EECS4312 Messenger Project

Parastoo Baghaei (paras273@cse.yorku.ca)
Bhumika Patel (bhumika8@my.yorku.ca)

December 1, 2016

You may work on your own or in a team of no more than two students. **Submit only one document under one Prism account.**

Prism account used for submission:paras273@cse.yorku.ca

Keep track of your revisions in the table below.

Revisions

Date	Revision	Description
19 November 2016	1.0	Initial requirements document
24 November 2016	2.0	In progress requirements document
28 November 2016	3.0	In progress requirements document
29 November 2016	3.0	Updated after feedback from the customer

Requirements Document:
Secure Messenger

Contents

List of Figures

List of Tables

1 System Overview

The System Under Development (SUD) is messenger service. Messenger allows users to have different groups for communication and securely send and receive messages within their group. The messenger can be base of so many application and services we use on our smartphones and PCs.
The purpose of the messenger is allow users join different groups and all the messages they send or received will stay within the group.
This requirements document is especially for simple operations of messenger such as

1. add a new user to the system
2. add a new group to the system
3. register a user in a group
4. send a message to the group members
5. read a new message
6. delete an old/read message
7. sets length (characters) for message preview
8. list user's new messages
9. list user's old/read message
10. list all groups in alphabetical order
11. list all users in alphabetical order

We will discuss the updates that needed to be made in case of any of the available commands.

2 Use Case Diagram

The system won't differentiate between users, meaning any user has equal access to any of the use cases. In case the user has not been added to the system yet, the action won't take place and would return an error (See the error message table 20.)

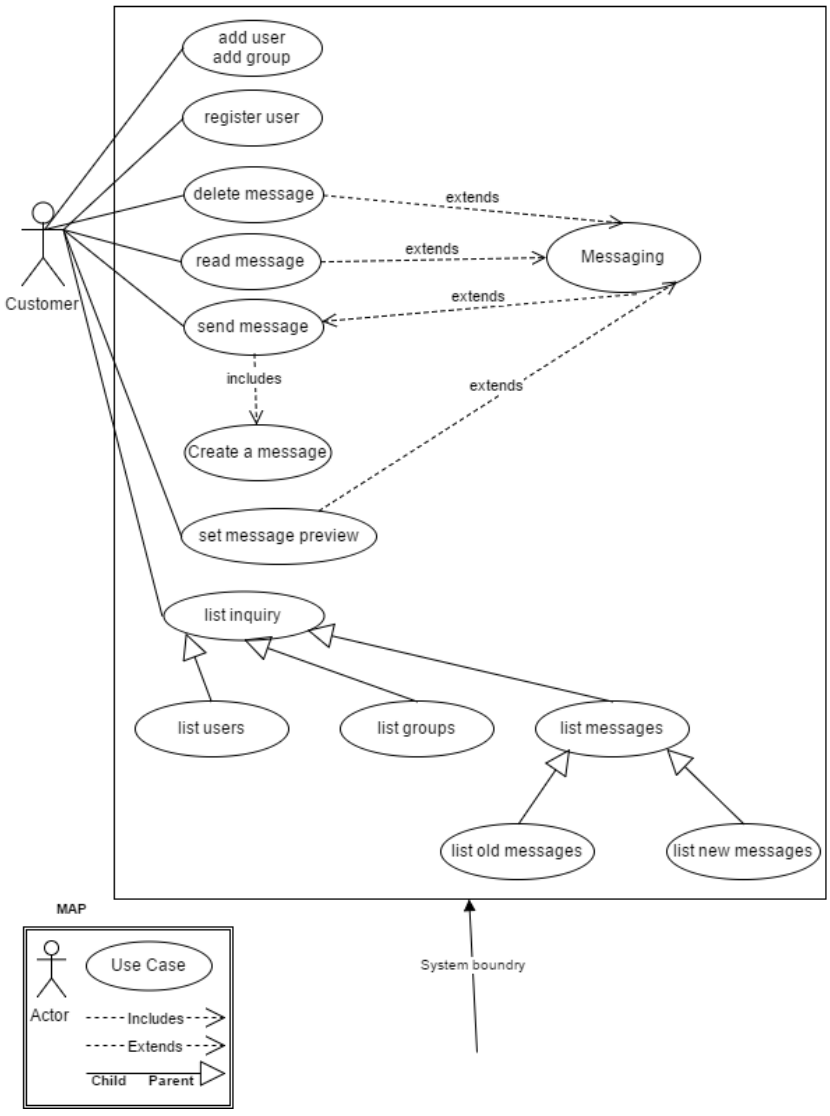

Figure 1: Use Case Diagram

3 Goals

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

- G1—Development of secure messenger system.
- G2—Restricted access to features such as registering in groups, sending, deleting and reading and listing messages to registered users.
- G3—Adding groups, setting message preview and listing groups and users can be done by none registered users, which doesn't threaten the system security.
- G4—The users must be able to send messages to groups they are registered in. Only members of that group can see the message. All the members of a group should be able to see a message that is sent to that group.

4 Monitored Variables

Monitored variables in case of business systems (not safety critical systems) can be represented in form of monitored events, because these systems are event oriented. In messenger a change only happens in when user enters a command (takes an action).

Table 1: System Types

	Type	**Description**
GID	INT	group id
UID	INT	user id
MID	INT	message id
USER	SRTING	user name
GROUP	STRING	group name
TEXT	STRING	message body

Table 2: Monitored Events

Event	Description
nothing	this is the event when user does nothing
add_user (u:UID,un:USER)	this event is adding a new user to system
add_group (g:GID,gn:GROUP)	this event is true when user is adding a new group
register_user (u:UID,g:GID)	this command is the event of user is getting assigned to a group
send_message (u:UID,g:GID,txt:TEXT)	even of a user sending a message
read_message (u:UID,m:MID)	this event is happening when a user reads a message
delete_message (u:UID,m:MID)	this event is happening when a user, u deletes a message with id m
set_message_preview (n: INT)	this event is happening when the user wants to set the length of message previewed. default value is 15 character
list_new_messages (u:UID)	this event happens when user wants to see a set of unread messages he received
list_old_messages (u:UID)	this event happens when user wants to see a set of read messages he received
list_gropus	this event happens when user wants to see a set of all groups
list_users	this event happens when user wants to see a set of all users (including himself)

5 Controlled Variables

The controlled variables can be extracted from the expected output given for a particular acceptance test in addition to the grammar given.

MSG_STATUS and MSG_INFO are both defined types to store some information about every message without populating the controlled variable tables. In PVS MSG_STATUS is a Enumeration Type and MSG_INFO is a Record Type.

Table 3: Message Status

MSG_STATUS
read
unread
unavailable

Table 4: Message Info

MSG_INFO [U, G : TYPE]
sender ∈ U
recipient ∈ G
content ∈ TEXT

Table 5: State Record

STATE		
Record Variable	**Type**	**Description**
users(i)	$SET[UID]$	set of user ids
names(i)	$[(users) \rightarrow USER]$	set of (id,user name) pairs
groups(i)	$SET[GID]$	set of group ids
gnames(i)	$[(groups) \rightarrow GROUP]$	set of (id, group name) pairs
msgs(i)	$SET[MID]$	set of message ids
membership(i)	$SET[[(users), (groups)]]$	set of all users mapped to all the groups which they are part of set of all the user-group relations
info(i)	$[(msgs) \rightarrow$MSG_INFO$[(users), (groups)]]$	set of all the messages mapped to their info. a message info is its sender user and recipient groups. Refer to table 4
ms(i)	$[[(users), (msgs)] \rightarrow$MSG_STATE]	set of all the messages of all the users or also called (user,message) pairs mapped to the message state which can be read, unread,unavailable. Refer to table 3

At the beginning, the state is initialized.

Table 6: Initial State

INITIAL STATE	
users	{}
names	{} → USER
groups	{}
gname	{} → GROUP
msgs	{}
membership	{}
info	{} → MSG_INFO
ms	{} × {} → MSG_INFOMSG_STATE

Table 7: Output Datatype

OUTPUT DATATYPE	**Description**
ok	When everything is okay
error	In case of error
list_msg (id: UID,ms: set[MID])	When queries list_old_messages and list_new_messages are called
show_msg (id: UID,msg:MID,txt: TEXT)	When user wants to view an specific message

Table 8: Controlled Variables

Variable	**Type**	**Description**
st(i)	STATE	Returns state of the system, which includes all the variables in table 5 as the output
output(i)	OUTPUT	Returns one of the four possible options in table 7 as output

6 E/R-descriptions

6.1 R-descriptions

REQ1	Users may become members of any number of groups.	See the definitions[1]

Rationale: This is according to the definitions. As long as a group exists no restrictions apply when users of the messenger request joining.

REQ2	Users may send messages only to the members of a group they are registered in.	This is to keep the system secure.

Rationale: The message a user sends in only relevant to a group he is registered in. If user wants to send message to another group he can always join that group first (REQ1) This requirement also prevents spamming the groups.

REQ3	Users may only access/read a message from a group they are registered in.	Privacy of a messenger relies on this

Rationale: Messenger is only secure if the messages are secure meaning that they can only be seen by the users who are audience of the message. If a user is interested in messages of other group they can register in that group.

REQ4	Once a message is read by the recipient, its status changes from new to old.	See the `messenger.definitions.txt`

Rationale: Differentiating between old an new messages ables the users to stay updated with possibly lots of messages from different groups they are registered in.

REQ5	Users may delete their old messages.	Old message is a message that has already been read.

Rationale: Deleting messages the have been already read can help ease navigation through old messages and help staying up to date with new messages.

REQ6	The system provides queries List of users ($ID- > name$) and list of groups ($ID- > name$), both sorted by name	This is a customer request.

Rationale: Alphabetic sorting means easier search and navigation.

REQ7	Lists of new and old messages for a user, are sorted by message ID.	This is a customer request.

Rationale: For navigation and search purposes.

REQ8	Components of the abstract state are always sorted by ID, except for list of users, or lists of groups (as mentioned in REQ7).	This is according to customer need.

Rationale: For navigation and search purposes.

REQ9	Each user command shall be followed by printing the abstract state space.	This is according to customer need.

Rationale: To make it easier for the user to stay updated with the system.

REQ10	Queries shall display results only, not the abstract state.	Refer to table 2, the queries list specific sets of information.

Rationale: Queries do not update the abstract state, therefore there is no need to show the abstract state after queries.

REQ11	The system shall set the default message length(number of characters) to 15 . The user may change the message length previewed.	This is a customer request.

Rationale: In real life the text messages can be very long e.g. hundred or thousands of characters. The system truncates the message to the default of 15 characters length, which can be changed by users.

REQ12	The system shall return the correct error message, one at instance of an error, according to error priority.	See table 20 for error messages and their precedence given by customer.

Rationale: Notifies user with proper error message with highest precedence.

REQ13	System only allows users and groups get added only with a unique positive ID.	This makes the system reliable and secure.

Rationale: Without having unique ID there is really no way to diffrenciate between users and groups in our system.

REQ14	System only allows users and groups with valid names to get added.	Any string that starts with a letter is a valid name.

Rationale: Restricting names can eliminate future errors that might happen because of special characters.

REQ15	System assigns a unique ID to every new non-empty message that gets sent.	This makes it possible to keep track of messages for different users. Empty messages are not allowed in the system.

Rationale: We need ID to keep track of messages for different users, however there is no need for user to provide this because system can assign numbers in creation process.

6.2 E-Descriptions

ENV16	The command user enters are limited to Monitored Events, one at any given time.	See Monitored Events table2.

Rationale: Without putting limitations on monitored events, there is infinitely many possibility.

ENV17	The command user enters is grammatically correct, correctness is evaluated according to definitions given by customer.	Refer to `messenger.definitions.txt`.

Rationale: There is no complexity in elimination of grammar errors.

ENV18	Commands and their inputs are type correct.	Refer to table 1

Rationale: There is no complexity in elimination of grammar errors.

7 Abstract variables needed for the Function Table

Abstract variables are not needed.

8 Function Table

For simplicity and clarity, only the controlled variables of the STATE (table 5) that get updated by the command issued are shown in the command function table. Any variable of STATE record that is not in the table, implies the value of that variable is unchanged (same as its value at previous time instant).

8.1 Function table for Add User: add_user

Table 9: Add User Function Table

add_user (u: UID, un: USER)		**u** $\in$ **users** $_{-1}$	**u** $\notin$ **users** $_{-1}$
i = 0		initial values [2]	initial values
i >0	*users*	NC	{u} $\cup$ users $_{-1}$ [3]
	names	NC	names $_{-1}$ $\restriction$ u $\rightarrow$ un [4]
	membership	NC	NC_* [5]
	ms	NC	ms $_{-1}$ $\restriction$ ({u} $\times$ msg) $\times$ {unavailable}
	output	ERROR	OK

8.2 Function table for Add Group: add_group

Table 10: Add Group Function Table

add_group (g: GID, gn: GROUP)		**g** $\in$ **groups** $_{-1}$	**g** $\notin$ **groups** $_{-1}$
i = 0		initial values	initial values
i >0	*groups*	NC	{g} $\cup$ groups $_{-1}$
	gnames	NC	gnames $_{-1}$ $\restriction$ g $\rightarrow$ gn
	membership	NC	NC_*
	output	ERROR	OK

[1] messenger.definitions.txt

[2] Refer to table 6

[3] This notation simply refers to the users set at i-1 instant of time.

[4] The $\restriction$ notation is same as WITH notation in set theory, overwriting the previous set

[5] Eventhough the users set changed the new user is not assigned to any group yet, therefore membership is unchanged.

8.3 Function table for Registering user to group: register_user

Table 11: Registering user Function table

register_user (u: UID, g: GID)		**u** $\notin$ **users** $_{-1}$ $\vee$ **g** $\notin$ **groups** $_{-1}$ $\vee$ **membership** $_{-1}$ **(u,g)**	**u** $\in$ **users** $_{-1}$ $\wedge$ **g** $\in$ **groups** $_{-1}$ $\wedge$ $\neg$ **membership** $_{-1}$ **(u,g)**
i = 0		initial values	initial values [6]
i >0	*membership*	NC	membership $_{-1}$ $\restriction$ u $\rightarrow$ g
	output	ERROR	OK

8.4 Function table for Send Message: send_message

Table 12: Send Message Function Table

read_message (u: UID, m: MID)		**u** $\notin$ **users** $_{-1}$ $\vee$ $\neg$**g** $\in$ **groups** $_{-1}$ $\vee$ $\neg$ **mem** $_{-1}$ **(u,g)**	**u** $\in$ **users** $_{-1}$ $\wedge$ **g** $\in$ **groups** $_{-1}$ $\wedge$ **mem** $_{-1}$ **(u,g)**
i = 0		initial values	initial values
i >0	msgs		mem $_{-1}$ union m1[7]
	info		info $_{-1}$ $\restriction$ m1 $\rightarrow$ [u,g,txt]
	ms	NC	ms $_{-1}$ $\restriction$ {m1} $\times$ (users $_{-1}$) $\rightarrow$ *allowed_or_not*(u, g) $\restriction$ m1 $\times$ (u) $\rightarrow$ read
	outpu	ERROR	OK

allowed_or_not(u,g) = mem $_{-1}$ (u,g) ? unread : unavailable

[6] See table6

8.5 Function table for Read Message: read_message

Table 13: Read Message Function table

read_message (u: UID, m: MID)		**u ∉ users $_{-1}$ ∨ ¬ msg_avb(u,m) ∨ ¬ ms $_{-1}$(u,m) = unread**	**u ∈ users $_{-1}$ ∧ msg_avb(u,m) ∧ ms $_{-1}$(u,m) = unread**
i = 0		initial values	initial values
i >0	ms	NC	ms $_{-1}$ ↾ (u,m) → read
	outpu	ERROR	OK

msg_avb(u,m) = m ∈ msgs $_{-1}$ AND membership (u, info $_{-1}$(m)` recipient)

8.6 Function table for delete Message: delete_message

Table 14: Delete Message Function table

delete_message (u: UID, m: MID)		**u ∉ users $_{-1}$ ∨ ¬ msg_avb (u,m) ∨ ¬ ms $_{-1}$(u,m) = read**	**u ∈ users $_{-1}$ ∧ msg_avb (u,m) ∧ ms $_{-1}$(u,m) = read**
i = 0		initial values	initial values
i >0	*ms*	NC	ms $_{-1}$ ↾ (u,m) → unavailable
	output	ERROR	OK

msg_avb(u,m) = m ∈ msgs $_{-1}$ AND membership (u, info $_{-1}$(m)` recipient)[8]

8.7 Function table for List of New Message: list_new_messages

Table 15: List New Messages Function Table

list_new_messages (u: UID)}		**u ∉ users $_{-1}$∨ groups $_{-1}$ = ∅**	**u ∈ users $_{-1}$∧ groups $_{-1}$ ≠ ∅**
i = 0		-	-
i >0	output	ERROR	list_msg(u,{m | ms $_{-1}$ (u,m) = unread }) [9]

[7] m1 is a id that system shall assign to the new message in a way that m1 is not assigned to any other message.

[8] *info$_1$(m)` recipient* means the recipient of the message m from the MSG_INFO 4

8.8 Function table for List of Old Message: list_old_messages

Table 16: List Old Messages Function Table

list_old_messages (u: UID)}		**u** $\notin$ **users** $_{-1}\vee$ **groups** $_{-1} = \varnothing$	**u** $\in$ **users** $_{-1}\wedge$ **groups** $_{-1} \neq \varnothing$
i = 0		-	-
i >0	output	ERROR	list_msg(u,{m | ms $_{-1}$ (u,m) = read }) [10]

8.9 Function table for List of Groups: list_groups

Table 17: List of groups function table

list_groups		groups $_{-1} = \varnothing$	groups $_{-1} \neq \varnothing$
i = 0		-	-
i >0	output	ERROR [11]	list_of_groups [12]

8.10 Function table for List of Users: list_users

Table 18: List of users function table

list_groups		groups $_{-1} = \varnothing$	groups $_{-1} \neq \varnothing$
i = 0		-	-
i >0	output	ERROR [13]	list_of_groups [14]

[9] Refer to table 7
[10] Refer to table 7
[11] A warning to be more precise.
[12] Sorted alphabetically.

8.11 Function table for error messages and warnings of Queries

Table 19: Error messages and Warnings for Queries

Queries	Error Messages	Warnings
list_groups	-	There are no users registered in the system yet.
list_users	-	There are no groups registered in the system yet.
list_new_messages (uid: UID)	ID must be a positive integer.	There are no new messages for this user.
	User with this ID does not exist.	
list_old_messages (uid: UID)	ID must be a positive integer.	There are no old messages for this user.
	User with this ID does not exist.	

[13] see footnote 11
[14] see footnote 12

8.12 Function table for error messages of commands

Table 20: Error Messages of Commands

Commands	Error messages
add_user (u: UID; un: USER)	ID must be a positive integer.
	ID already in use.
	User name must start with a letter.
add_group (g: GID; gn: GROUP)	ID must be a positive integer.
	ID already in use.
	Group name must start with a letter.
register_user (u: UID; g: GID)	ID must be a positive integer.
	User with this ID does not exist.
	Group with this ID does not exist.
	This registration already exists.
send_message (u: UID; g: GID; txt: TEXT)	ID must be a positive integer.
	User with this ID does not exist.
	Group with this ID does not exist.
	A message may not be an empty string.
	User not authorized to send messages to the specified group.
read_message (u: UID; m: MID)	ID must be a positive integer.
	User with this ID does not exist.
	Message with this ID does not exist.
	User not authorized to access this message.
	Message has already been read. See 'list_old_messages'.
delete_message (u: UID; m: MID)	ID must be a positive integer.
	User with this ID does not exist.
	Message with this ID does not exist.
	Message with this ID not found in old/read messages.
set_message_preview (n: INT)	Message length must be greater than zero.

9 Validation

Completeness and Disjointness of the function tables have been proved by PVS. The proof report is included here and the code is included in the Appendix. Validation of requirements would be done using invariants, theorems and use cases.

Here we had two privacy theorems to prove.

Privacy_weak : Is to prove (with exception of "nothing" case) that list_message and show_message do not leak private information.

Privacy : Same as Privacy_weak except we take into account the "nothing" case.

inv1_send : after send_message command is done, we can prove if the status of message is not unavailable for a user, then that user is in a group that message was sent to. This reflects REQ3.

inv2_send : after send_message command is done, we can prove if the message exists in the set of messages (has been correctly created), then its sender is a member of recipient group. This reflect REQ2.

inv1_read : after read_message command is done, we can prove if the status of message is not unavailable for a user, then that user is in a group that message was sent to. This reflect REQ3.

inv2_read : after read_message command is done, we can prove if the message exists in the set of messages (has been correctly created), then its sender is a member of recipient group. This reflect REQ2.

inv1_hold and **inv2_hold**: after any of the commands, both invariants 1 and 2 stated previously hold. So we can prove the sender of message is part of recipient group after the message exists in set of messages and we can prove if the status of message is not unavailable they user must be part of recipient group of the specific message.

```
***
*** top (1:7:43 11/30/2016)
*** Generated by proveit - ProofLite-6.0.9 (3/14/14)
*** Trusted Oracles
***   MetiTarski: MetiTarski Theorem Prover via PVS proof rule metit
***
 Proof summary for theory top
    Theory totals: 0 formulas, 0 attempted, 0 succeeded (0.00 s)

 Proof summary for theory messenger_prelude
    emptyfun_TCC1 ..... proved - complete   [shostak](0.02 s)
    Theory totals: 1 formulas, 1 attempted, 1 succeeded (0.02 s)

 Proof summary for theory message
    Theory totals: 0 formulas, 0 attempted, 0 succeeded (0.00 s)
```

```
Proof summary for theory Time
   r2d_TCC1 ..... proved - complete   [shostak](0.22 s)
   d2r_TCC1 ..... proved - complete   [shostak](0.02 s)
   held_for_TCC1 ..... proved - complete   [shostak](0.08 s)
   Theory totals: 3 formulas, 3 attempted, 3 succeeded (0.32 s)

Proof summary for theory message_reader
   Theory totals: 0 formulas, 0 attempted, 0 succeeded (0.00 s)

Proof summary for theory insert
   insertLeft_TCC1 ..... proved - complete   [shostak](0.05 s)
   insertRight_TCC1 ..... proved - complete   [shostak](0.03 s)
   insertRightWith_TCC1 ..... proved - complete   [shostak](0.04 s)
   Theory totals: 3 formulas, 3 attempted, 3 succeeded (0.12 s)

Proof summary for theory messenger
   empty_tuples_are_empty ..... proved - complete   [shostak](0.01 s)
   init_state_TCC1 ..... proved - complete   [shostak](0.01 s)
   init_state_TCC2 ..... proved - complete   [shostak](0.01 s)
   init_state_TCC3 ..... proved - complete   [shostak](0.00 s)
   init_state_TCC4 ..... proved - complete   [shostak](0.01 s)
   add_user_TCC1 ..... proved - complete   [shostak](0.01 s)
   add_user_TCC2 ..... proved - complete   [shostak](0.12 s)
   add_user_TCC3 ..... proved - complete   [shostak](0.02 s)
   add_user_TCC4 ..... proved - complete   [shostak](0.05 s)
   add_group_TCC1 ..... proved - complete   [shostak](0.03 s)
   add_group_TCC2 ..... proved - complete   [shostak](0.09 s)
   add_group_TCC3 ..... proved - complete   [shostak](0.04 s)
   register_user_TCC1 ..... proved - complete   [shostak](0.05 s)
   register_user_TCC2 ..... proved - complete   [shostak](0.05 s)
   register_user_TCC3 ..... proved - complete   [shostak](0.10 s)
   register_user_TCC4 ..... proved - complete   [shostak](0.05 s)
   register_user_TCC5 ..... proved - complete   [shostak](0.06 s)
   read_message_TCC1 ..... proved - complete   [shostak](0.04 s)
   read_message_TCC2 ..... proved - complete   [shostak](0.06 s)
   read_message_TCC3 ..... proved - complete   [shostak](0.07 s)
   read_message_TCC4 ..... proved - complete   [shostak](0.08 s)
   delete_message_TCC1 ..... proved - complete   [shostak](0.06 s)
   delete_message_TCC2 ..... proved - complete   [shostak](0.06 s)
   allowed_or_no_TCC1 ..... proved - complete   [shostak](0.02 s)
   send_message_TCC1 ..... proved - complete   [shostak](0.03 s)
   send_message_TCC2 ..... proved - complete   [shostak](0.06 s)
   send_message_TCC3 ..... proved - complete   [shostak](0.06 s)
   list_new_messages_TCC1 ..... proved - complete   [shostak](0.08 s)
   list_new_messages_TCC2 ..... proved - complete   [shostak](0.05 s)
   messenger_ft_TCC1 ..... proved - complete   [shostak](0.05 s)
   messenger_ft_TCC2 ..... proved - complete   [shostak](0.02 s)
   messenger_ft_TCC3 ..... proved - complete   [shostak](0.01 s)
   list_message_privacy_TCC1 ..... proved - complete   [shostak](0.03 s)
   show_message_privacy_TCC1 ..... proved - complete   [shostak](0.03 s)
   show_message_privacy_TCC2 ..... proved - complete   [shostak](0.03 s)
   privacy_weak ..... proved - complete   [shostak](2.69 s)
   privacy ..... proved - complete   [shostak](8.72 s)
   inv1_send_TCC1 ..... proved - complete   [shostak](0.08 s)
   inv1_send ..... proved - complete   [shostak](0.77 s)
   inv2_send ..... proved - complete   [shostak](0.38 s)
   inv1_read_TCC1 ..... proved - complete   [shostak](0.07 s)
   inv1_read ..... proved - complete   [shostak](0.44 s)
```

```
    inv2_read ..... proved - complete   [shostak](0.17 s)
    inv1_holds ..... unfinished         [shostak](5.04 s)
    inv2_holds ..... untried            [Untried]( n/a s)
    Theory totals: 45 formulas, 44 attempted, 43 succeeded (19.92 s)

  Proof summary for theory use_cases
    uc1_state_0 ..... untried           [Untried]( n/a s)
    uc1_state_1 ..... untried           [Untried]( n/a s)
    uc1_state_2 ..... untried           [Untried]( n/a s)
    uc1_state_3 ..... untried           [Untried]( n/a s)
    uc1_state_4 ..... untried           [Untried]( n/a s)
    uc1_state_5 ..... untried           [Untried]( n/a s)
    uc1_state_6 ..... untried           [Untried]( n/a s)
    use_case1_correct ..... untried     [Untried]( n/a s)
    Theory totals: 8 formulas, 0 attempted, 0 succeeded (0.00 s)

  Grand Totals: 60 proofs, 51 attempted, 50 succeeded (20.39 s)
```

10 Use Cases

USE CASE 1.1

This use case describes the how system adds users and groups , registers users to groups. It also describes the system response to the inquiry done by users

- Related System Goals: G1, G2 and G3
- Primary Actor: Customer
- Precondition : The user inputs commands and queries as shown in messenger.definitions.txt
- Post-condition: The system responses as expected as shown in the acceptance tests attached in submission
- Main Success Scenario:
  1. Add user un1 with a positive ID u1.
     - The system adds the user un1.
  2. Add user un2 with a positive ID u1.
     - The system produces an error message “ID already in use.”
  3. Add user un2 with a positive ID u2.
     - The system adds the user un1.
  4. Add group gn1 with a positive ID g1.
     - The system adds the group gn1.
  5. Add group gn2 with a positive ID g2

   - The system adds the group gn2.
6. Register user un1 to group gn1.
   - The system registers user u1 to group g1.
7. Register user un2 to group gn1.
   - The system registers user u2 to group g1.
8. Register user un1 to group gn1.
   - The system produces an error message "This registration already exists."
9. The user inquires the list of groups
   - The system outputs the list of groups in the system , sorted alphabetically.
10. The user inquires the list of users
    - The system outputs the list of users in the system , sorted alphabetically.

- Exception Case Scenario : The system produces error messages to notify user with the type of error that occurred. The user may give correct input and try again.

USE CASE 1.2

This use case describes the how system adds users and groups , registers users to groups. It also describes the messaging process and restrictions.

- Related System Goals: G1, G2, G3 and G4
- Primary Actor: Customer
- Precondition : The user inputs commands and queries as shown in messenger.definitions.txt
- Post-condition: The system responses as expected as shown in the acceptance tests attached in submission
- Main Success Scenario:
  1. Add user un1 with a positive ID u1.
     - The system adds the user un1.
  2. Add user un2 with a positive ID u2.
     - The system adds the user un1.

3. The user inquires the list of groups
   - The system gives a warning message "There are no groups registered in the system yet."
4. Add group gn1 with a positive ID g1.
   - The system adds the group gn1.
5. Add group gn2 with a positive ID g2
   - The system adds the group gn2.
6. Register user un1 to group gn1.
   - The system registers user un1 to group gn1.
7. Register user un2 to group gn1.
   - The system registers user un2 to group gn1.
8. User un1 sends a nonempty message to group gn1.
   - The message becomes available as New/Unread message to all the users registered in group gn1.
   - The message is saved in the system with id 1.
9. User un1 issues command to read message 1.
   - The system responses with "Message has already been read. See 'list_old_messages'."
10. User un1 issues command to delete message 1.
    - The system responses with "Message with this ID not found in old/read messages."
    - The system does not delete message 1.

- Exception Case Scenario : The system produces error messages to notify user with the type of error that occurred. The user may give correct input and try again.

USE CASE 1.3

This use case describes the how system adds users and groups , registers users to groups. It also describes the messaging process and restrictions. It also shows how the system responses to all the inquiries done by users.

- Related System Goals: G1, G2, G3 and G4
- Primary Actor: Customer

- Precondition : The user inputs commands and queries as shown in messenger.definitions.txt
- Post-condition: The system responses as expected as shown in the acceptance tests attached in submission
- Main Success Scenario:
  1. Add user un1 with a positive ID u1.
     - The system adds the user un1.
  2. Add user un2 with a positive ID u2.
     - The system adds the user un1.
  3. Add user un2 with a positive ID u3.
     - The system adds the user un2.
  4. Add group gn1 with a positive ID g1.
     - The system adds the group gn1.
  5. Add group gn2 with a positive ID g2
     - The system adds the group gn2.
  6. Register user un1 to group gn1.
     - The system registers user un1 to group gn1.
  7. Register user un2 with ID u2 to group gn1.
     - The system registers user un2 to group gn1.
  8. Register user un2 with ID u3 to group gn1.
     - The system registers user un2 to group gn1.
  9. Register user un2 with ID u3 to group gn2.
     - The system registers user un2 to group gn2.
  10. User un1 sends a nonempty message to group gn1.
      - The message becomes available as New/Unread message to all the users registered in group gn1.
      - The message is saved in the system with id 1.
  11. User un2 with ID u2 issues command to read message 1.
      - The system reads the message 1 for user un2.
      - And for this user, the message is moved to Old/Read messages.

12. User un1 issues command to delete message 1.
    - The system responses with "Message with this ID not found in old/read messages."
13. User un1 issues command to list old messages.
    - The system responses with a warning "There are no old messages for this user."
14. User un2 with ID u2 issues command to list old messages.
    - The system lists all the Old/Read messages user has.
15. User un2 with ID u2 issues command to list new messages.
    - The system responses with a warning "There are no new messages for this user."
16. User un2 with ID u3 issues command to list new messages.
    - The system lists all the New/Unread messages user has.
17. User un2 with ID u2 issues command to delete message 1.
    - The system deletes the message 1 for user.
    - This message will not appear in Old/Read messages for user.
18. The user sets the message preview length to 25.
    - The system sets the message length to 25.
    - All the messages displayed from now on will be of length 25.

- Exception Case Scenario : The system produces error messages to notify user with the type of error that occurred. The user may give correct input and try again.

USE CASE 1.4

This use case describes the error messages displayed by system.

- Related System Goals: G1, G2, G3 and G4
- Primary Actor: Customer
- Precondition : The user inputs commands and queries as shown in messenger.definitions.txt
- Post-condition: The system responses as expected as shown in the acceptance tests attached in submission
- Main Success Scenario:

1. Add user un1 with ID 0.
   - The system produces an error message “ID must be a positive integer.”
   - The user is not added to the system.
2. Add user “89Bh” with a positive ID u1.
   - The system produces an error message “ User name must start with a letter.”
   - The user is not added to the system.
3. Add group “ Doctors” with a positive ID g1.
   - The system produces an error message “ Group name must start with a letter.”
   - The group is not added to the system.
4. Add group gn1 with a positiveID g1
   - The system adds the group gn2.
5. Add group gn2 with a positive ID g2
   - The system adds the group gn2.
6. Add user un1 with a positive ID u1.
   - The system adds the user un1.
7. Register user un1 to group gn1.
   - The system registers user u1 to group g1.
8. Register user un2 to group gn1.
   - The system produces an error message “User with this ID does not exist.”
9. Register user un1 to group gn1.
   - The system produces an error message “This registration already exists.”
10. The user un1 sends an empty message to group gn1.
    - The system produces an error message “A message may not be an empty string.”
    - The message is not sent to the group.
11. The user un1 sends a nonempty message to group gn2.

  - The system produces an error message "User not authorized to send messages to the specified group."
  - The message is not sent to the group.

12. User un1 issues command to read message 1.
  - The system produces an error message "Message with this ID does not exist."
  - The system does not read any message.

- Exception Case Scenario : The system produces error messages to notify user with the type of error that occurred. The user may give correct input and try again.

11 Acceptance Tests

n this section, the use cases have to be converted into precise acceptance tests (using the function table to describe pre/post conditions) to be run when the design and implementation are complete. The acceptance tests and the expected output for each acceptance test can be found in submission folder.

Table 21: Acceptance tests for use cases

Use cases	Acceptance tests
UC 1.1	at1.txt
	at2.txt
	at4.txt
	at5.txt
UC 1.2	at2.txt
	at4.txt
	at5.txt
UC 1.3	at2.txt
	at3.txt
	at4.txt
	at5.txt
UC 1.4	at1.txt
	at3.txt

12 Traceability

This matrix shows which acceptance tests passed, and which R-descriptions they checked.

Table 22: Traceability Matrix

REQ	at1.txt	at2.txt	at3.txt	at4.txt	at5.txt
REQ1	**X**	**X**		**X**	**X**
REQ2		**X**		**X**	**X**
REQ3		**X**		**X**	**X**
REQ4		**X**		**X**	**X**
REQ5		**X**		**X**	**X**
REQ6	**X**	**X**	**X**		
REQ7			**X**	**X**	**X**
REQ8	**X**	**X**	**X**	**X**	**X**
REQ9	**X**	**X**	**X**	**X**	**X**
REQ10	**X**	**X**	**X**	**X**	**X**
REQ11				**X**	**X**
REQ12	**X**	**X**	**X**	**X**	**X**
REQ13	**X**	**X**	**X**	**X**	**X**
REQ14	**X**	**X**	**X**	**X**	**X**
REQ15		**X**	**X**	**X**	**X**

13 Appendix

```
messenger_prelude : THEORY
BEGIN

precond : TYPE = {precond}

PRE (p : bool) : TYPE = { x : precond | p }

UID, GID, MID, USER, GROUP, TEXT: TYPE+

MSG_STATE : TYPE = {read, unread, unavailable}

  % Definition of an empty function
  emptyfun [T, U : TYPE] (x : {x : T | FALSE}) : RECURSIVE U =
    emptyfun(x)
    MEASURE 0


END messenger_prelude

insert [A,B,Z : TYPE, A2 : TYPE FROM A] : THEORY
BEGIN

   insertLeft (a: A, z:Z, f : [A2, B -> Z])(a0:(add(a,(A2_pred))),b:B) : Z =
     IF a = a0 THEN z
               ELSE f(a0,b)
     ENDIF
   insertRight (b: A, z:Z, f : [B, A2 -> Z])(a:B,b0:(add(b,(A2_pred)))) : Z =
     IF b = b0 THEN z
               ELSE f(a,b0)
     ENDIF

   insertRightWith (b: A, z:[B -> Z], f : [B, A2 -> Z])
        (a:B,b0:(add(b,(A2_pred)))) : Z =
          IF b = b0 THEN z(a)
            ELSE f(a,b0)
          ENDIF

END insert

message [U,G : TYPE] : THEORY
BEGIN
IMPORTING messenger_prelude

  MSG_INFO: TYPE =
    [#
       sender : U
     , recip: G
     , content: TEXT
         % sender is a member of the group the message
    % is sent to
    #]
END message
```

```
message_reader [U,G,M : TYPE] : THEORY
BEGIN
IMPORTING messenger_prelude
IMPORTING message

   readership
     ( mem  : set[[U,G]]
           , info : [ M -> MSG_INFO[U,G] ] )
           ( u : U, m : M ) : bool =
      mem(u,info(m)`recip)
  % a set of pairs (user, message) such that user is allowed
  % to read the message

END message_reader

messenger : THEORY

BEGIN
delta: posreal % sampling time
IMPORTING Time[delta]
IMPORTING message_reader
IMPORTING insert
i: VAR DTIME

STATE: TYPE =

  [#
    users: set[UID]
   ,names: [ (users) -> USER ]
   ,groups: set[GID]
   ,gname: [ (groups) -> GROUP ]
   ,msgs: set[MID]
   ,membership: set[[(users),(groups)]]
   ,info : [(msgs) -> MSG_INFO[(users),(groups)]]
     % specification of individual messages
   ,ms : [ [(users),(msgs)] -> MSG_STATE ]
         % message state: for every message and every user that can read it
   % a message can either be unread, read or deleted
  #]

OUTPUT : DATATYPE
  BEGIN
    OK: OK?
    error: error?
    list_msg (id: UID,ms: set[MID]): list_msg?
    show_msg (id: UID,msg:MID,txt: TEXT): show_msg?
  END OUTPUT

empty_tuples_are_empty : THEOREM
      FORALL (x2: [(emptyset[UID]), (emptyset[MID])]): FALSE

%|- empty_tuples_are_empty : PROOF
%|- (then (skeep) (typepred "x2`1") (expand "emptyset") (propax))
%|- QED

msgOf (st : STATE, u : (st`users))(m : MID): bool =
      st`msgs(m) AND st`membership(u,st`info(m)`recip)
```

```
init_state : STATE =
  (# users := emptyset
   , names := emptyfun
   , groups := emptyset
   , gname := emptyfun
   , msgs := emptyset
   , membership := emptyset
   , info := emptyfun[(emptyset[MID]),
      MSG_INFO[(emptyset[UID]),(emptyset[GID])]]
   , ms := emptyfun[[(emptyset[UID]),(emptyset[MID])],MSG_STATE]
   #)
st : VAR [DTIME -> STATE]
output: VAR [DTIME -> OUTPUT]
u,u2  : VAR UID
un : VAR USER
gn : VAR GROUP
txt : VAR TEXT
g,g2  : VAR GID
m,m1,m2 : VAR MID

add_user (id: UID, name: USER)(st,output)(i : POS_DTIME) : bool =
  COND users_(id) -> output(i) = error AND st(i) = st(i-1)
     , NOT users_(id) ->
             st(i) = st(i-1) WITH
             [ users := add(id, users_)
       , names := names_ WITH [id := name]
       , membership := insertLeft(id,FALSE,mem_)
       , ms   :=  insertLeft(id,unavailable,ms_)
       ]
  AND output(i) = OK
  ENDCOND
   WHERE
       users_   = st(i-1)`users
     , newUsers = add(id,users_)
     , names_   = st(i-1)`names
     , info_    = st(i-1)`info
     , mem_  = st(i-1)`membership
     , ms_   = st(i-1)`ms
     , msgs_ = st(i-1)`msgs

     , groups_ = st(i-1)`groups

add_group (id: GID, name: GROUP)(st,output)(i : POS_DTIME) : bool =
  COND groups_(id) -> output(i) = error AND st(i) = st(i-1)
     , NOT groups_(id) ->
             st(i) = st(i-1) WITH
             [ groups := add(id, groups_)
       , gname := gname_ WITH [id := name]
       , membership := insertRight(id,FALSE,mem_)
       ]
  AND output(i) = OK
  ENDCOND
   WHERE
       users_   = st(i-1)`users
     , gname_   = st(i-1)`gname
     , info_    = st(i-1)`info
     , mem_  = st(i-1)`membership
```

```
      , ms_   = st(i-1)`ms
      , msgs_ = st(i-1)`msgs
      , groups_ = st(i-1)`groups



register_user (u: UID, g: GID)(st,output)(i : POS_DTIME) : bool =
     COND users_(u) AND groups_(g) AND NOT mem_(u,g) ->
               st(i) = st(i-1) WITH
               [ membership := add((u,g),mem_)
             , ms := insertLeft(u,unavailable,ms_)
             ]
    AND output(i) = OK
  ,    NOT users_(u)
    OR NOT groups_(g)
    OR mem_(u,g)
  -> st(i) = st(i-1) AND output(i) = error
    ENDCOND
   WHERE
       users_   = st(i-1)`users
     , names_   = st(i-1)`names
     , info_    = st(i-1)`info
     , mem_   = st(i-1)`membership
     , ms_    = st(i-1)`ms
     , msgs_  = st(i-1)`msgs
     , groups_ = st(i-1)`groups

     % check whether we are allowed to read the message
     % change the state of the message to `read`
     % output the message content


read_message (u, m)(st,output)(i : POS_DTIME) : bool =
     COND users_(u) AND msgOf(st(i-1),u)(m) AND ms_(u,m) = unread ->
               st(i) = st(i-1) WITH
             [ ms := ms_ WITH [ (u,m) := read ] ]
           AND output(i) = OK
       ,      NOT users_(u)
           OR NOT msgOf(st(i-1),u)(m)
      OR NOT ms_(u,m) = unread
       ->       st(i) = st(i-1)
           AND output(i) = error
     ENDCOND
   WHERE
       users_   = st(i-1)`users
     , names_   = st(i-1)`names
     , info_    = st(i-1)`info
     , mem_   = st(i-1)`membership
     , ms_    = st(i-1)`ms
     , msgs_  = st(i-1)`msgs
     , groups_ = st(i-1)`groups

delete_message (u, m)(st,output)(i : POS_DTIME) : bool =
     COND users_(u) AND msgOf(st(i-1),u)(m) ->
               st(i) = st(i-1) WITH
             [ ms := ms_ WITH [ (u,m) := unavailable ] ]
           AND output(i) = OK
       ,      NOT users_(u)
```

```
          OR NOT msgOf(st(i-1),u)(m)
       ->      st(i) = st(i-1)
          AND output(i) = error
    ENDCOND
  WHERE
      users_   = st(i-1)`users
    , names_   = st(i-1)`names
    , info_    = st(i-1)`info
    , mem_  = st(i-1)`membership
    , ms_    = st(i-1)`ms
    , msgs_ = st(i-1)`msgs
    , groups_ = st(i-1)`groups

  % Allocate a fresh message id and give access to the authorized users.
  % This message's state is 'unread' for all user of the group except
  % for the sender. The state must be set to 'read' for the sender.
allowed_or_no (s:STATE, g: (s`groups))(u:(s`users)) : MSG_STATE =
  COND
     s`membership(u,g) -> unread
   , NOT s`membership(u,g) -> unavailable
  ENDCOND

send_message (u:UID,g:GID,txt: TEXT)(st,output)(i : POS_DTIME) : bool =
(EXISTS (m1: MID) : NOT msgs_(m1) AND
    (COND users_(u) AND groups_(g) AND mem_(u,g)  ->
       st(i) = st(i-1) WITH
       [ msgs := add(m1, msgs_)
          ,info := info_ WITH [ m1 := (#sender := u
                                    , recip:= g
                      , content:= txt #)]
       , ms := insertRightWith(m1,allowed_or_no(st(i-1),g),ms_)
                 WITH [ (u,m1) := read ]
         ]
       AND output(i) = OK
     , NOT users_(u)
       OR NOT groups_(g)
    OR NOT mem_(u,g) ->
      st(i) = st(i-1)
          AND output(i) = error
    ENDCOND)
)
  WHERE
      users_   = st(i-1)`users
    , names_   = st(i-1)`names
    , info_    = st(i-1)`info
    , mem_  = st(i-1)`membership
    , ms_    = st(i-1)`ms
    , msgs_ = st(i-1)`msgs
    , groups_ = st(i-1)`groups
list_new_messages (u: UID)(st,output)(i:POS_DTIME): bool =
    st(i) = st(i-1) AND
    COND users_(u) -> output(i) =
        list_msg(u, {m : (msgOf(st(i-1),u)) | ms_(u,m) = unread})
    , NOT users_(u) -> output(i) = error
    ENDCOND
  WHERE
      users_   = st(i-1)`users
    , names_   = st(i-1)`names
```

```
      , info_     = st(i-1)`info
      , mem_  = st(i-1)`membership
      , ms_   = st(i-1)`ms
      , msgs_ = st(i-1)`msgs
      , groups_ = st(i-1)`groups

list_old_messages (u: UID)(st,output)(i:POS_DTIME): bool =
    st(i) = st(i-1) AND
    COND users_(u) -> output(i) =
        list_msg(u, {m : (msgOf(st(i-1),u)) | ms_(u,m) = read})
    , NOT users_(u) -> output(i) = error
    ENDCOND
   WHERE
        users_   = st(i-1)`users
      , names_   = st(i-1)`names
      , info_    = st(i-1)`info
      , mem_  = st(i-1)`membership
      , ms_   = st(i-1)`ms
      , msgs_ = st(i-1)`msgs
      , groups_ = st(i-1)`groups

command : DATATYPE
  BEGIN
    nothing: nothing?
    e_add_user(u:UID,un:USER): add_user?
    e_add_group(g:GID,gn:GROUP): add_group?
    e_register(u:UID,g:GID): register_user?
    e_send(u:UID,g:GID,txt:TEXT): send_message?
    e_read(u:UID,m:MID): read_message?
    e_delete(u:UID,m:MID): delete_message?
    e_list_new(u:UID): list_new_message?
    e_list_old(u:UID): list_old_message?
  END command

cmd: VAR [POS_DTIME -> command]

messenger_ft (cmd,st,output)(i: DTIME): bool =
   COND i = 0 -> st(i) = init_state AND output(i) = OK
      , i > 0 -> CASES cmd(i) OF
                   nothing:        st(i) = st(i-1) AND output(i) = output(i-1)
                 , e_add_user(u,un):  add_user(u,un)        (st,output)(i)
     , e_add_group(g,gn): add_group(g,gn)       (st,output)(i)
     , e_register(u,g):   register_user(u,g)    (st,output)(i)
     , e_send(u,g,txt):   send_message(u,g,txt) (st,output)(i)
     , e_read(u,m):       read_message(u,m)     (st,output)(i)
     , e_delete(u,m):     delete_message(u,m)   (st,output)(i)
     , e_list_new(u):     list_new_messages(u)  (st,output)(i)
     , e_list_old(u):     list_old_messages(u)  (st,output)(i)
     ENDCASES
   ENDCOND

     % When listing messages (either the new messages or the new messages),
     % this property asserts that
     %   0. it is requested by a valid user
     %   1. the user has access to all the returned messages
list_message_privacy(st: STATE,output: OUTPUT): bool =
     list_msg?(output) IMPLIES st`users(id(output))
      AND subset?(ms(output), msgOf(st,id(output)))
```

```
     % When showing a message, this property asserts that
     %   0. it is requested by a valid user
     %   1. the user has access to the requested message
     %   2. the displayed text is actually the body of the message
show_message_privacy(st: STATE, output: OUTPUT): bool =
     show_msg?(output) IMPLIES
               st`users(id(output))
    AND msgOf(st,id(output))(msg(output))
    AND st`info(msg(output))`content = txt(output)

    % If we ignore the case where the user can do "nothing", we can
    % show (without induction) that list_message and show_message
    % do not leak private information.
privacy_weak : THEOREM
   FORALL (i: DTIME): messenger_ft(cmd,st,output)(i) AND (i = 0 OR cmd(i) /= nothing)
         IMPLIES
              list_message_privacy(st(i),output(i))
   AND show_message_privacy(st(i),output(i))

    % Like privacy_weak except that we account for doing "nothing"
    % This requires induction.
privacy : THEOREM
        (FORALL (i: DTIME): messenger_ft(cmd,st,output)(i))
  IMPLIES FORALL (i: DTIME):
              list_message_privacy(st(i),output(i))
   AND show_message_privacy(st(i),output(i))

%|- privacy : PROOF
%|- (then (skeep)
%|-  (spread (induct i)
%|-   ((then (inst -1 0) (grind)) (then (inst -1 0) (grind))
%|-    (then (skeep) (inst -3 "j+1") (grind)))))
%|- QED

inv1 (s : STATE) : bool = FORALL (u:(s`users),m:(s`msgs)):
           s`ms(u,m) /= unavailable IMPLIES
      readership((s`membership),(s`info))(u,m)
inv2 (s : STATE) : bool = FORALL (m:(s`msgs)):
      s`membership( s`info(m)`sender,s`info(m)`recip )

inv1_send : THEOREM
    i > 0
      AND send_message (u,g,txt)(st,output)(i) AND inv1(st(i-1))
   IMPLIES inv1(st(i))

inv2_send : THEOREM
    i > 0
      AND send_message (u,g,txt)(st,output)(i) AND inv2(st(i-1))
   IMPLIES inv2(st(i))

inv1_read : THEOREM
    i > 0
     AND  read_message (u,m)(st,output)(i) AND inv1(st(i-1))
   IMPLIES inv1(st(i))

inv2_read : THEOREM
    i > 0
```

```
     AND  read_message (u,m)(st,output)(i) AND inv2(st(i-1))
   IMPLIES inv2(st(i))

inv1_holds : THEOREM
    (FORALL (i: DTIME): messenger_ft(cmd,st,output)(i))
    IMPLIES (FORALL (i: DTIME): inv1(st(i)))

inv2_holds : THEOREM
    (FORALL (i: DTIME): messenger_ft(cmd,st,output)(i))
    IMPLIES (FORALL (i: DTIME): inv2(st(i)))

END messenger

use_cases : THEORY
BEGIN

IMPORTING messenger

n1,n2 : USER
% Existence TCC generated (at line 352, column 0) for  n1: USER
  % unfinished
%n1_TCC1: OBLIGATION EXISTS (x: USER): TRUE;
gn1,gn2 : GROUP
g1 : GID
u1,u3 : UID
t1,t2: TEXT
cmd : [POS_DTIME -> command]
st : [DTIME -> STATE]
output : [DTIME -> OUTPUT]
i : VAR DTIME

distinct_users : AXIOM
  u1 /= u3

post_st6(s : STATE) : bool =
          NOT s`users(u3)
       OR NOT s`groups(g1)
       OR NOT s`membership(u3,g1)

use_case1 : bool =
        (FORALL i: messenger_ft(cmd,st,output)(i))
    AND cmd(1) = e_add_user(u1,n1)
    AND cmd(2) = e_add_group(g1,gn1)
    AND cmd(3) = e_register(u1,g1)
    AND cmd(4) = e_send(u1,g1,t1)
    AND cmd(5) = e_add_user(u3,n2)
    AND cmd(6) = e_send(u3,g1,t2)

uc1_state_0 : LEMMA
       use_case1 IMPLIES post_st6(st(0))

uc1_state_1 : LEMMA
              use_case1
       IMPLIES post_st6(st(1))
           AND st(1)`users(u1)

uc1_state_2 : LEMMA
              use_case1
```

```
        IMPLIES post_st6(st(2))
            AND st(2)`users(u1)
      AND st(2)`groups(g1)

uc1_state_3 : LEMMA
                use_case1
        IMPLIES post_st6(st(3))
            AND st(3)`users(u1)
      AND st(3)`groups(g1)
      AND st(3)`membership(u1,g1)

uc1_state_4 : LEMMA
                use_case1
        IMPLIES post_st6(st(4))
            AND st(4)`users(u1)
      AND st(4)`groups(g1)
      AND st(4)`membership(u1,g1)
      AND NOT empty?(msgOf(st(4),u1))

uc1_state_5 : LEMMA
                use_case1
        IMPLIES post_st6(st(5))

uc1_state_6 : LEMMA
                use_case1
        IMPLIES output(6) = error

% state 0: post_st6
% state 1: post_st6
%      AND users(u1)
% state 2: post_st6
%      AND users(u1) AND groups(g1)
% state 3: post_st6
%      AND membership(u1,g1)
%      AND users(u1) AND groups(g1)
% state 4:
%          post_st6
%      AND output(4) = OK
%      AND membership(u1,g1)
%      AND NOT (msgOf(u1))
% state 5: post_st6
% state 6: output = error

use_case1_correct : THEOREM
        use_case1
    IMPLIES
        output(4) = OK
    AND st(4)`users(u1)
    AND st(4)`groups(g1)
    AND st(4)`membership(u1,g1)
    AND NOT empty?(msgOf(st(4),u1))
    AND output(6) = error

END use_cases
```