

Mini-Project Saya

Main Loop Architecture

Improvement and Implementation

Final Report

**In Guidance of Professor Shlomi Dolev
and Mr. Michael Orlov**

Ma'ayan Ben-Nun

Liran Maimon

Ben-Gurion University of the Negev

Department of Computer Science

December 2007

Table of Contents:

1 Introduction	3
2 Presenting the Problem	4
3 Our Solution - Architecture and Implementation.....	5
3.1 Architecture (design)	5
3.1.1 Rules Table.....	6
3.1.2 Actions Table.....	7
3.1.3 Grammar Definition.....	7
3.2 Implementation.....	8
3.2.1 "DataBase" Class.....	8
3.2.2 Rules Table.....	8
3.2.3 Special Events.....	9
3.2.4 Setting Responses/Grammar Files.....	10
3.2.5 Other Additions	10
4 Examples of State Switching – Simulation.....	11
4.1 First Example	11
4.2 Second Example	12
5 Summary and Future Development.....	13
6 References	14

1 Introduction

SAYA was invented and developed by Hiroshi Kobayashi, an associate professor at the Tokyo University of Science. Her development began in 1993 at TUS, and has evolved into giving her the use of over 300 words and 700 phrases. Her lingual abilities span the use of Japanese, English, and now possibly Hebrew. In their article ["Development of Face Robot for Emotional Communication between Human and Robot"](#) published by Kobayashi and his colleagues, they explain their efforts to develop an interactive communication system to communicate with human beings emotionally as well as logically. The robot's face, expressions, her first name, and the personal attitude all contribute to the final product, a human friendly communicative robot, such as Saya. She has a human-like face and can express facial expressions similar to human beings. In their study they've shown the development of the new type face robot and new head motion mechanism. They've also shown basic ability for expressing facial expressions and in addition, they describe how to develop the new neck mechanism for realizing human-like head motion.

Saya was purchased by the Department of Computer Sciences in the Ben-Gurion University to serve as a learning and research tool. Saya is a robotic receptionist, located now in the new "Alon" building. She was brought to us as a departmental effort, a project led by Prof. Shlomi Dolev, including Prof. Eyal Shimony, Head of the Paul Ivanier Center for Robotics Research and Production Management; Prof. Matya Katz, Chairman of the Department and Dr. Ohad-Ben Shahaar.

In this report we shall describe our work on Saya's main loop. Our main goal in this project was to improve Saya's recognition abilities, relate her responses to be more in context for the current conversation, and more specific in given situations. The main idea was to modify Saya's loop to allow grammar switching in context of the conversation and in the right timing. In order to do this we defined certain "states" for Saya's conversation and integrated them into the main loop.

We will introduce the problem and present the design we used in an effort to solve the problem. We will describe our implementation and integration into the Saya code as well as an example of possible state switches. Finally we will suggest ideas for possible improvements and future work on the subject.

2 Presenting the Problem

Saya's main loop consists of periodical recognition (capture) and response. A grammar file and a responses file are loaded upon initialization, and are set for the entire run. The main problem derived is that all conversations start at the same point and can not "evolve" anywhere. Saya can not react to specific events and adapting her to different situations requires many changes in the initial grammar. In fact expanding her ability to react to given situations is problematic and forces the initial **static** configuration (grammar) to be very large. While using a large grammar we might "pay" with accuracy of the recognition because the captured word might be confused as another word in the grammar, and the Saya conversations may lose context. In addition this way of modifying the grammar does not satisfy an 'easy to change' interface and is very awkward.

Our work tries to support the given problem by allowing context switching according to given states of the conversation. The grammar would be reloaded as a result of certain events and we therefore tune the conversation to be more specific, and the entire design to be more dynamic and modular.

3 Our Solution - Architecture and Implementation

3.1 Architecture (design)

We will define three general states, which can be expanded into several specific sub-states, defined by demand.

State 1: General conversation – Initial/general mode

In this state we will use a general grammar which will be loaded as the default grammar and will be used for non-specific events.

State 2: Specific events

This state includes several sub-states which define all the different events that require the use of specific grammars or performing specific actions. These events may require certain reactions from Saya, and can be modularly modified for more/less events and actions.

State 3: Chatbot (free speech)

This state allows a functionality of “free speech” with Saya in which she will reply to all queries, i.e. closed grammar. We will not implement this functionality since it is not included in our project definitions, however we will allow transitions to/from this state.

State switches are event oriented, according to pre-defined rules. An event is defined as input related to a certain query. Rules will be set in the following format: **(state, event) -> state.**

This event-oriented state model allows easy integration of different functions and events into the Saya code, as well as easily reconfiguring existing functions. Therefore the implementation of new ideas becomes simpler to programmers that use this code.

A query can belong to one of sub-categories within state 2 (a “specific event”), such as a name of a department personnel, in which case this is an event that requires certain state switches. Thus every time a new query is received, it needs to be compared to a list of the special events that exist. If the query does not match any of the events in the list, then we assume it is a non-specific event and the loop code resumes. If we find a match within the list, then the correct rule is applied.

List of special events for a query:

1. “Chatbot”
2. Name of a department personnel
3. Goodbye statement

3.1.1 Rules Table

Current State	Event	New state	Grammar Switch
1 Initial	Query is recognized in the Initial grammar	1 Initial	-
All States	Query is a goodbye statement	1 Initial	Initial Grammar
All States	Query is a non-specific name* (first name Y)	2b Non Specific Name	Grammar for specifying name Y (possible last names of name Y)
2b Non Specific Name	Query is a last name(Specific name)	2a Specific Name	Grammar for person X
2a Specific Name	Person X is not in the office	2c Msg Option	Yes/No grammar
2c Msg Option	The visitor would like to leave a message	2f Recording State	-
2f Recording State	Finished recording	2a Specific Name	Grammar for person X
2c Msg Option	The visitor would not like to leave a message	2a Specific Name	-
All States	Query = “Chatbot”	3 Chatbot Mode	Initial Grammar
3 Chatbot Mode	Query is includes a recognized word from the grammar	1 Initial	-
3 Chatbot Mode	Query includes the word “message”	2d Chatbot Msg Option	Yes/No Grammar
2d Chatbot Msg Option	The visitor would not like to leave a message	1 Initial	Initial Grammar
2d Chatbot Msg Option	The visitor would like to leave a message	2e Chatbot Msg	Names Grammar
2e Chatbot Msg	Query is a specific name (Person X)	2g Chatbot Recording	-
2g Chatbot Recording	Finished recording	1 Initial	Initial Grammar
2e Chatbot Msg	Query is a non-specific name (first name Y)	2h Chatbot Non Specific Msg	Grammar for specifying name Y (possible last names of name Y)
2h Chatbot Non Specific Msg	Query is a last name	2g Chatbot Recording	-

*non-specific names are names that can be applied to more than one person, such as “danny”.

3.1.2 Actions Table

State	Actions
1 Initial State	Check if query belongs to one of the specific events
2a Specific Name	Check if person X is in the office
2b Non Specific Name 2h Chatbot Non Specific Msg	Say: "Please specify name"
2c Msg Option 2d Chatbot Msg Option	Say: "Would you like to leave a message?"
2e Chatbot Msg	Say: "To whom would you like to leave a message?"
2f Recording State 2g Chatbot Recording	Say: "Please leave your message", record message for person X
3 Chatbot	Check if input includes the word "message"

3.1.3 Grammar Definitions

Each grammar has the following properties open/closed, small/large. In a closed grammar, the query search will always produce the closest match possible within the grammar and return its corresponding response, meaning there will never be a null response. In an open grammar, when the query search doesn't produce a close enough match the query is not recognized and a null response is returned. Small/large refers to the size of the grammar. A larger grammar increases the chance of a query mismatch, in which Saya "confuses" what was actually said with a similar but different word.

Grammar	Open/Closed	Small/Large
Initial	Open	Large
Person X	Open	Small
Non-specific name Y	Closed	Small
Yes/No	Closed	Small
Names	Closed	Small

3.2 Implementation

3.2.1 “DataBase” Class

In order to organize our implementation we created a new class called “DataBase”. This class is being used by several existing classes such as “Loop” and “Responses”, and we wanted it to be initialized only once and have all the other classes use the same instance. Therefore we decided to build this class in a Singleton design pattern, which matches our needs.

This class includes two hashtables – “grammars” table and “rules” table. The grammars table is used by the Responses and RecognitionImpl classes, and its purpose is to help find the correct name of a grammar/responses file that needs to be uploaded. Its input is an int that symbolizes a certain situation and its output is another int that represents the actual number that needs to be added to the file name in order to upload the appropriate file for the requested situation.

The rules table is used by the Loop class, and will be described in the next section. The needed rules and additions to file names should be implemented by the programmers into the “DataBase” constructor, where they are loaded onto the hashtables during the first time this class is called.

3.2.2 Rules Table

Our implementation is based on the design shown in the previous section, of which the main part is the “rules table”. This tables’ inputs(keys)/outputs(values) are different states and events that are defined as two sets of enums in class “DataBase”. This class also includes the “rules” hashtable that simulate the rules table itself.

The input for this table is a pair of a state and an event, which represent the current state that Saya is in at the moment and the event that has occurred. To make this pair we created another new class called “Pair”, and such Pairs of two enums are the input of the rules table. The output is a single enum of a certain state that is the next state that Saya should be in, according to the previous state and the event that were given.

3.2.3 Special Events

Different events bring on the need for applying different rules. In order to know which rule is needed we first need to identify the event that has occurred. In our design we included a list of “special events” that need to be recognized in regard to a new query, and then certain actions must be taken according to the event at hand.

The first part of recognizing a special event we may be dealing with is implemented as a function called “checkSpecialEvents” in the existing “Responses” class. In this function we check all possible special events and compare them to the query to see if there is a match. Every possible special event has a corresponding responses file that is loaded into a hashmap called “test”. Each of these responses files include only entries that could belong to their specific special event. Therefore if on one of the checks for a given query we receive a response that is different than null, then this query belongs to that special event. The function returns a number symbolizing the special event that was found, or 0 if no special event was found.

The second part of taking the proper actions according to the special event that was found is implemented as a switch on the int “special” that is returned from the “checkSpecialEvents” function. For each case the code includes the matching actions and state switches if necessary, based on the rules and actions tables in our design.

The case of no special event (case 0) includes another switch on the enum “state”. Certain queries that are not a special event themselves may be part of a series of events that were initialized by a special event. Therefore they still require taking special actions and/or state switches, and these cases are handled in this switch.

3.2.4 Setting Responses/Grammar Files

Some additions have been made to the Responses and Recognition classes to support the changes we have made in the code. In the past Saya uploaded one grammar file and one responses file during initialization and used these files only. Since our project involves using many different responses/grammar files and constantly changing between them, we needed to add functions that allow setting these files as needed during the run of the code.

In each of the Responses and Recognition classes we have added two different functions that set a file in the same way it is done in the corresponding constructors. The first function uses the grammars table in the “DataBase” class to find the file name, for example in the Responses class it is named “setResponses”.

The second function uses a given query to find the file name and is named “setResponsesByQuery”. This function is used so far only in the case where the query is a name of a staff personnel and the file needed is specific to that person. Note that the file name should be identical to the name in the query.

3.2.5 Other Additions

As stated in section 3.2.3, we have added a function called “checkSpecialEvents” in the Responses class. Also in the Loop class we have added a function called “recording”. This function is used when Saya needs to record a message, and can be changed during the integration of the code with the implementation of the message recording process.

Our work on this project was done using our PC which does not have java speech API installed, therefore the query was received via keyboard instead of a microphone. This meant that all of the Recognition sections could not be used, including the entire use of the grammar files. Therefore we needed to add a class called “RecognitionDummy” in order to simulate the recognition part of the code.

Another problem that came up as a result of this was that when a query was mistyped or not in the currently used responses file, then Saya couldn’t identify the query and its response would then be null. To deal with this problem we added in class Loop within the main loop a section that checks if the response that was received is null. If that is the case then the response is set to be “I’m sorry I didn’t understand you”, and the loop can carry on. This could never happen when using the code on Saya normally since she always operates on a closed grammar and therefore always recognizes the input as the closest word she can find in the grammar.

4 Examples of State Switching - Simulation

We will present two examples of possible runs including several state switches

4.1 First Example

Current State	Event	Response	New state	Grammar Switch
1 Initial	Query is "hi"	"hello"	1 Initial	none
1 Initial	Query is "danny" – a non specific name	"please state Danny's last name, Barash or Berend"	2b Non Specific Name	Grammar for specifying "Danny" (possible last names of "Danny")
2b Non Specific Name	Query is "barash" – a specific name	"his office number is 105"	2a Specific Name	Grammar for Danny Barash
2a Specific Name	Query is "mood"	"Danny's mood is tired"	2a Specific Name	none
2a Specific Name	Query is "shlomi"	"his office number is 210" **	2a Specific Name	Grammar for Shlomi Dolev
2a Specific Name	Shlomi is out of his office	"shlomi is not in the office, would you like to leave a message?"	2c Msg Option	Yes/No grammar
2c Msg Option	Query is "yes"	"please state your message now"	2f Recording State	none
2f Recording State	Message received, finished recording	"What else would you like to know about Shlomi?"	2a Specific Name	Grammar for person X
2a Specific Name	Query is "good-bye" – a goodbye statement	"bye-bye"	1 Initial	Initial Grammar

** This response is not actually said by Saya, since the following event occurs immediately and before the response is spoken.

4.2 Second Example

Current State	Event	Response	New state	Grammar Switch
1 Initial	Query is "be sad"	"I'm sad"	1 Initial	none
1 Initial	Query is "chatbot"	"switching to chatbot mode, say 'message' to leave chatbot mode"	3 Chatbot Mode	none
3 Chatbot Mode	Query includes the word "message"	"Leaving chatbot mode, would you like to leave a message to someone?"	2d Chatbot Msg Option	Yes/No Grammar
2d Chatbot Msg Option	Query is "yes"	"Who would you like to leave a message to?"	2e Chatbot Msg	Names Grammar
2e Chatbot Msg	Query is "dolev" – a specific name	"Please state your message now"	2g Chatbot Recording	none
2g Chatbot Recording	Message received, finished recording	"finished recording"	1 Initial	Initial Grammar

5 Summary and Future Development

We have implemented new dynamic context switch oriented architecture for Saya's main loop. We modified the grammar/responses loading and created an event oriented model. This model's advantages are expressed in a modular, easier to manipulate environment. The new model is the basic platform for many different states and events which could be later on integrated as necessary. Various specific situations could be handled easily by only adding the necessary rules and grammars, and the adaptation to changes becomes simpler.

Since Saya is a development teaching tool, and since it is frequently modified and improved, its basic platforms should be easy to manipulate and to understand. Therefore it is of great importance that future development on Saya should conserve this dynamic modular model. As for our project, future integrations should be made with the message recording process and the “out of office” data which could be extracted via an online database of some sort.

One possible suggestion we thought of for further improving Saya's abilities is equipping her with a digital notepad. Since Saya sometimes has difficulties in recognizing words, it may be possible for someone speaking with her to write down what they want to say if after several tries the word isn't understood. This can be also used to verify the identity of a person speaking to Saya by their signature.

6 References

1. [News@BGU](#), Winter 2007

<http://cmsprod.bgu.ac.il/NR/ronlyres/F3311087-8BB0-4C3D-83D0-4A544AAA07C5/0/newsatBGUwinter2007.pdf>

2. "[Development of Face Robot for Emotional Communication between Human and Robot](#)", Hashimoto, T.; Hiramatsu, S.; Kobayashi, H.; Mechatronics and Automation, Proceedings of the 2006 IEEE International Conference on 25-28 June 2006 Page(s):25 – 30

<http://ieeexplore.ieee.org/search/wrapper.jsp?arnumber=4026050>

3. [Japanator](#)

<http://www.japanator.com/?t=robots>

4. [Saya, Michael Orlov's Website](#)

<http://www.cs.bgu.ac.il/~orlov/teaching/saya/>