https://doi.org/10.2352/ISSN.2470-1173.2017.6.MOBMU-315

Two-Tier State-Machine Programming for Messaging Applications

J.Morales, R.Escobar, S.Kaghyan, G.Natarajan, D.Akopian, The University of Texas at San Antonio; San Antonio; P. Chalela, A. Ramirez, UT Health Science Center at San Antonio; San Antonio; A. McAlister, the University of Texas, School of Public Health at Austin Regional Campus; Austin; TX/USA.

Abstract

Personal computers with Internet access are the most common way to transfer information nowadays. Furthermore, messaging technologies give us the ability to create dynamic interactions between systems. The purpose of this research work is to introduce an interactive messaging system that allows for a human user to communicate with a dedicated software application via messaging dialogues. The conversation is modelled as a system of interconnected state machines. Several techniques are presented that allow to manipulate complex dialogues preserving integrity, even in multiple languages. A smoking cessation case study is also presented, in which the proposed system holds 6-month long personalized conversations with all users concurrently. In such case study, the system monitors the smoking habits of these users independently and helps them quit smoking. The two-tier architecture provides a flexibility for relatively quick system updates when intervention protocols are revised.

1. Introduction

With the global prevalence of mobile technology, text messaging has become a sound approach to promote patient engagement and a proven method to promote health and behavior change. Short message service (SMS) texting has been used for health service appointment reminders, preventive activities, medication adherence, smoking cessation, and monitoring and the self-management of chronic disorders such as diabetes [1], [2], [3]. It provided an extraordinary opportunity for innovation in the delivery of tailored interventions to improve the health of the US population and reach traditionally underserved groups, by offering services that can be used by participants in their natural environment and in real-time, where the person is located, without them having to attend services, and providing the anonymity people like.

Volume text messaging systems are currently broadly used for health promotion and other campaigns. To avoid biasing impacts of diverse mobile phone technologies, most of the health applications use the least advanced, yet broadly utilized and inexpensive plain text messaging feature. While simple, the texting allows to collect a feedback from participants using polling mechanisms and process related data. One can mention the following systems in reported studies: Twillio [4], Rapid SMS [5], Frontline SMS [6], Mobenzi Researcher [7], Magpi [8] and Trumpia [9] among many others. They have been utilized for health diagnostic services, nutrition surveillance, household surveys and marketing. Systems like Twilio serve as message relay aggregators that offer API's to the developers to send and receive messages programmatically. They also provide simplistic automation capabilities such as confirmation responses.

While the text messaging applications are numerous, very few systems provide automated services, when preprogrammed protocols can be used to completely exclude human campaigners from the communication chain. A smart server application will know what to send and how to respond conditioned by the interactions with the participants. The messages will be communicated through API's provided by the aggregators. This paper provides a methodology for automated messaging systems design. Smoking cessation is selected as the case study.

Smoking remains the leading cause of preventable death in the US and it is a well-known risk of twelve types of cancers, cardiovascular disease and other health problems, imposing substantial health and financial costs on our nation [10],[11],[12]. In fact, smoke related issues account for more deaths than the aggregate of drug abuse, suicide, motor accidents, murder, and AIDS [13]. Thousands of people die of active and passive smoking every year. As such, several intervention techniques have been designed to help with smoking cessation.

The proposed smoking cessation program aims to create a flexible two-tier state machine architecture that allows for relatively quick system adjustments with protocol revisions. There are similar smoking cessation programs which help people quit smoking. Few of existing programs are reported such as Text2Quit [14], Miquit [15], and Quitplan [16]. These systems typically use pre- and post-quit messages, peer-ex smoker messages, medication reminders and multiple opportunities for reminders. Unfortunately there are no reports on upgrade flexibility of these systems. Meanwhile, any health promotion campaign needs continuous adjustments to optimize intervention efficiency. In addition, multiple language support might be needed to involve diverse population segments. Thus both upgrade flexibility and scaling up the system for multilingual support motivated the development of a two-tier system, where one tier is used for the protocol definition, while the second tier maps any protocol definition onto a system architecture. In particular, the first tier of the software application consists of "state machine definition files". These files contain the logic of the conversation. A designer can create a complex, interactive conversation simply by modifying these configuration files. The second tier of the system is the subsystem that takes these configuration files and uses them to implement an automated text messaging system.

2. Dialogue trees

There are cases in which the conversation between machine and user may be more complex than just a sequence of questions and answers. In a complex conversation, the system may need to evaluate variables in real-time before deciding which message or question to send next to the user. A decision tree is a simple way to program such type of conversations. This technique has been applied in the so-called multiple-choice games (MCGs). In these type of games, the player traverses a decision tree. At each node, the player gets a multiple choice question (i.e., "you find a dragon: do you want to kill the dragon or do you want to ride the dragon?") and he must choose among the available answers. ChoiceScript is a programming language for designing these type of games [17]. Interactive novels are another example application of decision trees

to conversations. These novels have become more popular with the arrival of HTML since now the reader only needs to click on a hyperlink to get the text that corresponds to the decision he has made in the story. Nowadays, the reader does not need to be referred to the page number or fragment of the book that contains the text that corresponds to the decision he has made, as it was done before with the printed version of interactive stories. Conversation tree logic is used in the video-game industry as well. This type of logic allows human players to have a conversation with non-player characters (NPCs). In a menu-driven conversation with a magician, for example, the NPC may ask the player "How can I help you?". The player may then be offered three choices: A) to buy a potion, B) to buy a spell or C) to learn where the dragon is hiding. Keyword-based conversations have also been used to program NPCs. This allows the player to access specific conversations on-demand, which can also be applied to automated messaging systems.

3. State machine constructs

The proposed system models conversations between the user and the messaging agent as a set of interconnected state machines. A conversation is specified with two configuration files. The first file contains the definition of each state or node (i.e., an action such as sending a message, or a sequence of actions). The second file specifies the transitions between different nodes in the state machine. We refer here to the first file as the *imports* file and the second file as the *successors* file. Additionally, conversations that repeat very often can be encapsulated into their own *routine-state-machines* to which the system can transition in and out as it moves through the *main-state-machine*/conversation. An example of a conversation broken down into states and actions is shown in Fig. 1. Blocks in the diagram are stored in the *imports* file. Arrows are stored in the *successors* file.

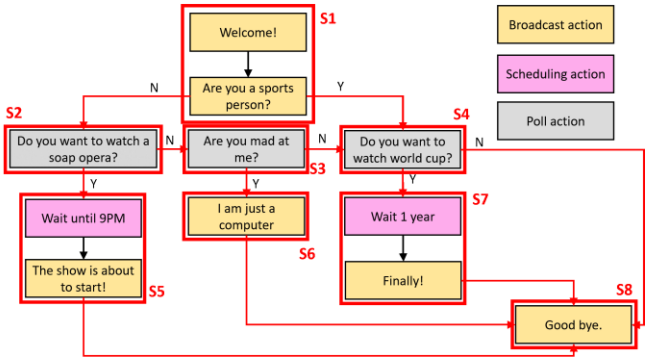

Figure 1. *Example conversation broken down in states and actions*

3.1. States and actions

Different types of actions can be executed inside a node or state. There is no limit to the number of actions that can be performed on a single node. An example of a node with several actions is shown in Figure 2. The system allows for five different action types, denoted as follows:

B: This action can be utilized for delivering a broadcast message. A reply is not expected from the user. The action is completed as soon as the message is sent to the user.

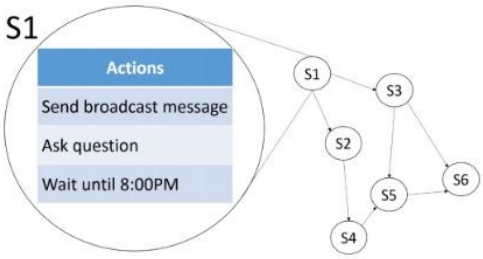

Figure 2. *Each state consists of one or more actions*

P: This action is used for sending a poll message. The action is completed as soon as the state machine receives a reply from the user, whether it is an automatic reply or an actual reply from the user.

U: This action is used for updating the response from the user to a previous poll. An example of an update command is as follows:
UPDATE(previous->poll); R(new->poll))
This command indicates to the state machine that it should update the reply to the action previous->poll with the reply to the action new->poll.

T: This action indicates to the state machine that it should wait for some time before transitioning to a new state. It can be used to insert an artificial delay. The delay starts as soon as the state to which the T action belongs is entered.

D: This action is used for loopback (i.e., sending back the user to a node in the protocol where he has been before). In this case, the user's history between such previous node and the current node is deleted. Disabling history is needed to avoid having the system traverse the entire state machine, all the way back to the current node by processing the user's history. History disabling and loopback are graphically depicted in Fig. 3 below.

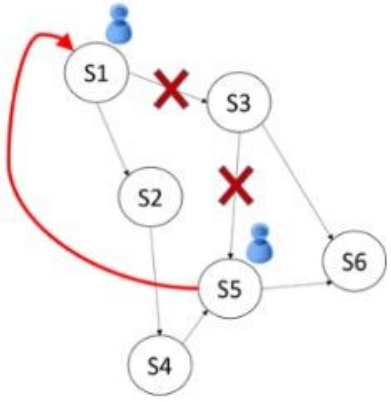

Figure 3. *The system can loopback to a previous point in the conversation and disregard the user's replies to any message from then on*

3.2. Action scheduling

The system is able to perform actions via timestamp-based and delay-based scheduling. In some cases it may be desirable to send a question, for example, at a specific time of the day, while in others, the question should be sent some time after the system has performed the previous action. These scheduling is available for any action type. Examples of timestamp-based and delay-based scheduling for poll type of action is shown in Fig. 4.

```
January 5,2017 9:56 PM[Day 1] <<247367>> :.: Are you Verizon Carrier? Text Y or N
(Answer): Y
January 5,2017 9:57 PM[Day 1] <<247368>> :.: What is your age? Text the number.
(Answer): 18
January 5,2017 9:58 PM[Day 1] <<247370>> :.: Are you male or female? Text M or F
(Answer): M
January 5,2017 9:59 PM[Day 1] <<247371>> :.: Are you Hispanic or Latino? Text Y or N
```

Figure 4 *Output log from a simulation of a system that uses timestamp and delay-based scheduling.*

4. Nodes file

States and actions are specified in an *imports* file. This file contains one row for every action. The columns in the file specify the node that the action belongs to, the order in which the action is to be executed with regards to other actions in the same node, the action type, scheduling (i.e., execute action after a certain delay or at a specific time of the day), whether the action is the entrance point of the system or the exit point of the system. Some of the fields apply only to polls (i.e., actions of type *P)*, such as, the number of times the system should resend the poll if the user does not answer, the delay between such repetitions, and the prefix and suffix that should be appended to the last repetition. There is also a field that can be used to pass an indexing parameter which may be used by a broadcast action to index messages from a table. The layout for the imports or nodes file is shown in Fig. 5.

Init/Stop	Action	Type	Sequence	Message
INIT	Welcome	B	0	Welcome t
	Welc	Spanish	Category	Accepted Re Delay Timestamp
	Welc	¡Bienvenido a Quitxt del	UT Health	0 -1
	Welc	Te ayudaré	Deadline Type	Repeat Repeat Del Default An
	Carri	Para dejar	-1	0 0
	Age	Pop quiz a	-1	Last repeti Last repetition Suffix
	Iam	¿Es Verizon	-1	END
		¿Cuàl es tu	-1	END
		Lo sentimo	-1	END
			-1	END
			-1	We can help you quit smoking! Ju END
			-1	We can help you quit smoking! Ju END
				END

Figure 5. *Import file*

5. Transitions file

Transitions within the main state machine are specified in the *successors* configuration file. Entries in the successors file specify source nodes (i.e., states) and destination nodes. Regular expressions are used to check for validity of answers from the user. Conditional expressions are then used to decide where the user should go in the state machine depending on his response. All source and destination nodes must be specified in the imports file, unless the destination is another state machine, such as a routine state-machine. Routine state-machines are specified in their own imports and successors files. The last transition made in a routine state-machine must be a transition to a *return* node. This node takes the system back to the main state machine. Figure 6 shows the part of the "successors" file. The first column shows the initial state. The next column shows the last action associated with the state. As mentioned before, more than one action (such as sending a message) may be executed by the system within a single node, in sequence. Actions can have arbitrary names and the name can be the same as the state name, as shown in Fig. 6. Even though the system can transition out of a node from any action within the node, it is more convenient to always transition out of the last action in the node, as this simplifies the state machine design and maintenance of the design.

The third column shows whether it is a broadcast message or a polling message. If it is polling (i.e., question) message, then there must also be a regular expression associated specified with the transition, which is specified as *R={exp}*. For broadcast messages, this parameter is specified as *S*, indicating that the transition to the destination node should be made right after the last message in the current node is sent, or the last action in its sequence is executed. The fourth column specifies whether the expression is a numerical comparison or a regular expression. The last column shows the destination state.

Welcome	CarrierChe	R=.+	M	Age
Welcome	CarrierChe	R=EXIT	M	GoodBye
Sorry	Iam	S	M	exitNode
GoodBye	Goodbye	S	M	exitNode
TheEnd	finMessag	S	M	exitNode
Age	Age	R=EXIT	M	GoodBye
Age	Age	R=[0-9]+	M	InitialQuestions1
Age	Age	R=>=18	N	InitialQuestions1
Age	Age	R=[0-9]+	M	Sorry
Age	Age	R=<18	N	Sorry

Figure 6. *Successor File*

6. Polling data from users

In any messaging system, users are sometimes reluctant to answer a question, or the system may not receive a reply due to temporary technical issues. To deal with these situations, the system can retransmit the poll message for a certain number of times with a delay between each repetition, which can be specified by the designer of the conversation. If no answer is received from the poll the system can proceed to move to the next action by using a default answer to the poll, but a flag is raised to indicate that the user did not actually reply the default answer that the system is using for branching or content publishing purposes. For politeness, the designer could also specify a message to be prepended or appended to the body of the last repetition of the question or poll.

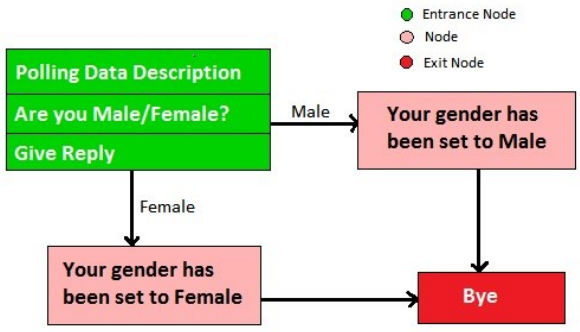

Figure 7. *Example application where poll data is taken from the user, printed in a message and used for decision-making*

To have insight into how polling and recalling of poll data works, an example is illustrated in Fig. 7. The figure also covers data recall (printing an answer from the user in a later on message).

7. The default pathway

Assuming a default answer whenever the user does not reply to a poll means that there is a predefined path that the user will follow if he/she does not reply to any poll. This is shown in Fig. 8. Designers and testers of the state machine can travel to different places inside the state machine by moving through this default road and deviating from it wherever needed for maintenance purposes. However, relocation may be a better idea in cases in which only a small region of the protocol is to be tested. Continuity of default pathways should always be verified in order to avoid having the user reach a dead end by default.

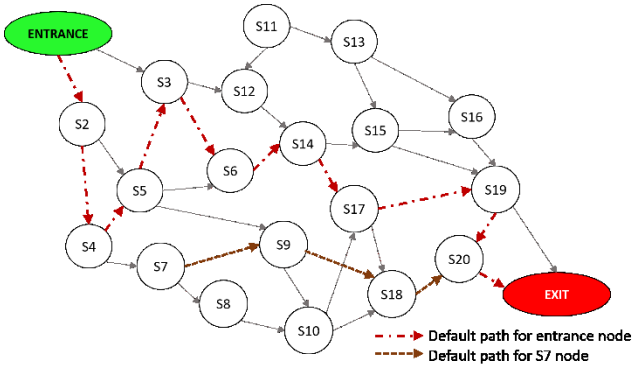

Figure 8. *After every node, there is a default path that the system will follow if the user does not reply to any poll messages*

8. Ability to update user data

The user can update his/her answers to specific questions in the conversation. This is done via on-demand conversations. For example, at the beginning of the conversation the system may ask the user about his/her age. The system then lets the user know that he may text the word AGE if he/she wants to change his answer later. At any point, the user may text AGE and the system sends him to an on-demand conversation that asks him about his new "age". The answer to the original "age" question is then replaced by the answer to the question given in this on-demand conversation. Any subsequent parts of the system that make use of the "age" parameter will then be using the updated data.

9. On-demand conversations

Throughout the program, participants can send a variety of keywords at any time to receive additional help. These keywords are called on-demand keywords. For instance, the user could send the word HELPNOW anytime. The system reacts by entering into a state machine that has a conversation of its own. Once the on-demand conversation is over, the system will exit this on-demand sate machine and return to the state that is next in the main state machine. Figure 9 depicts the concept of on-demand text messaging.

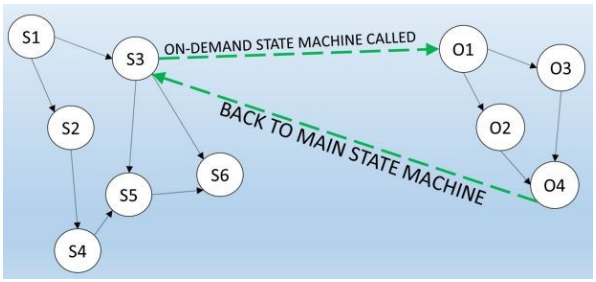

Figure 9. *On-demand a text messaging state machine*

10. Conversation routines

There are cases in which the designer of the state machine might need to use a sequence of states and actions more than once. For example, consider a scenario in which the system must have a conversation with the user every day. As part of the conversation, the system must first salute the user every day before sending any other message. In such scenario, the salutation may include saying 'Hi! How are you?', and a specific message must then be sent depending on whether the user replied 'good' or 'bad'. In such case, it is desirable to have a routine for saluting the user, and to replicate this routine every day in the morning. This approach is similar to calling a routine several times in a computer program. This routine is modelled in the system as a state machine of its own. But to the main state machine it looks as a single node with an entrance and an exit. The concept of routine state machines is depicted in Fig. 10.

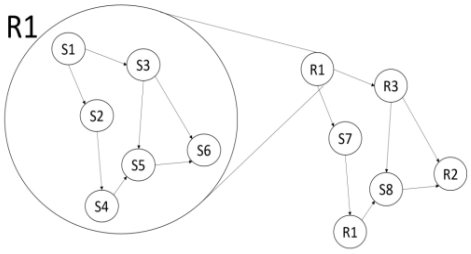

Figure 10. *State machine that uses states and state machines encapsulated into routines*

This approach means that any change in logic or content that is done in the master copy of the routine state machine will propagate through the entire multiday conversation, which can save a significant amount of work to the designer of the dialogue tree. The first state specified for a routine would be the entrance point. A Return state would be used to indicate to the state machine engine that it should exit the routine state machine and return to the main state machine. State machine routines are used when a sequence of messages repeats in many places, for efficient debugging and to cope with rapidly changing specifications. Figure 4.5 shows how an entire routine can be treated as a single node within the main state machine. Figure 11 depicts a conversation that makes use of state machine routines.

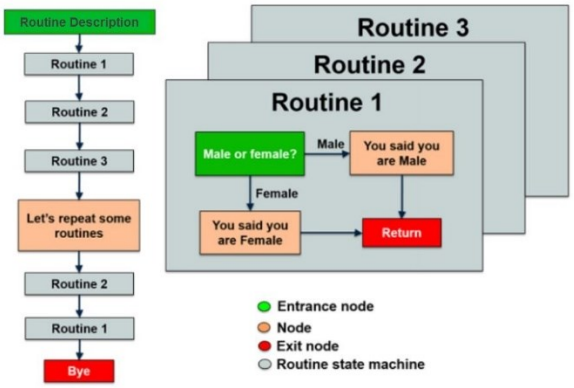

Figure 11. *General structure of a routine and their connection*

11. Message rotation and indexing

For more human-like behavior, the designer of the state machine might want to use the same state machine routine several times, but modify the content to seem less repetitive to the user. For example, in the salutation routine the designer may want to say "hello" to the user in some cases and "hi" in other cases. The architecture of the routine state machine (i.e., transitions between nodes, scheduling, regular expression evaluations) may need to remain intact since only the content of the messages needs to be changed. In such cases, the designer could specify content variations for the messages delivered by the routine. These variations are specified in a separate table. This feature is useful when implementing routines inside a state machine. In cases where the branching logic is the same, but the content of the messages is different, the system should be able to use the same routine, but rotate the content of the messages inside the routine. For these scenarios, the system has the ability to either rotate messages from a given table, or to choose particular messages which are specified with an index given to the routine state machine as a parameter. The screenshot in Fig. 12 shows an example of a rotation table, the *crave* table. In this case, each time user sends the keyword CRAVE, a different message is sent back to the user. Message variations are not restricted to on-demand messaging. They can occur at any action in the system.

	A	B	C
1	CRAVE	Let"s help you lighten the crav	Te ayudaremos a reducir el antojo;
2	CRAVE	Think of your craving like traffi	Piensa de tu antojo como el tràfico
3	CRAVE	Hit up Youtube for new cat vide	Busca en YouTube los nuevos vide
4	CRAVE	Talk to yourself...say "I can han	Convèncete a ti mismo... di "yo pu
5	CRAVE	Rocky"s inspiration: "It"s about	Inspiraciòn de Rocky Balboa: Se tra
6	CRAVE	Call or text friends who know a	Llama a tus amigos y pideles apoyo

Figure 12. *Screenshot explaining message rotation*

12. Translation table

Users can have conversations with the system in different languages. This is done by signing up with a keyword that corresponds to each language. Once the user signs up, he sends and receive messages in whichever language he has chosen. The logic of the system remains the same. By using translations tables, broadcast messages are translated from English to whichever language the user has chosen before they are sent to the user. Incoming messages, on the other hand, are translated to English with a different translation table before they are interpreted by the state-machine. Figure 13 shows a translation table for incoming messages.

A	B
SEGUIR	GO
HOMBRE	MALE
MUJER	FEMALE
H	M
M	F
SEXO	GENDER
S	Y
N	N
CIGS	CIGS
DINERO	DOLLARS
BEBIDAS	DRINKS
RAZONES	REASONS
APOYO	SUPPORT
NICOTINA	NICOTINE
NOANSWE	NOANSWE
RN	NR

Figure 13. *Example translation table for user replies*

13. Relocation capability

For maintenance purposes, system admins can immediately be transported to specific places of the conversation, allowing them to bypass the delays associated with state machine transitions, which are often due to traffic, scheduling, and processing overheads in the server. In such cases, the system makes use of default conditions for content publishing and branching purposes, since some states and transitions may need to make use of historical data, which in this case may be non-existent. If this historical data needs to be initialized to test its effect in other parts of the conversation, the administrator must first relocate to the state(s) to be initialized before relocating to the state(s) that he wants to explore. Relocation of a user is shown in Fig. 14. The ability to immediately relocate a user without the need to traverse the state machine nodes between the two locations is a valuable tool when it comes to debugging the conversation.

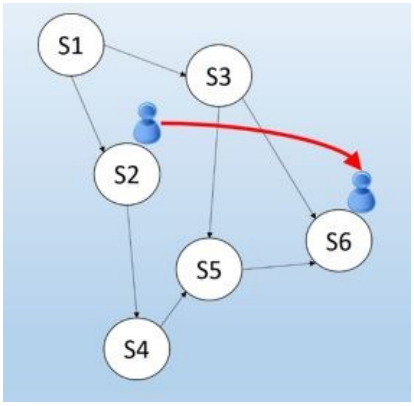

Figure 14. *Relocation capability allows for administrators to perform maintenance faster*

14. User migrations

There is a portability feature that allows admins to transfer active users from one state machine to another. This feature is used whenever the dialogue tree has been updated to a new version. This

migration feature is critical for being able to make changes to an active project.

To migrate the active users, the users must be located in a *transfer state*. A transfer state is any state that is common between two state machines (i.e., has the same name in both state machines). Figure 15 depicts the migrations of users from one version to another. A user can be migrated to an updated version only if there is a common state located in both, the source and the destination state machines. Migration is scheduled in such a way that it does not affect the functionality of the protocol, i.e., to prevent the new state machine from executing a broadcast or delay action that was already executed by the previous state machine.

Data like polling replies and default flags are transferred to the new state machine so that historic data is available to the new state machine. This historic data may be needed for branching or content publishing purposes. The new state machine will make use of default answers in case historic data is unavailable. This may be the situation when a new question has been introduced into the conversation.

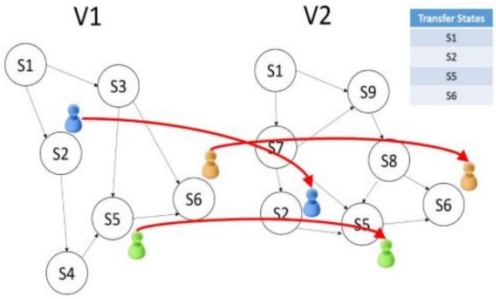

Figure 15. *Migration of users from one version of the automated messaging system to another*

An example of a migration is shown in Figure 5.3. The user is asked for his/her gender, and he/she is then migrated to a destination project. His/her answers are recalled later on in the destination project.

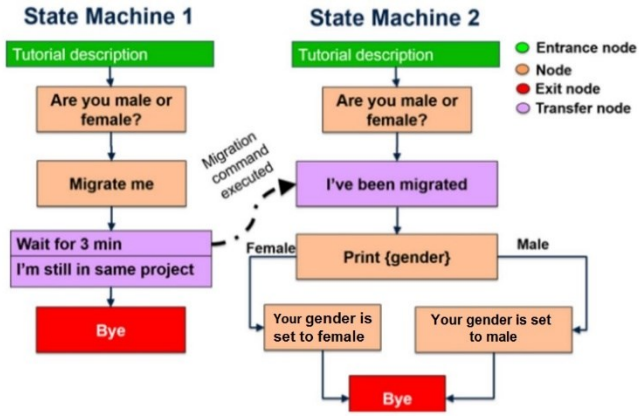

Figure 16. *Example of a migration of users from one state machine to another*

15. Simulation Mode

Being able to simulate an automated messaging system is essential. Simulation is needed mainly due to the limitations of messaging technologies (possible delays), and also the limitations of human testers. SMS technology in particular introduces many delays so there is a limit to how fast the system can be tested in the deployment environment. Also, it would be extremely difficult for a human to test a long protocol such as the use case presented in this work. If the protocol is very long and it contains many branches, it may turn out to be very burdensome for a human to traverse it by hand. In fact, it would take many years to traverse by hand every branch of the test case presented here, given that the default conversation pathway takes more than 200 days.

A screenshot of the simulator web page is shown in Figure 17. The interface allows for the user to specify the segment of the state machine that he wants to test (i.e., entrance node and exit node), the project or conversation protocol that he wants to test, the language, and an answer sheet to any questions asked in the conversation. Additionally, the tester can specify a participant number, in case he wants to traverse the conversation using the answers given by another participant, which may be useful in case the tester wants to recall a previous simulation. There is also a time acceleration feature which allows the tester to simulate delays.

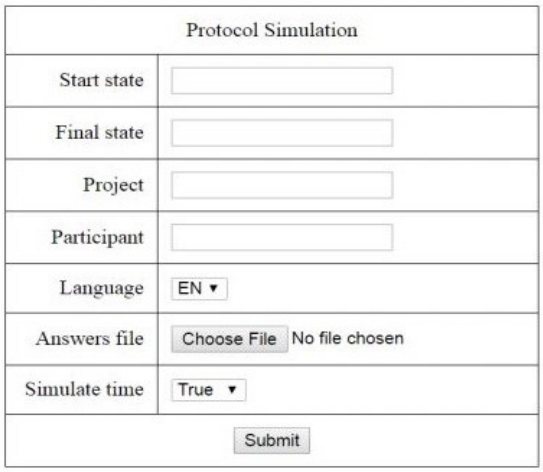

Figure 17. *Screenshot of the online simulation environment*

16. Time acceleration Feature

The system can work with an accelerated time for a tester to be able to traverse the entire program in a matter of hours or days and not months. The designer can choose a time acceleration factor. For example, if an acceleration factor of 60 is chosen, an entire day of the real system can be simulated in 24 minutes.

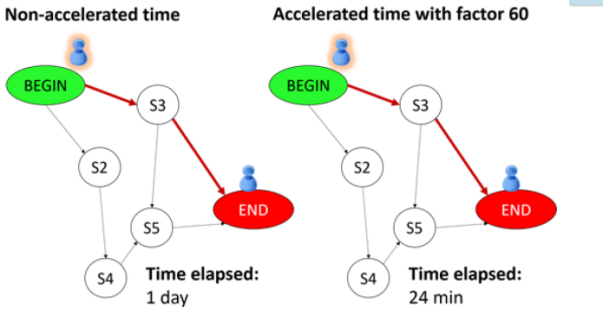

Figure 18. *Automated messaging interactions can be accelerated with an adjustable acceleration factor for testing purposes*

However, the acceleration factor should not be too high because the overhead time (i.e., time it takes for the system to process an event, such as a transition) cannot be accelerated. This means that the time interval between two events in the log file of the simulation, Δt_l, will not be exactly the same as the difference

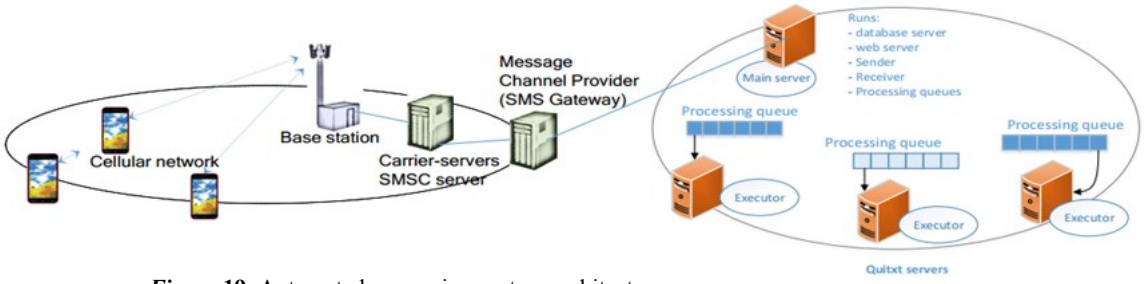

***Figure* 19**. Automated messaging system architecture

between the times at which the two events are scheduled to happen in the real system. The interval between the two events in the log file can be modeled using following formula:

$$\Delta t_l = (\Delta t_s/a + T_0) * a,$$

where T_0 is overhead time, and a is the acceleration factor. The lower the acceleration factor, the more accurate the simulation will be. A pictorial representation of the simulation acceleration process is shown in Fig. 18.

17. System architecture

The architecture of the messaging system is composed of several logical components, including a database server, a sender process, a set of processing queues and executors, a web server, a receiver process, and one or more data aggregators. Internally, the system represents a project as a state machine. Each project has a set of states, actions, participants, and messages associated with it. This organization allows the users to execute more than one state machine (i.e. a project) in the same system. Execution of tasks related to a given state machine is distributed among different servers. This feature allows the users to process in parallel multiple actions corresponding to participants and at the same time it allows to increase our processing resources using a scale-out paradigm. The design of the system is presented in Fig. 19. The functionality of each component is described in detail below.

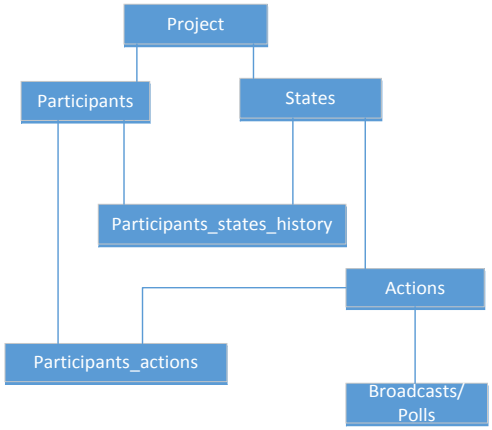

Figure 20. Main entities and their relationships

18. Database servers

The database server provides the required persistence for all components across the system. The data stored in the system database per project includes a state machine, participants, actions, messages (broadcast, polls or on-demand), and logs, among others. Figure 20 presents the main entities part of the system and their relationships. Projects, states, actions, and texts messages are usually static after they are defined in the formerly described MS Excel files and imported into the database. After the state machine has been imported, the most changes come into the database from new participant registrations, logging records corresponding to executing participant actions, moving participants from one state to another and logging messages that are sent/received to/from each participant. Such records are inserted into the database as soon as they occur.

19. Processing queues and executors

In the system, *executors* are entities that are in charge of processing participants received from the sender process. Each executor is associated with a *processing queue* on the sender process. The sender process serializes and sends participant objects to the executors. The executors then process the participants and return a status value to the sender process. Status values that can be send to the sender process after processing a participant include: processing error, success, or time to wait for the next action. Processing queues are priority queues managed by the Sender process. Each processing queue is associated with an executor.

20. Sender process

The *sender* process is the main component of the proposed automated messaging system and is in charge of performing all actions that need to be executed for all participants of a project. It uses a poll mechanism that traverses the whole set of participants in a project and inserts each participant into one of the processing queues. Under this insertion mechanism, participants are assigned low priority. Each processing queue managed by the sender process corresponds to an executor that runs in a separate physical server. Executing a participant means that it is checked to see if an action is due to be performed for that particular participant. As mentioned in section 3.1, there are several types of actions that can be executed for participants. For each participant, at most one action is executed before it is removed from the processing queue and has to wait for the next round of processing. Each processing queue has a thread pool associated with it on executor side.

There is a use case possible when the next action to be executed may need to occur after a certain period of time. This can happen, for example, when the next message should be sent to the participant two hours after the regular expression leading to it has been evaluated. In such case, the sender caches the time when the next action is due and refrains from inserting the participant into any of the processing queues before that time.

This caching mechanism is used to reduce the workload on the database server and improve processing performance and texting interactivity with users. During low activity hours, the average CPU consumption in main server is reduced from 80% to 5% when this cache mechanism is used since many queries are not sent to the database server for actions that are known to be due at a later time.

21. Receiver process

The receiver process is in charge of processing messages that are sent by users to the system. Two components intervene in the receiving process: 1) a web server and 2) a *receiver* process. The web server is used to receive messages from our data aggregators. Once a message sent by any of our users is received on the data aggregator side, an HTTP Post request is sent to our web server. Each request contains details such as originating phone number, content of the message, and time. The web server then stores the received message into the database. The receiver process takes each message received on the web server and associates that message with a participant in the database. This association is based on the originating phone number (i.e. the user's phone number). The receiver process checks whether the message is the reply to a previously sent poll message, whether or not the message corresponds to an on-demand message.

Depending on the outcome of this check three different scenarios might occur:

1. Received message is an on-demand message: In this case the participant has sent a message to the system that matches one of the registered on-demand keywords. These type of messages usually trigger a conversation between the system and the user. In such cases the system transports the user to another state machine. Once the on-demand state machine is traversed, the user is moved back to the main state machine.
2. Received message is a reply: For poll messages sent from the system an answer from the user is expected. For each poll message, a regular expression is used to define the format of all possible valid answers. For each message received from users that are not on-demand keywords, the receiver process checks whether the content of the message matches the regular expression specified for a previously sent poll. In case a match is found, the body of the message is associated to the poll for that particular user.
3. Received message is not an on-demand keyword nor a reply: In case the received message is not either an on-demand keyword or a reply to a poll message, the system sends an "Invalid message" text to the user.

Scenarios 1 and 2 are very likely to start an interactive session with the system. In order to maintain a high interactivity, once messages are processed by the receiver process, the system inserts the participant into one of the processing queues and assigns the maximum priority to that participant.

In addition to the previous tasks, the receiver process periodically (once per minute) contacts the data aggregators to check if any messages that were received by them were not received by the web server. Situations like this may arise due to network errors or any other issue causing a server downtime.

22. QUITXT as a use case: a smoking cessation protocol

QUITXT is a mobile text messaging service for young adult smokers aged 18-29 living in South Texas to assist them quit smoking . This service has been implemented as a case study of the described platform. The *Quitxt* system responds to text codes with a sequence of interactive messaging, beginning with collection of baseline data on demographics, cigarette smoking, e-cigarette use and binge drinking. Cessation is assessed with a texted question about abstinence from cigarette use approximately at one, three and seven months (222 days) following the quit date selected. After providing baseline data in reply to texted questions, *Quitxt* participants are prompted to either choose to "quit tomorrow" or to set a "quit date" within 14 days. The service then provides daily messages with expanded mobile content including links to nine mobile webpages with content appropriate for point-of-progress in the cessation process: (1) recommendation for use of nicotine replacement but discouraging e-cigarette use, (2) motivations for quitting, (3) obtaining social support, (4) avoiding binge drinking, (5) increasing physical activity, (6) breathing exercises for managing stress, (7) things to do instead of smoking (counterconditioning), (8) avoiding relapse by talking yourself out of smoking, and by (9) predicting, planning and practicing for difficult situations. Each of these mobile webpages contained educational tips and information, links to peer modeling videos and music with lyrics reinforcing the messages on each topic. The *Quitxt* texting system is designed to send prompts, motivational messages and messages about the nine content areas listed above, sequenced according to the enrollees' progress from pre-quitting preparation through initial and longer-term cessation. Participants can text specific code words when they need help, i.e., if they experience cravings, are in a bad mood, have slipped, or if they want to exit the program. It is split into stages that are displayed in Fig. 21. In each stage the users get different messages throughout the day.

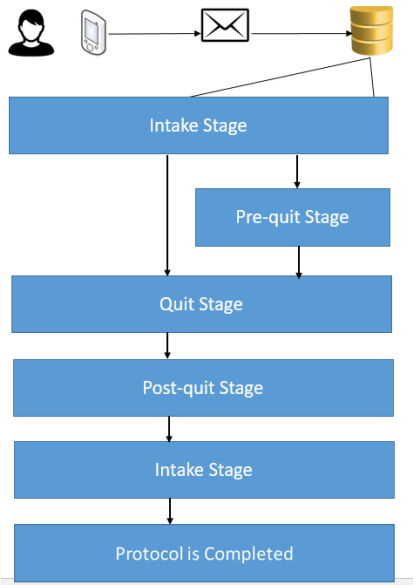

Figure 21. *Flow diagram for smoking cessation protocol*

Start/Intake: Initially when the participant wants to enroll into the system, he sends a text message with the enrollment keyword to a short code phone number which sets him in the intake stage of the state machine. This stage is a conversation that collects basic information such as gender, ethnicity, nicotine addiction, alcohol use, and so forth. Each poll message (i.e., question such as age) will be asked three times, with an interval of 24 hours between each repetition. At the end of the questionnaire, the user is given the choice to quit tomorrow or to go through a pre-quit stage.

Pre-Quit Days: Pre-quit days are the days between enrollment day and quit day. This period can last between 1 and 7 days as decided by the user. During the pre-quit stage, the participant receives motivational messages for successful quitting. After the stage is completed, participants are given the choice to be redirected to the quit stage of the state machine or to start the pre-quit stage again.

Quit days: The first month is the most important phase of the smoking cessation service (called here the quit stage). Participants can opt out of the system whenever they want to by texting EXIT as the keyword. The users holds conversations with the system every day during the quit stage. The system starts conversations with the user at times that the user has chosen. The user can also start conversations with the system by sending certain keywords, such as HELPNOW, or CRAVE. The wording in these conversations may vary for a more human interaction. Cessation is assessed at days 7 and 28 of the quit stage, participants are asked whether they have managed to remain smoke-free. After the stage is completed, the participant are redirected to the post-quit part of the state machine where follow-up assessment are repeated at 72, 32 and 222 days.

Post-quit days: Post-quit days are the days that last from days 29 to 222. This is the longest lasting stage, in which conversation with the user is intermittent and does not happen every day. After 222 days is passed, the program is completed.

23. Conclusion and Further Work

This paper proposes an automated messaging platform. The platform greatly enhances the capabilities of automated messaging. Designers can build very complex dialogue-tree conversations by interconnecting state machines and manipulating them at ease. Conversational logic blocks may be easily replicated in different parts of the system by implementing routines and the delivered content may be configured to change from one instance to another. Users may be migrated from one version of the conversation to another without even noticing. The system also offers on-demand conversations so that the users may trigger a specific conversation (i.e., a request for help) at any point without disrupting the principal conversation. The system is also very robust to some of the problems associated with text messaging. For example, the system may repeat a question if the user fails to answer or the answer is invalid. Any replies made by the user can also be changed later on. A simulation environment has also been built which allows administrators to quickly simulate long conversations, such as the 222 day long conversation used as a test case for the system. Messages from the user are provided to the simulation as an answer sheet. Administrators may also "jump" from one place of the conversation to another instantly. For future work, we wish to give the system the ability to map open-ended answers to multiple choice answers via natural language processing techniques.

References:

[1] B. Holtz, C. Lauckner, "Diabetes management via mobile phones: a systematic review," Telemedicine and e-Health, Vol. 18, No.3, 2012, pp. 175-184.

[2] R. Whittaker, H. McRobbie, C. Bullen, A. Rodgers, Y. Gu, "Mobile phone-based interventions for smoking cessation," Cochrane Database of Systematic Reviews, Issue 4. Art. No. CD006611, 2016. [Accessed October 2016].http://onlinelibrary.wiley.com/doi/10.1002/14651858.CD006611.pub4/epdf

[3] C. Free, G. Phillips, L. Watson, L. Galli, L. Felix, P. Edwards, V. Patel, A. Haines, "The effectiveness of mobile-health technologies to improve health care service delivery processes: a systematic review and meta-analysis," PLoS Med, Vol. 10, No.1, 2013, p.e1001363.

[4] Twilio messaging system. https://www.twilio.com/ . [Accessed October 2016].

[5] RapidSMS messaging system. https://www.rapidsms.org/. [Accessed October 2016].

[6] FrontlineSMS messaging system. http://www.frontlinesms.com/. [Accessed October 2016].

[7] Mobenzi Technologies. http://www.mobenzi.com/researcher/home. [Accessed October 2016].

[8] Magpi messaging system. http://home.magpi.com/ [Accessed October 2016].

[9] Trumpia messaging system. http://trumpia.com/ [Accessed October 2016].

[10] R.L. Siegel, E.L. Jacobs, C.C. Newton, D. Feskanich, N.D. Freedman, R.L. Prentice, A. Jemal, "Deaths due to cigarette smoking for 12 smoking-related cancers in the United States. JAMA internal medicine," Vol. 175, No.9, pp. 1574-1576.

[11] US Department of Health and Human Services. (2014). The health consequences of smoking—50 years of progress: a report of the Surgeon General. Atlanta, GA: US Department of Health and Human Services, Centers for Disease Control and Prevention, National Center for Chronic Disease Prevention and Health Promotion, Office on Smoking and Health, 17.

[12] Centers for Disease Control and Prevention. (2016). Smoking and Tobacco Use Fact Sheet: Current cigarette smoking among adults in the United States. Available https://www.cdc.gov/tobacco/data_statistics/fact_sheets/adult_data/cig_smoking/

[13] G.M.T. Ahsan, I.D. Addo, S. I. Ahamed, D. Petereit, S. Kanekar, L. Burhansstipanov, L. U. Krebs, "Toward an mHealth Intervention for Smoking," In Proc of COMPSAC 2013, doi: 10.1109/COMPSACW.2013.61.

[14] L.C. Abroms, M. Ahuja, Y. Kodl, L. Thaweethai, J. Sims, J. P. Winickoff, and R.A. Windsor, "Text2Quit: Results from a pilot test of a personalized, interactive mobile health smoking cessation program," Journal of Health Communication, Vol. 17, No.sup1, May 2012, pp. 44-53.

[15] F. Naughton, S. Cooper, K. Bowker, K. Campbell, S. Sutton, J. Leonardi-Bee, M. Sloan, T. Coleman, "Adaptation and uptake evaluation of an SMS text message smoking cessation programme (MiQuit) for use in antenatal care," BMJ Open 2015; 5: e008871. doi:10.1136/bmjopen-2015-008871.

[16] Quitplan service. https://www.quitplan.com/ [Accessed October 2016].

[17] Introduction to ChoiceScript, Choice Of Games. https://www.choiceofgames.com/make-your-own-games/choicescript-intro/. [Accessed October 2016].