EECS4312 Messenger Project

Randy Agyapong (randya25@my.yorku.ca)

December 1, 2016

Prism account used for submission: randya25

Requirements Document:
Messenger System

Contents

List of Figures

List of Tables

1 System Overview

The System Under Development (SUD) is a secure messenger system. The messenger system supports the ability for users to send text messages.

This requirements document contains all the requirements needed to build a secure messenger system. There are users, (e.g. doctors, nurses, administrators) that can send text messages, and there are groups (e.g. cardiology, nephrology, endocrinology) that users can be a part of. A user can register to become a member of any group, and they can send and read messages. A user within a group can send a message to other members of that group. The main concern in this messenger system is privacy. Users may only access/read a message from a group that they are registered in. Accessing messages from a group that a user is not registered in is prohibited.

2 Context Diagram

The System Under Description (SUD) is a messenger system. The Administrator, the messenger system, and the output display are all under the environment of the messenger system. The monitored commands and controlled variables for the messenger can be found in Table 1 and Table 2 respectively. The primary actor is the administrator or user. These names are used interchangeably. See figure 1 for the context diagram of the system.

3 Goals

The high-level goals (G) of the system are:

- G1—Users can register to a group to read messages
- G2—Users can send and read messages to/from other users of the same group
- G3—Privacy is maintained, a user cannot read a message they do not have access to

4 Monitored Commands

The monitored commands are issued by the administrator. Each command interacts with the system. See Table 1 for the full list of monitored commands and what each of them do.

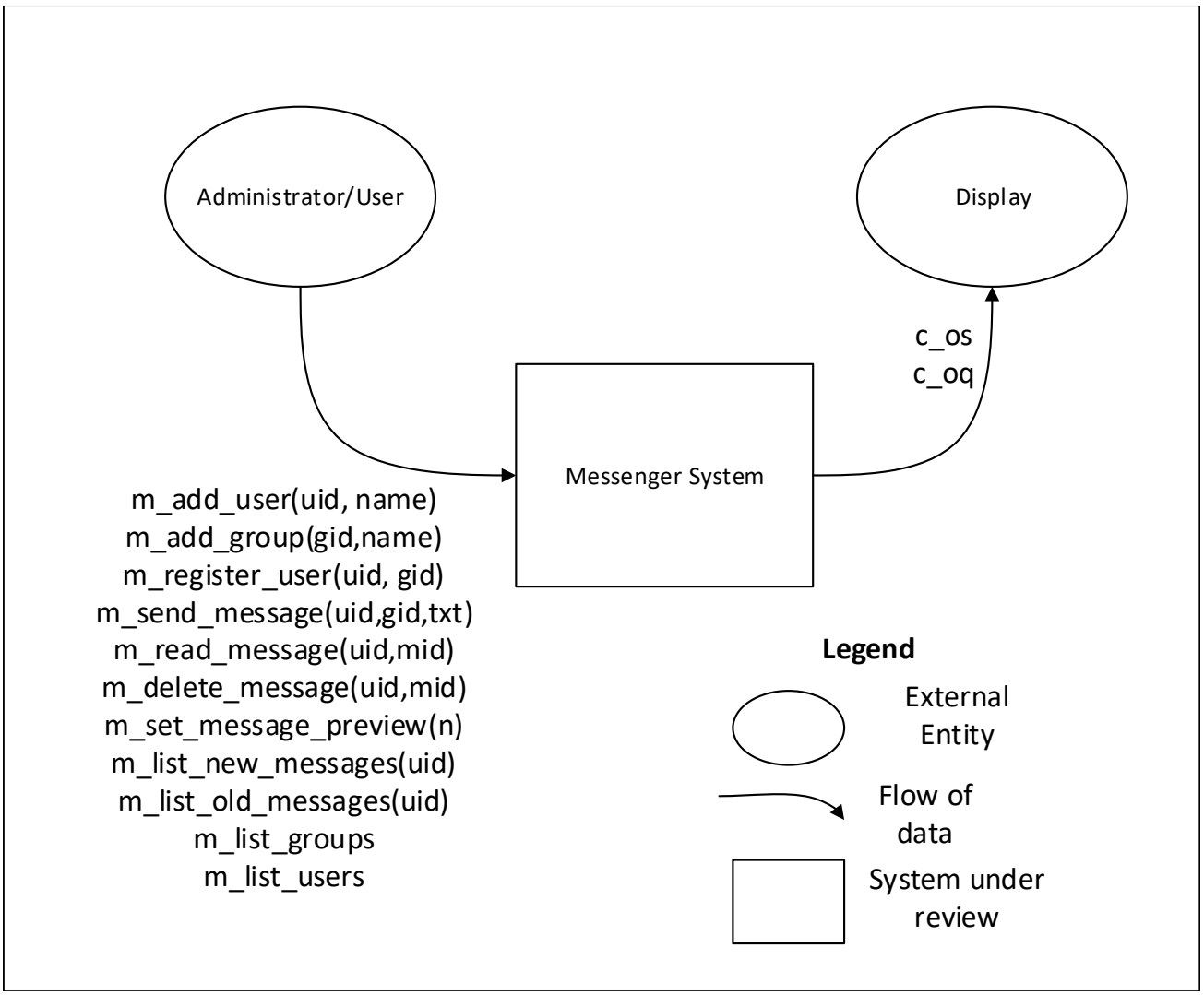

Figure 1: Context diagram of the Messenger System

5 Controlled Variables

The controlled variables are the output status of each command inputted into the system. Table 2 list all the controlled variables. c_os is a controlled variable for the output status of the system. The type of error it outputs can be found in section 9. c_oq is another controlled variable for the output query. It is non-empty when the administrator issues a query command, and empty otherwise. See the function tables in section 8 for the full range of outputs possible for c_oq.

Name	Physical Interpretation
m_add_user(uid,name)	add new user to the system
m_add_group(gid,name)	add new group to the system
m_register_user(uid,gid)	register a user to a group
m_send_message(uid,gid,txt)	send a message to the group members
m_read_message(uid,mid)	read a message
m_delete_message(uid,mid)	delete an old/read message
m_set_message_preview(n)	set message preview length
m_list_new_messages(uid)	list user's new messages
m_list_old_messages(uid)	list user's old messages
m_list_groups	list groups
m_list_users	list users

Table 1: Monitored variables of Messenger System

Name	Type	Range	Physical Interpretation
c_os	ENUMERATED	See error list section 9	Output status of the system
c_oq	TEXT	show_msg, list_msg, show_msg, ϵ	Output query by the user. It can be empty if a monitored command was a query

Table 2: Controlled Variables

6 E/R-descriptions

The requirements for the Messenger system are as follows:

REQ1	The messenger system shall process each monitored command as found in table 1	See the monitored variables in Table 1.

REQ2	An error shall occur and shall be stated in the output if the monitored command parameters are incorrect, or if the command can not be processed.	See Section 9 for the full list of errors.

REQ3	A user shall neither send nor access a message they do not have access to.	It ensures the privacy of the system is maintained. The PVS proof in the Appendix ensures this invariant is maintained. Also see the function tables for these preconditions.

REQ4	An administrator shall be able to query the messenger system for user/group information	See Monitored variables for the type of commands for queries. These include *list_new_messages* and *list_groups*

REQ5	The messenger system shall not process multiple commands simultaneously.	Only one command can be sent at a time instance. The function table demonstrates that each command is treated as a case, where if one command occurs, all other commands do not occur.

REQ6	The messenger system shall store new users and new groups into the system, and ensure they have unqiue IDs.	See the function Tables 10 and 11 . *add_user* and *add_group* have a precondition where the ids must not already be in the system, else an error occurs.

REQ7	Each sent message shall be given a unique message ID to distinguish them from other messages.	See Function Tables

The following environmental assumptions are made:

ENV8	The MID, UID, n, and MID parameters for the monitored commands are positive natural numbers.	This simply makes the system easier to read and understand, and it ensures that each number can be unique.

ENV9	The NAME and TEXT in the parameters for the monitored commands are a sequence of characters (String)	These sequence of characters are denoted in most programming languages as a string. The messenger system can then produce an error if these strings are empty.

ENV10	There are no additional monitored commands	The system may not be able to process commands that it does not know about, or commands that are not exactly as the same format as the monitored commands.

7 Abstract variables needed for the Function Table

In the tables below, in order to completely define each function table, these set of variables/functions are defined in Table 3. The Messager system also consists of an

Name	Type	Physical Interpretation
UID	$\mathbb{N}+$	A user ID
MID	$\mathbb{N}+$	A message ID
GID	$\mathbb{N}+$	A group ID
i	DTIME*	Time for a computer to process a command
TEXT	A sequence of characters (String)	Text sent or read in messages
NAME	A sequence of characters	Name of a group or user
INT	INTEGER	An integer
MSG_STATE	{read, unread, unavailable}	The state of the message. A message could be read, unread, or unavailable

Table 3: Variable definitions for fucntion tables and abstract variable states

abstract variable state that changes the internal state with each monitored command. This can be seen in Table 4.

Table 7 is an internal variable for the monitored command m_send_message. When a user sends a message, this variable makes sure that the users who have access to that message are marked as 'read', and those who do not have access to the variable are marked as 'unavailable'. An unavailable message means the user is permitted from reading that file.

ABSTRACT_STATE		
Variable	Type	Description
users(i)	SET[UID]	set of user ids
name(i)	[(users(i)) → NAME]	name associated with an account id
groups(i)	SET[GID]	set of group ids
gname(i)	[(groups(i)) → NAME]	name associated with a group id
msgs(i)	SET[MID]	set of message ids
membership(i)	(users) ↔ (group)	membership between user id and group id
info(i)	(msg) → MSG_INFO[(user),(groups)]	message info associated with a message id
ms(i)	(user)x(msgs) → MSG_STATE	the state of the message associated with a user's message

Table 4: Abstract variable state of the Messenger system

Abstract State : MSG_INFO		
Variable	Type	Description
sender	UID	the sender of the message
recip	GID	the group recipient of the message
content	TEXT	the message contents

Table 5: Abstract variable state of MSG_INFO

Internal Variable msgOf?(uid : UID, mid : MID)	
	msgOf?
mid ∈ msgs_−1 ∧ membership$_{-1}$(uid, info(mid)'recip)	TRUE
¬ (mid ∈ msgs_−1 ∧ membership$_{-1}$(uid, info$_{-1}$(mid)'recip)	FALSE

Table 6: Internal Variable: msgOf?, used to determine if there is a membership between a message and an id.

select_mailbox(uid: UID, gid : GID)	
(uid, gid) ∈ membership$_{-1}$	'read'
(uid, gid) ∉ membership$_{-1}$	'unavailable'

Table 7: Internal variable select_mailbox

8 Function Table

Here are the function tables that correspond to each of the monitored events. They will be split into two sections. One section consists of tables at time interval i=0. These tables will define the contents of the abstract state at i = 0, where a single command that can only be used at the beginning. This is described in table 8. In the next section, the function tables apply only to those minotored commands when the time interval i >0. The overall function Table 9 describes what commands are available at each time interval.

8.1 At time interval i=0

ABSTRACT STATE at i = 0		
i	Variable	Value
i = 0	users(i)	$\varnothing$
	name(i)	$\varnothing \mapsto$ NAME
	groups(i)	$\varnothing$
	gname(i)	$\varnothing \mapsto$NAME
	msgs(i)	$\varnothing$
	membership(i)	$\varnothing \leftrightarrow \varnothing$
	info(i)	$\varnothing \mapsto$ MSG_INFO
	c_os(i)	ϵ
	c_oq(i)	ϵ

Table 8: Values of the abstract variables at i = 0; the initial state

8.2 Function Tables when i >0

Here are the function commands when i is greater than 0. Note that the monitored commands for set_message_preview, list_users and list_groups are purposely withheld, as they are trivial and are not necessary to prove the overall system for completeness, disjointness, and well-definedness.

i	Monitored Events
i = 0	nothing
i >0	m_add_user(uid : UID, n : NAME)
	m_add_group(gid : GID, n : NAME)
	m_register_user(uid: UID, gid : GID)
	m_send_message(uid : UID, gid : GID, txt : TEXT)
	m_read_message(uid: UID, mid : MID)
	m_delete_message(uid: UID, mid: MID)
	m_set_message_preview(n : INT)
	m_list_new_messages(uid : UID)
	m_list_old_messages(uid: UID)
	m_list_groups
	m_list_users

Table 9: Overall function table of the Messenger system

m_add_user(uid : UID, n : NAME)		
	uid $\in$ users$_{-1}$	$\neg$uid $\in$ users$_{-1}$
users	NC	{uid} $\cup$ users$_{-1}$
names		names$_{-1}$ $\restriction$ uid $\mapsto$ n
membership		NC*
ms		ms$_{-1}$ ({uid} x msgs) x {'unavailable'}
c_os	ERROR	OK

Table 10: Function table of m_add_user

m_add_group(gid : GID, n : NAME)(i)			
i	abstract variable	gid $\in$ groups$_{-1}$	gid $\notin$ groups$_{-1}$
i >0	groups	NC	{gid} $\cup$ groups$_{-1}$
	gnames	NC	gnames$_{-1}$ $\restriction$ gid $\mapsto$ n
	membership	NC	NC
	c_os	ERROR	OK

Table 11: Function table of m_add_group

9 Implementation Tables: Errors

From tables 18 to 24 are error messages for each command in the abstract table of Messenger. These error messages will be displayed to the administrator in the controlled variable c_os (output_status). It is essential that the administrator knows why his input leads to an error. The commands list_user and list_groups do not have error messages

m_register_user(uid : UID, gid : GID)(i)		
	uid ∈ $users_{-1}$ ∧ gid ∈ $groups_{-1}$ ∧ (uid, gid) ∉ $membership_{-1}$	uid ∉ $users_{-1}$ ∨ gid ∉ $groups_{-1}$ ∨ (uid, gid) ∈ $membership_{-1}$
membership	({uid}, {gid}) ∪ $membership_{-1}$	NC
ms	ms_{-1} ↾ ({uid} x msgs) x {'unavailable'}	
c_os	OK	ERROR

Table 12: Function table for m_regiser_user

m_send_message(uid: UID, gid: GID, txt: TEXT) (i)		
	uid ∈ $users_{-1}$ ∧ gid ∈ $groups_{-1}$ ∧ (uid,gid) ∈ $membership_{-1}$	uid ∉ $users_{-1}$ ∨ gid ∉ $groups_{-1}$ ∨ (uid,gid) ∉ $membership_{-1}$
msgs	∃ mid ∉ $msgs_{-1}$: {mid} ∪ $msgs_{-1}$	NC
ms	∃ mid ∉ $msgs_{-1}$: {uid}x{mid} ↦ {'read'} ∧ select_mailbox(uid,gid)(mid)	NC
info	∃ mid ∉ $msgs_{-1}$: {mid} ↦ (sender ↦ mid, recip ↦ gid, content ↦ txt)	NC
c_os	OK	ERROR

Table 13: Function table for m_send_message

m_read_message(uid : UID, mid : MID) (i)		
	uid ∉ $users_{-1}$ ∨ ¬msgOf?(uid, mid)	uid ∈ $users_{-1}$ ∧ msgOf?(uid, mid)
ms	NC	ms_{-1}↾({uid} x {mid} ↦{'read'})
c_os	ERROR	OK
c_oq	ϵ	show_msg(uid,mid, st(i)‘info(mid)‘content)

Table 14: Function table for m_read_message

associated with them. The administrator may be notified if there are no users or groups in the system.

m_delete_message(uid : UID, mid : MID) (i)		
	id $\notin$ users$_{-1}$ $\vee$ $\neg$ msgOf?(uid,mid)	id $\in$ users$_{-1}$ $\wedge$ msgOf?(uid,mid)
ms	NC	ms$_{-1}$ $\upharpoonright$ (uid x mid) $\mapsto$ {'unavailable'}
c_os	ERROR	OK

Table 15: Abstract command for m_delete_message

m_list_new_messages (uid: UID) (i)		
	uid $\in$ users$_{-1}$	uid $\notin$ users$_{-1}$
c_os	OK	ERROR
c_oq	list_msg(uid, {m : (msgOf(uid)) |ms$_{-1}$(uid,m) = {'unread'}}	ϵ

Table 16: Function table for m_list_new_messages

m_list_old_messages (uid: UID) (i)		
	uid $\in$ users$_{-1}$	uid $\notin$ users$_{-1}$
c_os	OK	ERROR
c_oq	list_msg(uid, {m : (msgOf(uid)) |ms$_{-1}$(uid,m) = {'read'}}	ϵ

Table 17: Function table for m_list_old_messages

10 Validation

The PVS proofs in the apendix shows the completeness, disjointness, and well-definedness of the function tables for each monitored variable. Here is the proof:

The function tables are disjoint because at no period of time can an input be in more than 1 row on the table. This avoids inconsistencies that may have resulted in different and possibly contradictory behaviours.

The invariants also hold. In reference to requirement 3, a user cannot access or send a message that they are not a group of. These invariants can also be found in the submission folder, which include all the PVS files. Here is one of the crucial invariants in PVS:

```
inv1 (s : STATE) :
  bool = FORALL (u:(s`users),m:(s`msgs)):
   s`ms(u,m) /= unavailable
    IMPLIES readership((s`membership),(s`info))(u,m)
```

This means that for all users and messages, if the messages for that user is available ie. read or unread, this implies that the the group that the user is apart of is the same as the recipient of those messages. Another invariant for the system in code is:

Errors: m_add_user(uid: UID, n:NAME)		
Condition	c_os	Description
uid ≤ 0	error1	ID must be a positive integer
uid $\in$ users$_{-1}$	error2	ID already in use
n is empty or does not start with a letter	error3	User name must start with a letter
ELSE	OK	OK

Table 18: Error messages for command m_add_user

Errors: m_add_group(gid: GID, n:NAME)		
Condition	c_os	Description
gid ≤ 0	error4	ID must be a positive integer
gid $\in$ groups$_{-1}$	error5	ID already in use
n is empty or does not start with a letter	error6	Group name must start with a letter
ELSE	OK	OK

Table 19: Error messages for command m_add_group

Errors: m_register_group(uid: UID, gid: GID)		
Condition	c_os	Description
gid $\leq 0 \vee$ uid ≤ 0	error7	ID must be a positive integer.
uid $\notin$ users$_{-1}$	error8	User with this ID does not exist.
gid $\notin$ groups$_{-1}$	error9	Group with this ID does not exist.
(uid, gid) $\in$ membership$_{-1}$	error10	This registration already exists
ELSE	OK	OK

Table 20: Error messages for command m_register_user

```
inv2 (s : STATE) :
bool = FORALL (m:(s`msgs)):
  s`membership( s`info(m)`sender,s`info(m)`recip )
```

This means that for all messages, there is a membership between the sender and the recipient.

Errors: m_send_message(uid: UID, gid: GID; txt: TEXT)		
Condition	c_os	Description
gid $\leq 0 \vee$ uid ≤ 0	error11	ID must be a positive integer.
uid $\notin$ users$_{-1}$	error12	User with this ID does not exist.
gid $\notin$ groups$_{-1}$	error13	Group with this ID does not exist.
txt is empty	error14	A message may not be an empty string.
(uid, gid) $\notin$ membership$_{-1}$	error15	User not authorized to send messages to the specified group.
ELSE	OK	OK

Table 21: Error messages for command m_send_message

Errors: m_read_message(uid: UID, mid: MID)		
Condition	c_os	Description
gid $\leq 0 \vee$ mid ≤ 0	error16	ID must be a positive integer.
uid $\notin$ users$_{-1}$	error17	User with this ID does not exist.
mid $\notin$ msgs$_{-1}$	error18	Message with this ID does not exist.
msgOf?(uid,mid)	error19	User not authorized to access this message.
ms_{-1}(uid,mid) $\mapsto$ read	error20	Message has already been read. See 'list_old_messages'.
ELSE	OK	OK

Table 22: Error messages for command m_read_message

Errors: m_delete_message(uid: UID, mid: MID)		
Condition	c_os	Description
gid $\leq 0 \vee$ mid ≤ 0	error21	ID must be a positive integer.
uid $\notin$ users$_{-1}$	error22	User with this ID does not exist.
mid $\notin$ msgs$_{-1}$	error23	Message with this ID does not exist.
ms_{-1}(uid,mid) = read $\vee$ ms_{-1}(uid,mid) = unavailable	error24	Message with this ID not found in old/read messages
ELSE	OK	OK

Table 23: Error messages for command m_delete_user

Errors: m_set_message_preview(n: INT)		
Condition	c_os	Description
$n \leq 0$	error25	Message length must be greater than zero.
ELSE	OK	OK

Table 24: Errors messages for command m_set_message_preview

Errors: m_list_new_messages(uid: UID)		
Condition	c_os	Description
$uid \leq 0$	error26	ID must be a positive integer.
$uid \notin users_{-1}$	error27	User with this ID does not exist.
ELSE	OK	OK

Table 25: Error messages for command m_list_new_messages

Errors: m_list_old_messages(uid: UID)		
Condition	c_os	Description
$uid \leq 0$	error28	ID must be a positive integer.
$uid \notin users_{-1}$	error29	User with this ID does not exist.
ELSE	OK	OK

Table 26: Error messages for command m_list_old_messages

The function tables are all non-circular because at no point does an abstract varaible or controlled variable change and read at the same time. In each function table, the state of the variables in the previous time instance are observed, and then are changed in the current state.

11 Use Cases

11.1 Use Case: Adding and registering users

This use case decribes the normal operation of adding users and groups, and registering users to groups

- Related System Goals: G3
- Primary Actor: Administrator
- Precondition:
  - Messenger system is ready to recieve commands
  - There are no users or groups in the system
- Postcondition:
  - Users and groups are added into the system
  - Users are registered to groups
- Main Success Scenario:
  1. Administrator adds user "Dave" into the system through command *add_user*(1,"Dave")
  2. Messenger system outputs "OK"
  3. Administrator adds user "Joe" into the system through command *add_user*(2,"Joe")
  4. Messenger system outputs "OK"
  5. Administrator adds user "Ashley" into the system through command *add_user*(3,"Ashley")
  6. Messenger system outputs "OK"
  7. Administrator adds group "Nurses" into the system through command *add_group*(1,"Nurses")
  8. Messenger system outputs "OK"
  9. Administrator registers user "Dave" to group "Nurses" through command *register_group*(1,1)
  10. Messenger system outputs "OK"

11.2 Use Case: Sending and reading messages

This use case decribes the normal operation of sending messages and reading messages.

- Related System Goals: G1 and G2
- Primary Actor: Administrator
- Precondition:
  - Users and groups are added in the system
  - Some users are registered to some groups
- Postcondition
  - Messages are sent from one user to another
- Main Success Scenario:
  1. Administrator issues command *send_message*(1,1, "Hello everyone!") which sends a message from user "Dave" to group "Nurses"
  2. Messenger system outputs "OK". Message is assigned an ID
  3. Administrator lists new messages for user "Ashley" using the command list_new_messages(3)
  4. Messenger system outputs "OK" and lists new messages, including the message "Hello everyone!"
  5. Administrator issues command *read_message*(3,1). User "Ashley" reads new message
  6. Messenger system outputs "OK" and outputs the message
  7. Administrator deletes message with MID 1 from Ashley's old/read messages with the command *delete_message*(3,1)
  8. Messenger system outputs "OK"

11.3 Use Case: Handling Privacy

This use case decribes an operation of users sending messages to groups they are not registered in. It also decribes when a user reads a message they have no access to.

- Related System Goals: G1, G2 and G3
- Primary Actor: Administrator

- Precondition:
  - Users and groups are added in the system
  - Some users are registered to some groups but not to others
- Postcondition
  - Messages are sent from one user to another
  - Messenges of some users are read
- Main Success Scenario:
  1. Administrator issues command *send_message*(1,2, "Hello everyone!") which sends a message from user "Dave" to group "Staff"
  2. Messenger system outputs "error" with error message "User not authorized to send messages to the specified group."
  3. Adminstrator registers user "Dave" to group "Staff" with command *register_user*(1,2)
  4. Repeat step 1
  5. Messenger system outputs "OK". Message is assigned an MID
  6. Administrator issues command *read_message*(4,1) that makes user "Katie" of UID "4" read a message
  7. Messenger system outputs "error" with error message "User not authorized to access this message."

12 Acceptance Tests

In this section, the use cases have been converted into precise acceptance tests. See Table 27 for the full list.

	Acceptance Test
Use Case 1	at1.txt, at1.expected.txt
Use Case 2	at2.txt, at2.expected.txt
Use Case 3	at3.txt, at3.expected.txt

Table 27: Acceptance tests pertaining to each Use Case

13 Traceability

Table 28 shows which acceptance tests have passed, and which R-descriptor they have checked.

	R1	R2	R3	R4	R5	R6	R7
at1.txt	✓				✓	✓	
at2.txt	✓	✓		✓	✓		✓
at3.txt	✓		✓		✓	✓	✓

Table 28: Traceability for acceptance tests and R-descriptors

14 Appendix

All PVS proofs and code are in the submission folder, as well as the precise acceptance tests.