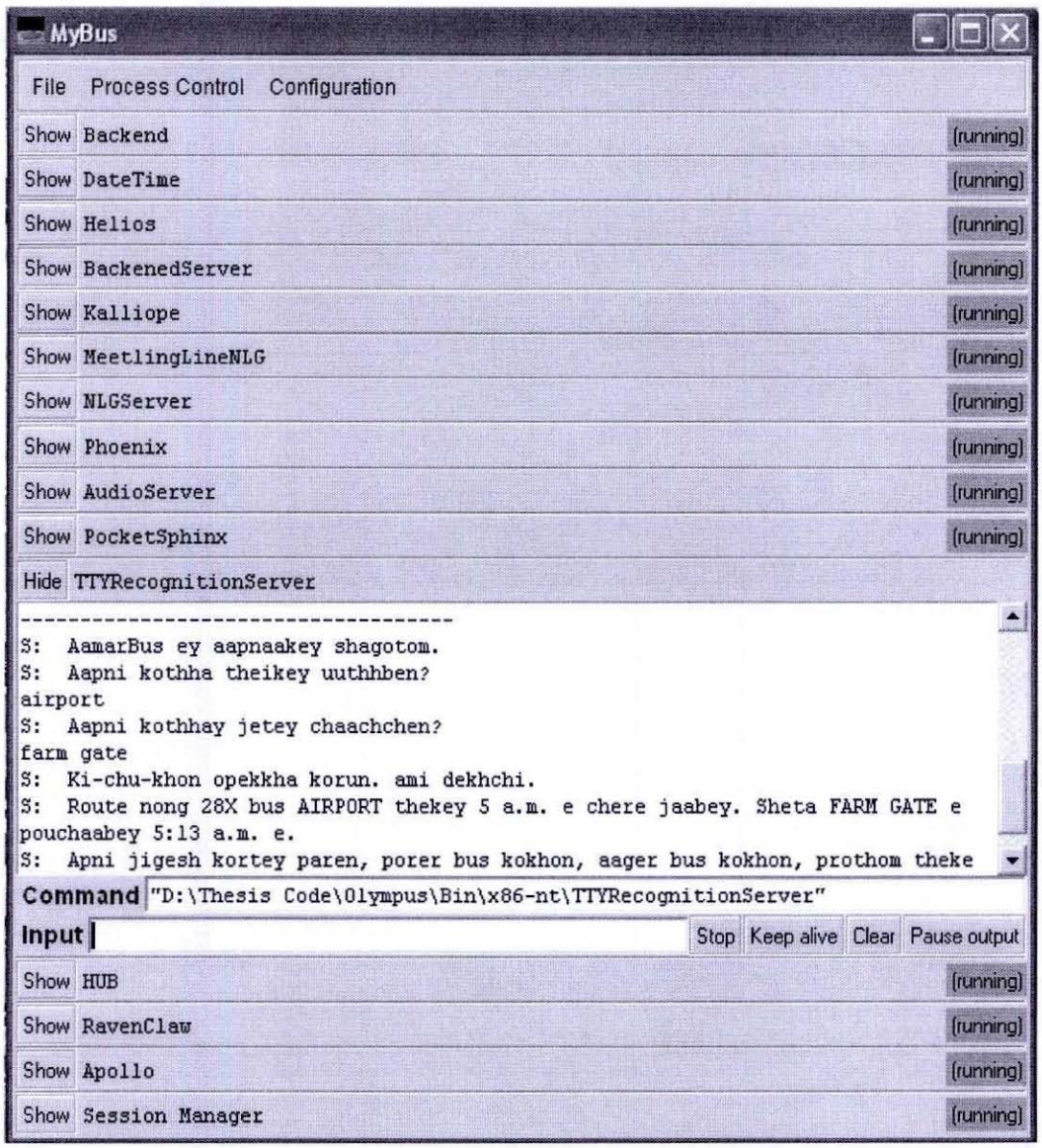
RavenClaw

```
DECLARE_DIALOG_TASK_ROOT(MyBus, CMyBus, "")
```

APPENDIX D

Screen Shot of AmarBus

Screenshot of AmarBus (Bangla Version of MyBus)



APPENDIX E

Bangla Unicode Chart

Bengali

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+098x		ঁ	ং	ঃ		অ	আ	ই	ঈ	উ	ঊ	ঋ	ঌ			এ
U+099x	ঐ			ও	ঔ	ক	খ	গ	ঘ	ঙ	চ	ছ	জ	ঝ	ঞ	ট
U+09Ax	ঠ	ড	ঢ	ণ	ত	থ	দ	ধ	ন		প	ফ	ব	ভ	ম	য
U+09Bx	র		ল				শ	ষ	স	হ			়	ঽ	া	ি
U+09Cx	ী	ু	ূ	ৃ	ৄ			ে	ৈ			ো	ৌ	্	ৎ	
U+09Dx								ৗ					ড়	ঢ়		য়
U+09Ex	ৠ	ৡ	ৢ	ৣ			০	১	২	৩	৪	৫	৬	৭	৮	৯
U+09Fx	ৰ	ৱ	৲	৳	৴	৵	৶	৷	৸	৹	৺					