Formal Modeling of Stack Overflow in VDM++

Master in Informatics and Computing Engineering

Formal Methods in Software Engineering

January, 2018

José Miguel Matos Lopes da Costa - up201402717

Tiago Rafael Ferreira da Silva - up201402841

Contents

1. Informal system description and list of requirements

1.1 Informal system description

This project attempts to model all the information managed by the *Stack Overflow* website. This website consists of a platform where users can post questions and answers on a large number of computer programming topics. Users can also vote (upvote and downvote) on both questions and answers. The OP (*Original Poster* - the user who posted the question) can also select one of the posted answers as the *accepted answer*, thus stating that specific answer solved the problem, or was the most helpful to finding a solution. A question is considered *answered* only when one answer has been accepted by the OP.

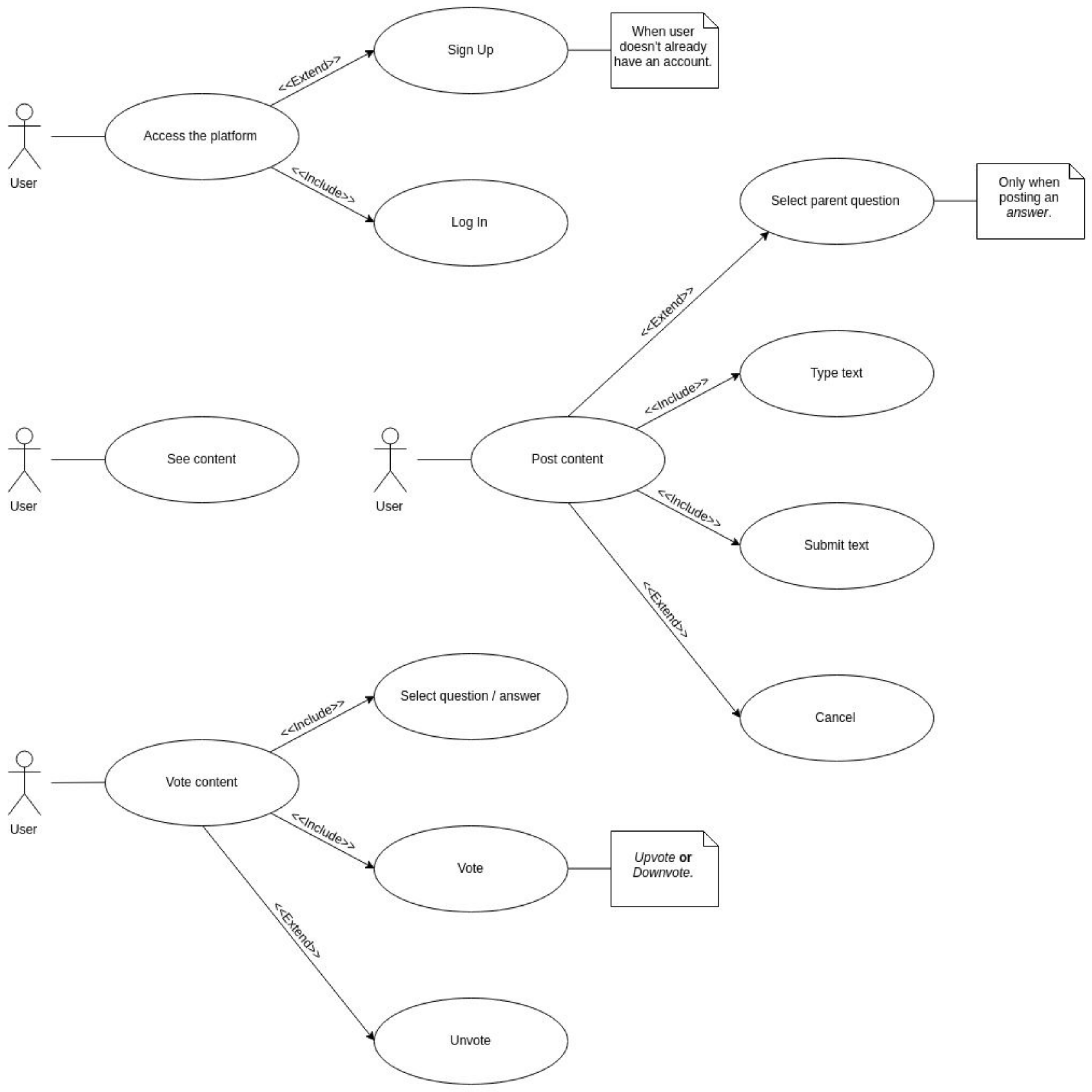
1.2 List of requirements

Id	Priority	Description
R1	Mandatory	The user should be able to access the platform by signing up and logging in.
R2	Mandatory	The user should be able to see questions and answers.
R3	Mandatory	The user should be able to post questions and answers.
R4	Mandatory	The user should be able to vote (upvote and downvote) questions and answers.

These requirements are directly translated onto use cases as shown next.

2. Visual UML model

2.1 Use case model



The major use case scenarios (to be used later as test scenarios) are described next.

Scenario	**Access the platform**
Description	Normal scenario for accessing the platform, using an username and a password.
Pre-conditions	1. The username is unique. *(input)*
Post-conditions	1. The account was created as intended. *(final system state)* 2. The new user was added to the list of existing users. *(final system state)* 3. The user logged in as intended. *(final system state)*
Steps	1. Create an account, using a unique username and password.

	2. Log in using the same username and password as in step 1.
Exceptions	1. The username is already in use by another user (step 1). 2. The username and password don't match (step 2).

Scenario	**See content**
Description	Normal scenario for browsing the website, viewing questions and answers.
Pre-conditions	1. There exists at least one question. *(initial system state)*
Post-conditions	1. The questions and answers were shown. *(output)* 2. No changes were made to the questions nor to the answers. *(final system state)* 3. If the user was logged in, remains logged in. *(final system state)*
Steps	1. The platform shows the existing questions and answers.
Exceptions	none

Scenario	**Post content**
Description	Normal scenario for posting content.
Pre-conditions	1. The user is logged in. *(initial system state)* 2. There exists at least one question, if posting an answer. *(initial system state)*
Post-conditions	1. The posted content was added to the platform. *(final system state)* 2. The posted content is shown. *(output)* 3. No changes were made to existing questions nor answers. *(final system state)* 4. The user remains logged in. *(final system state)*
Steps	1. Select parent question, if posting an answer. 2. Type text. 3. Submit text typed in step 2. 4. The platform shows submitted content.
Exceptions	1. No parent question selected, if posting an answer (step 1). 2. The user cancels the post (alternative to step 3) - see scenario description next. 3. Text to post is the empty string (step 3).

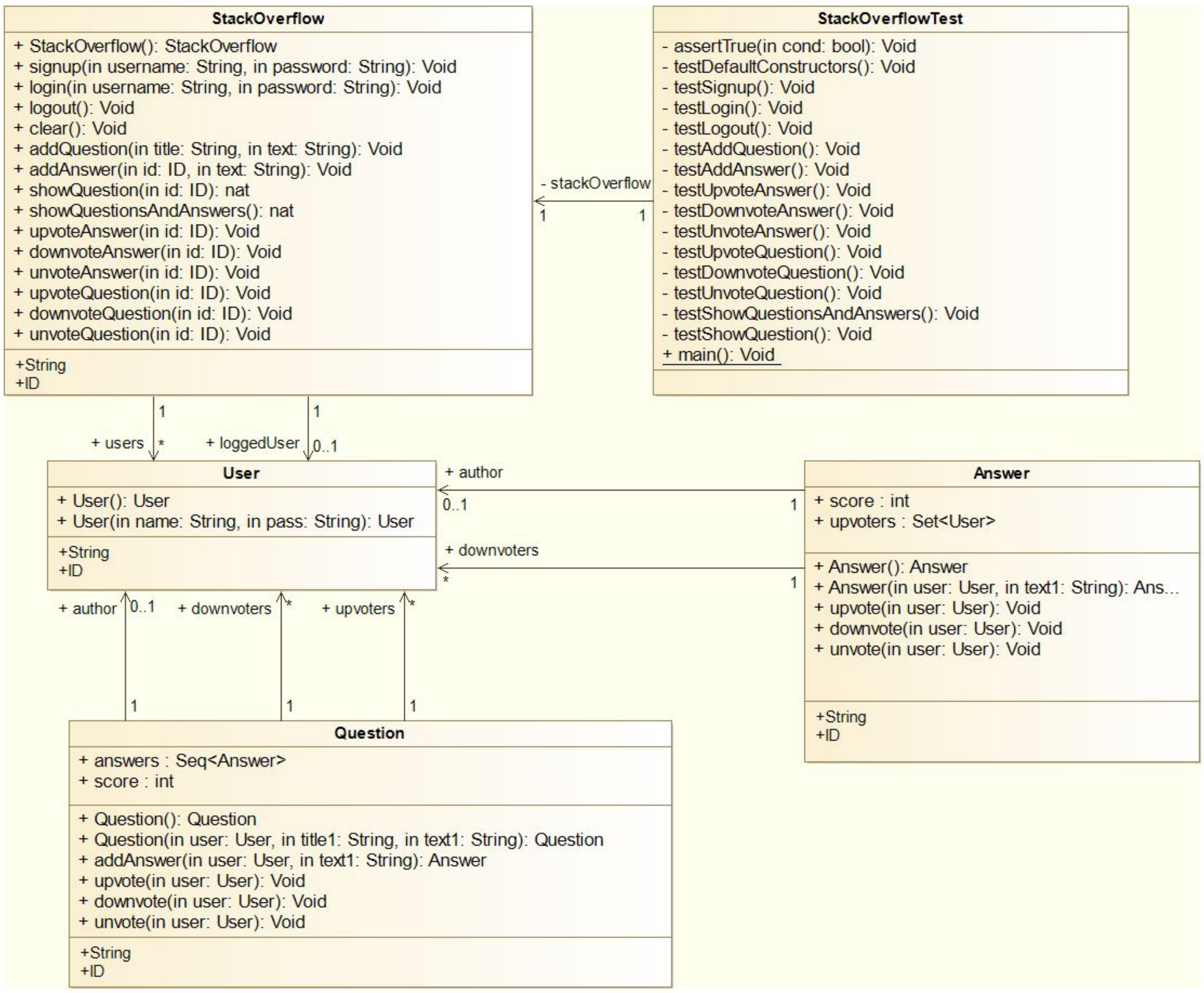
Scenario	**Cancel**
Description	Alternative scenario for posting content, in which the user cancels.
Pre-conditions	1. The user is logged in. *(initial system state)* 2. There exists at least one question, if posting an answer. *(initial system state)*
Post-conditions	1. No content was added to the platform. *(final system state)* 2. No changes were made to existing questions nor answers. *(final system state)* 3. The user remains logged in. *(final system state)*
Steps	1. Select parent question, if posting an answer. 2. Type text. 3. Cancel.
Exceptions	1. No parent question selected, if posting an answer (step 1).

Scenario	**Vote content**
Description	Normal scenario for voting on content.
Pre-conditions	1. The user is logged in. *(initial system state)*
Post-conditions	1. The vote was added to the question / answer vote count. *(final system state)* 2. The voted content is shown. *(output)*

	3. No changes were made to other existing questions nor answers. *(final system state)* 4. The user remains logged in. *(final system state)*
Steps	1. Select content (question or answer) to vote on. 2. Vote (Upvote or Downvote). 3. The platform shows voted content.
Exceptions	1. No question / answer selected (step 1). 2. The user undoes previous vote (alternative to step 2) - see scenario description next.

Scenario	**Unvote**
Description	Alternative scenario for voting on content, in which the user undoes previous vote.
Pre-conditions	1. The user is logged in. *(initial system state)* 2. The user has voted on selected content previously. *(initial system state)*
Post-conditions	1. The previous vote was removed from the question / answer vote count. *(final system state)* 2. The (un)voted content is shown. *(output)* 3. No changes were made to other existing questions nor answers. *(final system state)* 4. The user remains logged in. *(final system state)*
Steps	1. Select content (question or answer) to unvote. 2. Unvote. 3. The platform shows (un)voted content.
Exceptions	1. No question / answer selected (step 1).

2.2 Class model


StackOverflow
+ StackOverflow(): StackOverflow
+ signup(in username: String, in password: String): Void
+ login(in username: String, in password: String): Void
+ logout(): Void
+ clear(): Void
+ addQuestion(in title: String, in text: String): Void
+ addAnswer(in id: ID, in text: String): Void
+ showQuestion(in id: ID): nat
+ showQuestionsAndAnswers(): nat
+ upvoteAnswer(in id: ID): Void
+ downvoteAnswer(in id: ID): Void
+ unvoteAnswer(in id: ID): Void
+ upvoteQuestion(in id: ID): Void
+ downvoteQuestion(in id: ID): Void
+ unvoteQuestion(in id: ID): Void
+String
+ID
StackOverflowTest
- assertTrue(in cond: bool): Void
- testDefaultConstructors(): Void
- testSignup(): Void
- testLogin(): Void
- testLogout(): Void
- testAddQuestion(): Void
- testAddAnswer(): Void
- testUpvoteAnswer(): Void
- testDownvoteAnswer(): Void
- testUnvoteAnswer(): Void
- testUpvoteQuestion(): Void
- testDownvoteQuestion(): Void
- testUnvoteQuestion(): Void
- testShowQuestionsAndAnswers(): Void
- testShowQuestion(): Void
+ main(): Void
- stackOverflow
1
1
+ users
1
*
+ loggedUser
1
0..1
User
+ User(): User
+ User(in name: String, in pass: String): User
+String
+ID
+ author
0..1
1
+ downvoters
*
1
Answer
+ score : int
+ upvoters : Set<User>
+ Answer(): Answer
+ Answer(in user: User, in text1: String): Ans...
+ upvote(in user: User): Void
+ downvote(in user: User): Void
+ unvote(in user: User): Void
+String
+ID
+ author
0..1
+ downvoters
*
+ upvoters
*
1
1
1
Question
+ answers : Seq<Answer>
+ score : int
+ Question(): Question
+ Question(in user: User, in title1: String, in text1: String): Question
+ addAnswer(in user: User, in text1: String): Answer
+ upvote(in user: User): Void
+ downvote(in user: User): Void
+ unvote(in user: User): Void
+String
+ID

Class	Description
`Question`	Defines a question that can be added by a user, responded and voted on.
`Answer`	Defines an answer that can be added to a question, by a user, and voted on.
`User`	Defines a user that can use the model.
`StackOverflow`	Core model; defines the state variables and operations available to the users.
`StackOverflowTest`	Defines the test/usage scenarios and test cases for the model.

3. Formal VDM++ model

3.1 Class Answer

```
/**
* This class represents an answer
*/
class Answer
       types
              /** String */
              public String = seq of char;

              /** Identifier (starts at 1) */
              public ID = nat1;


       instance variables

              /** Id of the next answer */
              public static nextid: ID := 1;

              /** Id of this answer */
              public id: ID;

              /** Body of this answer */
              public text: String;

              /** Score of this answer */
              public score: int := 0;

              /** Set of users that have upvoted this answer */
              public upvoters: set of User := {};

              /** Set of users that have downvoted this answer */
              public downvoters: set of User := {};

              /** Author of this answer */
              public author: [User];

              -- author cant be null
              inv author <> nil;

              -- score is the sum of upvotes and downvotes
              inv score = card upvoters - card downvoters;


       operations

              /**
              * Default Constructor
              *
              * @post Answer was created
              */
              public Answer: () ==> Answer
              Answer() == (

                     author := new User();
                     text := " ";
                     id := nextid;
                     nextid := nextid + 1;

                     return self;
              )
```

```
post (
        text = " " and nextid = id + 1
);


/**
* Constructor
*
* @param user Author
* @param text1 Body
*
* @post Answer was created
*/
public Answer: User * String ==> Answer
Answer(user, text1) == (

        author := user;
        text := text1;
        id := nextid;
        nextid := nextid + 1;

        return self;
)
post (
        text = text1 and nextid = id + 1
);


/**
* Adds an upvote to this answer
*
* @param user User that upvoted this
*
* @pre User hasn't upvoted this
*
* @post User upvoted this
* @post User hasn't downvoted this
* @post Score updated
*/
public upvote: User ==> ()
upvote(user) == (

        atomic (

                -- add user to set of upvoters
                upvoters := upvoters union {user};

                -- update score
                score := score + 1;
        );

        if (user in set downvoters) then (

                atomic (

                        -- remove user from set of downvoters
                        downvoters := downvoters \ {user};

                        -- update score
                        score := score + 1;
                );
        );
)
pre  (
        user not in set upvoters
)
```

```
post (
        upvoters = upvoters~ union {user} and
        downvoters = downvoters~ \ {user} and
        score = card upvoters - card downvoters
);


/**
* Adds a downvote to this answer
*
* @param user User that downvoted this
*
* @pre User hasn't downvoted this
*
* @post User downvoted this
* @post User hasn't upvoted this
* @post Score updated
*/
public downvote: User ==> ()
downvote(user) == (

        atomic (

                -- add user to set of downvoters
                downvoters := downvoters union {user};

                -- update score
                score := score - 1;
        );

        if (user in set upvoters) then (

                atomic (

                        -- remove user from set of upvoters
                        upvoters := upvoters \ {user};

                        -- update score
                        score := score - 1;
                )
        );
)
pre  (
        user not in set downvoters
)
post (
        downvoters = downvoters~ union {user} and
        upvoters = upvoters~ \ {user} and
        score = card upvoters - card downvoters
);


/**
* Removes previously casted vote from this answer
*
* @param user User that unvoted this
*
* @pre User has voted on this
*
* @post User hasn't upvoted this
* @post User hasn't downvoted this
* @post Score updated
*/
public unvote: User ==> ()
unvote(user) == (
                        if (user in set upvoters) then (

                                atomic (

                                        -- remove user from set of upvoters
                                        upvoters := upvoters \ {user};

                                        -- update score
                                        score := score - 1;
                                )
                        );

                        if (user in set downvoters) then (

                                atomic (

                                        -- remove user from set of downvoters
                                        downvoters := downvoters \ {user};

                                        -- update score
                                        score := score + 1;
                                )
                        );
                )
                pre  (
                        user in set upvoters or user in set downvoters
                )
                post (
                        upvoters = upvoters~ \ {user} and
                        downvoters = downvoters~ \ {user} and
                        score = card upvoters - card downvoters
                );

end Answer
```

3.2 Class Question

```
/**
* This class represents a question
*/
class Question

   types

      /** String */
      public String = seq of char;

      /** Identifier (starts at 1) */
      public ID = nat1;


   instance variables

      /** Id of the next question */
      public static nextid: ID := 1;

      /** Id of this question */
      public id: ID;

      /** Title of this question */
      public title: String;

      /** Body of this question */
      public text: String;
      /** List of answers */
      public answers: seq of Answer := [];

      /** Score of this question */
      public score: int := 0;

      /** Set of users that have upvoted this question */
      public upvoters: set of User := {};

      /** Set of users that have downvoted this question */
      public downvoters: set of User := {};

      /** Author of this question */
      public author: [User];

      -- author cant be null
      inv author <> nil;

      -- score is the sum of upvotes and downvotes
      inv score = card upvoters - card downvoters;


   operations

      /**
      * Default Constructor
      *
      * @post Question was created
      */
      public Question: () ==> Question
      Question() == (

         author := new User();
         title := " ";
         text := " ";
         id := nextid;
         nextid := nextid + 1;

         return self;
      )
      post (
         title = " " and text = " " and nextid = id + 1
      );


      /**
      * Constructor
      *
      * @param user Author
      * @param title1 Title
      * @param text1 Body
      *
      * @post Question was created
      */
      public Question: User * String * String ==> Question
      Question(user, title1, text1) == (

         author := user;
         title := title1;
         text := text1;
         id := nextid;
         nextid := nextid + 1;

         return self
      )
```

```
post (
   title = title1 and text = text1 and nextid = id + 1
);


/**
* Adds an answer to this question
*
* @param user Author
* @param text1 Body
*
* @post Answer was added
*/
public addAnswer: User * String ==> Answer
addAnswer(user, text1) == (

   -- create answer
   dcl answer: Answer := new Answer(user, text1);

   -- add answer
   answers := answers^[answer];

   return answer;
)
post (
   len answers = len answers~ + 1 and
   exists answer in set elems answers & (
      answer.author = user and
      answer.text = text1
   )
);


/**
* Adds an upvote to this question
*
* @param user User that upvoted this
*
* @pre User hasn't upvoted this
*
* @post User upvoted this
* @post User hasn't downvoted this
* @post Score updated
*/
public upvote: User ==> ()
upvote(user) == (

   atomic (

      -- add user to set of voters
      upvoters := upvoters union {user};

      -- update score
      score := score + 1;
   );

   if (user in set downvoters) then (

      atomic (

         -- remove user from set of downvoters
         downvoters := downvoters \ {user};

         -- update score
         score := score + 1;
      );
   );
)
pre  (
   user not in set upvoters
)
post (
   upvoters = upvoters~ union {user} and
   downvoters = downvoters~ \ {user} and
   score = card upvoters - card downvoters
);


/**
* Adds a downvote to this question
*
* @param user User that downvoted this
*
* @pre User hasn't downvoted this
*
* @post User downvoted this
* @post User hasn't upvoted this
* @post Score updated
*/
public downvote: User ==> ()
downvote(user) == (

   atomic (

      -- add user to set of downvoters
      downvoters := downvoters union {user};

      -- update score
      score := score - 1;
   );

   if (user in set upvoters) then (

      atomic (

         -- remove user from set of upvoters
         upvoters := upvoters \ {user};

         -- update score
         score := score - 1;
      )
   );
)
pre  (
   user not in set downvoters
)
post (
   downvoters = downvoters~ union {user} and
   upvoters = upvoters~ \ {user} and
   score = card upvoters - card downvoters
);


/**
* Removes previously casted vote from this question
*
* @param user User that unvoted this
*
* @pre User has voted on this
*
* @post User hasn't upvoted this
* @post User hasn't downvoted this
```

```
      * @post Score updated
      */
      public unvote: User ==> ()
      unvote(user) == (

         if (user in set upvoters) then (

            atomic (

               -- remove user from set of upvoters
               upvoters := upvoters \ {user};

               -- update score
               score := score - 1;
            )
         );

         if (user in set downvoters) then (

            atomic (

               -- remove user from set of downvoters
               downvoters := downvoters \ {user};

               -- update score
               score := score + 1;
            )
         );
      )
      pre  (
         user in set upvoters or user in set downvoters
      )
      post (
         upvoters = upvoters~ \ {user} and
         downvoters = downvoters~ \ {user} and
         score = card upvoters - card downvoters
      );

end Question
```

3.3 Class User

```
/**
* This class represents an user
*/
class User

   types

      /** String */
      public String = seq of char;

      /** Identifier (starts at 1) */
      public ID = nat1;


   instance variables

      /** Id of the next user */
      public static nextid: ID := 1;

      /** Id of this user */
      public id: ID;
      /** Username */
      public username: String;

      /** Password */
      public password: String;


   operations

      /**
      * Default Constructor
      *
      * @post User was created
      */
      public User: () ==> User
      User() == (

         username := "";
         password := "";
         id := nextid;
         nextid := nextid + 1;

         return self;
      )
      post (
         username = "" and password = "" and nextid = id + 1
      );


      /**
      * Constructor
      *
      * @param name Username
      * @param pass Password
      *
      * @post User was created
      */
      public User: String * String ==> User
      User(name, pass) == (

         username := name;
         password := pass;
         id := nextid;
         nextid := nextid + 1;
         return self;
      )
      post (
         username = name and password = pass and nextid = id + 1
      );

end User
```

3.4 Class StackOverflow

```
/**
* This class represents the Stack Overflow website
*/
class StackOverflow

   types

      /** String */
      public String = seq of char;
      /** Identifier (starts at 1) */
      public ID = nat1;


   instance variables

      /** Set of existing users */
      public users: set of User := {};

      /** List of existing questions */
      public questions: seq of Question := [];

      /** The user that's currently logged in */
      public loggedUser: [User] := nil;


   operations

      /**
      * Default Constructor
      *
      * @post StackOverflow was created
      */
      public StackOverflow: () ==> StackOverflow
      StackOverflow() == (

         return self;
      )
      post (
         users = {} and
         questions = [] and
         loggedUser = nil
      );


      /**
      * Creates a new user
      *
      * @param username Username
      * @param password Password
      *
      * @pre Username is unique
      *
      * @post User was created
      */
      public signup: String * String ==> ()
      signup(username, password) == (

         -- create user
         dcl user: User := new User(username, password);

         -- add user
         users := users union {user};
      )
      pre  (
         forall user in set users & user.username <> username
      )
      post (
         card users = card users~ + 1 and
         exists user in set users & (
            user.username = username and
            user.password = password and
            user.id = User`nextid - 1
         )
      );
```

```
	/**
	* Logs in the user with the given username-password
	*
	* @param username Username
	* @param password Password
	*
	* @pre There isn't an user already logged in
	* @pre Username and password are a match
	*
	* @post User logged in
	*/
	public login: String * String ==> ()
	login(username, password) == (
	
	   loggedUser := iota user in set users & (user.username = username and user.password = 
password);
	)
	pre  (
	   loggedUser = nil and
	   exists1 user in set users & (
	      user.username = username and
	      user.password = password
	   )
	)
	post (
	   loggedUser.username = username and loggedUser.password = password
	);
	
	
	/**
	* Logs out the currently logged in user
	*
	* @pre User must be logged in
	*
	* @post User logged out
	*/
	public logout: () ==> ()
	logout() == (
	
	   loggedUser := nil;
	)
	pre  (
	   loggedUser <> nil
	)
	post (
	   loggedUser = nil
	);
	
	
	/**
	* Resets this instance
	*
	* @post StackOverflow was cleared
	*/
	public clear: () ==> ()
	clear() == (
	
	   users := {};
	   questions := [];
	   loggedUser := nil;
	
	   -- reset static IDs
	   User`nextid := 1;
	   Answer`nextid := 1;
```

```
   Question`nextid := 1;

)
post (
   users = {} and questions = [] and loggedUser = nil
);


/**
* Adds a new question
*
* @param user Author
* @param title Title
* @param text Body
*
* @pre User is logged in
*
* @post Question was added
* @post User stays logged in
*/
public addQuestion: String * String ==> ()
addQuestion(title, text) == (

   -- create question
   dcl question: Question := new Question(loggedUser, title, text);

   -- add question
   questions := questions^[question];
)
pre  (
   loggedUser <> nil
)
post (
   len questions = len questions~ + 1 and
   exists question in set elems questions & (
      question.author = loggedUser and
      question.title = title and
      question.text = text
   ) and
   loggedUser = loggedUser~
);


/**
* Adds an answer to given question
*
* @param id Parent question
* @param text1 Body
*
* @pre User is logged in
* @pre Question to reply to exists
*
* @post No changes were made to other questions
* @post User stays logged in
*/
public addAnswer: ID * String ==> ()
addAnswer(id, text) == (

   -- get question
   dcl question: Question := iota q in set elems questions & q.id = id;

   -- add answer
   dcl answer: Answer;
   answer := question.addAnswer(loggedUser, text);
)
pre  (
         loggedUser <> nil and
         exists q in set elems questions & (q.id = id)
      )
      post (
         elems questions \ {q | q in set elems questions & q.id = id} = elems questions~ \ {q | q
in set elems questions~ & q.id = id} and
         loggedUser = loggedUser~
      );


      /**
      * Shows given question and its answers
      *
      * @param id Question to show
      *
      * @return Number of questions-answers shown
      *
      * @pre At least one question exists
      *
      * @post No changes were made
      * @post User stays logged in
      */
      public showQuestion: ID ==> nat
      showQuestion(id) == (

         dcl answersTemp : seq of Answer;
         dcl answerTemp: Answer;
         dcl j : nat := 0;
         dcl num : nat1 := 1;
         dcl sum : nat := 0;

         dcl questionTemp: Question := iota q in set elems questions & q.id = id;

         IO`print("\n");
         IO`print(questionTemp.title);
         IO`print("(");
         IO`print(questionTemp.score);
         IO`print("): \n ");
         IO`print(questionTemp.text);
         IO`print("\n");
         sum := sum + 1;

         answersTemp := questionTemp.answers;
         num := 1;
         while j  < len answersTemp do (

            j := j + 1;
            answerTemp := answersTemp(j);

            IO`print(num);
            IO`print("(");
            IO`print(answerTemp.score);
            IO`print("): ");
            IO`print(answerTemp.text);
            IO`print("\n");
            sum := sum + 1;

            num := num + 1;
         );

         return sum;
      )
      pre  (
         len questions > 0
      )
      post (
         questions = questions~ and
         loggedUser = loggedUser~
      );


      /**
      * Shows all questions and their respective answers
      *
      * @return Number of questions-answers shown
      *
      * @pre At least one question exists
      *
      * @post No changes were made
      * @post User stays logged in
      */
      public showQuestionsAndAnswers: () ==> nat
      showQuestionsAndAnswers() == (

         dcl questionsTemp : seq of Question := questions;
         dcl questionTemp : Question;
         dcl i : nat := 0;
         dcl sum : nat := 0;

         while i < len questionsTemp do (
            i := i + 1;
            questionTemp := questionsTemp(i);

            sum := sum + showQuestion(questionTemp.id);
         );

         return sum;
      )
      pre  (
         len questions > 0
      )
      post (
         questions = questions~ and
         loggedUser = loggedUser~
      );


      /**
      * Upvotes the given answer
      *
      * @param id Answer to upvote
      *
      * @pre User is logged in
      * @pre Answer to upvote exists
      *
      * @post No changes were made to other answers
      * @post User stays logged in
      */
      public upvoteAnswer: ID ==> ()
      upvoteAnswer(id) == (

         -- get answer's parent
         dcl question: Question := iota q in set elems questions & (id in set {a.id | a in set
elems q.answers});

         -- get answer
         dcl answer: Answer := iota a in set elems question.answers & a.id = id;

         -- upvote
         answer.upvote(loggedUser);
      )
      pre  (
         loggedUser <> nil and
         exists q in set elems questions & (
            id in set {a.id | a in set elems q.answers}
         )
      )
      post (
         elems questions \ {q | q in set elems questions & id in set {a.id | a in set elems
q.answers}} = elems questions~ \ {q | q in set elems questions~ & id in set {a.id | a in set elems
q.answers}} and
         loggedUser = loggedUser~
      );


      /**
      * Downvotes the given answer
      *
      * @param id Answer to downvote
      *
      * @pre User is logged in
      * @pre Answer to downvote exists
      *
      * @post No changes were made to other answers
      * @post User stays logged in
      */
      public downvoteAnswer: ID ==> ()
      downvoteAnswer(id) == (

         -- get answer's parent
         dcl question: Question := iota q in set elems questions & (id in set {a.id | a in set
elems q.answers});

         -- get answer
         dcl answer: Answer := iota a in set elems question.answers & a.id = id;

         -- downvote
         answer.downvote(loggedUser);
      )
      pre  (
         loggedUser <> nil and
         exists q in set elems questions & (
            id in set {a.id | a in set elems q.answers}
         )
      )
      post (
         elems questions \ {q | q in set elems questions & id in set {a.id | a in set elems
q.answers}} = elems questions~ \ {q | q in set elems questions~ & id in set {a.id | a in set elems
q.answers}} and
         loggedUser = loggedUser~
      );


      /**
      * Unvotes the given answer
      *
      * @param id Answer to unvote
      *
      * @pre User is logged in
      * @pre Answer to unvote exists
      *
      * @post No changes were made to other answers
      * @post User stays logged in
      */
      public unvoteAnswer: ID ==> ()
      unvoteAnswer(id) == (

         -- get answer's parent
         dcl question: Question := iota q in set elems questions & (id in set {a.id | a in set
elems q.answers});

         -- get answer
         dcl answer: Answer := iota a in set elems question.answers & a.id = id;

         -- unvote
         answer.unvote(loggedUser);
      )
      pre  (
         loggedUser <> nil and
         exists q in set elems questions & (
            id in set {a.id | a in set elems q.answers}
         )
      )
      post (
         elems questions \ {q | q in set elems questions & id in set {a.id | a in set elems
q.answers}} = elems questions~ \ {q | q in set elems questions~ & id in set {a.id | a in set elems
q.answers}} and
         loggedUser = loggedUser~
      );


      /**
      * Upvotes the given question
      *
      * @param id Question to upvote
      *
      * @pre User is logged in
      * @pre Question to upvote exists
      *
      * @post No changes were made to other questions
      * @post User stays logged in
      */
      public upvoteQuestion: ID ==> ()
      upvoteQuestion(id) == (

         -- get question
         dcl question: Question := iota q in set elems questions & q.id = id;

         -- upvote
         question.upvote(loggedUser);
      )
      pre  (
         loggedUser <> nil and
         exists q in set elems questions & (q.id = id)
      )
      post (
         elems questions \ {q | q in set elems questions & q.id = id} = elems questions~ \ {q | q
in set elems questions~ & q.id = id} and
         loggedUser = loggedUser~
      );


      /**
      * Downvotes the given question
      *
      * @param id Question to downvote
      *
      * @pre User is logged in
      * @pre Question to downvote exists
      *
      * @post No changes were made to other answers
      * @post User stays logged in
      */
      public downvoteQuestion: ID ==> ()
```

```
	downvoteQuestion(id) == (
	
	   -- get question
	   dcl question: Question := iota q in set elems questions & q.id = id;
	
	   -- downvote
	   question.downvote(loggedUser);
	)
	pre  (
	   loggedUser <> nil and
	   exists q in set elems questions & (q.id = id)
	)
	post (
	   elems questions \ {q | q in set elems questions & q.id = id} = elems questions~ \ {q | q
in set elems questions~ & q.id = id} and
	   loggedUser = loggedUser~
	);


	/**
	* Unvotes the given question
	*
	* @param id Question to unvote
	*
	* @pre User is logged in
	* @pre Question to unvote exists
	*
	* @post No changes were made to other questions
	* @post User stays logged in
	*/
	public unvoteQuestion: ID ==> ()
	unvoteQuestion(id) == (
	
	   -- get question
	   dcl question: Question := iota q in set elems questions & q.id = id;
	
	   -- unvote
	   question.unvote(loggedUser);
	)
	pre  (
	   loggedUser <> nil and
	   exists q in set elems questions & (q.id = id)
	)
	post (
	   elems questions \ {q | q in set elems questions & q.id = id} = elems questions~ \ {q | q
in set elems questions~ & q.id = id} and
	   loggedUser = loggedUser~
	);

end StackOverflow
```

4. Model validation

4.1 Class TestStackOverlfow

```
/**
* This class is used for testing purposes.
*
* Commented lines are examples that break pre-conditions or post-conditions,
* and therefore cannot be executed.
*/
class StackOverflowTest

       instance variables

              /** Test subject */
              stackOverflow: StackOverflow := new StackOverflow();


       operations

              /**
              * Check if something is true
              *
              * @pre Cond is true
              */
              private assertTrue: bool ==> ()
              assertTrue(cond) == (
                     return;
              )
              pre cond;


              /**
              * Test Default Constructors
              */
              private testDefaultConstructors: () ==> ()
              testDefaultConstructors() == (

                     -- test
                     dcl answer: [Answer] := new Answer();
                     dcl question: [Question] := new Question();
                     dcl user: [User] := new User();

                     assertTrue(
                            answer <> nil and
                            question <> nil and
                            user <> nil
                     );
              );


              /**
              * Test signup
              */
              private testSignup: () ==> ()
              testSignup() == (

                     -- requirements
                     stackOverflow.clear();

                     -- test
                     stackOverflow.signup("user1", "pass1");
```

```
        stackOverflow.signup("user2", "pass2");
 -- stackOverflow.signup("user1", "pass3");

        assertTrue(exists1 user in set stackOverflow.users & (
                user.username = "user1" and
                user.password = "pass1"
        ));
);


/**
* Test login
*/
private testLogin: () ==> ()
testLogin() == (

        -- requirements
        stackOverflow.clear();
        stackOverflow.signup("user1", "pass1");

        -- test
        stackOverflow.login("user1", "pass1");
 -- stackOverflow.login("user1", "pass2");

        assertTrue(
                stackOverflow.loggedUser.username = "user1" and
                stackOverflow.loggedUser.password = "pass1"
        );
);


/**
* Test logout
*/
private testLogout: () ==> ()
testLogout() == (

        -- requirements
        stackOverflow.clear();
        stackOverflow.signup("user1", "pass1");
        stackOverflow.login("user1", "pass1");

        -- test
    stackOverflow.logout();
 -- stackOverflow.logout();

        assertTrue(
                stackOverflow.loggedUser = nil
        );
);


/**
* Test addQuestion
*/
private testAddQuestion: () ==> ()
testAddQuestion() == (

        -- requirements
        stackOverflow.clear();
        stackOverflow.signup("user1", "pass1");
        stackOverflow.login("user1", "pass1");

        -- test
        stackOverflow.addQuestion("title1","text1");
        assertTrue(
                exists1 q in set elems stackOverflow.questions & (
                        q.id = 1 and
                        q.title = "title1" and
                        q.text = "text1"
                )
        );

);


/**
* Test add Answer
*/
private testAddAnswer: () ==> ()
testAddAnswer() == (

        -- requirements
        stackOverflow.clear();
        stackOverflow.signup("user1", "pass1");
        stackOverflow.login("user1", "pass1");
        stackOverflow.addQuestion("title1","text1");

        -- test
        stackOverflow.addAnswer(1, "text1");

        assertTrue(
                exists1 q in set elems stackOverflow.questions & (q.id = 1) and
                exists1 a in set elems q.answers & (
                        a.id = 1 and
                        a.text = "text1"
                )
        );
);


/**
* Test upvoteAnswer
*/
private testUpvoteAnswer: () ==> ()
testUpvoteAnswer() == (

        -- requirements
        stackOverflow.clear();
        stackOverflow.signup("user1", "pass1");
        stackOverflow.login("user1", "pass1");
        stackOverflow.addQuestion("title1","question1");
        stackOverflow.addAnswer(1, "answer1");

        -- test
        stackOverflow.upvoteAnswer(1);
--    stackOverflow.upvoteAnswer(2);

        assertTrue(
                exists1 q in set elems stackOverflow.questions & (q.id = 1) and
                exists1 a in set elems q.answers & (
                        a.id = 1 and
                        a.score = 1
                )
        );
);


/**
* Test downvoteAnswer
*/
```

```
private testDownvoteAnswer: () ==> ()
testDownvoteAnswer() == (

        -- requirements
        stackOverflow.clear();
        stackOverflow.signup("user1", "pass1");
        stackOverflow.login("user1", "pass1");
        stackOverflow.addQuestion("title1","question1");
        stackOverflow.addAnswer(1,"answer1");

        -- test
        stackOverflow.downvoteAnswer(1);
--    stackOverflow.downvoteAnswer(2);

        assertTrue(
                exists1 q in set elems stackOverflow.questions & (q.id = 1) and
                exists1 a in set elems q.answers & (
                        a.id = 1 and
                        a.score = -1
                )

        );
);


/**
* Test unvoteAnswer
*/
private testUnvoteAnswer: () ==> ()
testUnvoteAnswer() == (

        -- requirements #1
        stackOverflow.clear();
        stackOverflow.signup("user1", "pass1");
        stackOverflow.login("user1", "pass1");
        stackOverflow.addQuestion("title1","question1");
        stackOverflow.addAnswer(1,"answer1");
        stackOverflow.upvoteAnswer(1);
        stackOverflow.downvoteAnswer(1);
        stackOverflow.upvoteAnswer(1);

        -- test #1
        stackOverflow.unvoteAnswer(1);
--    stackOverflow.unvoteAnswer(2);

        -- requirements #2
        stackOverflow.downvoteAnswer(1);

        -- test #2
        stackOverflow.unvoteAnswer(1);

        assertTrue(
                exists1 q in set elems stackOverflow.questions & (q.id = 1) and
                exists1 a in set elems q.answers & (
                        a.id = 1 and
                        a.score = 0
                )

        );
);


/**
* Test upvoteQuestion
*/
private testUpvoteQuestion: () ==> ()
```

```
testUpvoteQuestion() == (

        -- requirements
        stackOverflow.clear();
        stackOverflow.signup("user1", "pass1");
        stackOverflow.login("user1", "pass1");
        stackOverflow.addQuestion("title1","question1");

        -- test
        stackOverflow.upvoteQuestion(1);
--    stackOverflow.upvoteAnswer(2);

        assertTrue(
                exists1 q in set elems stackOverflow.questions & (
                        q.id = 1 and
                        q.score = 1
                )
        );
);


/**
* Test downvoteQuestion
*/
private testDownvoteQuestion: () ==> ()
testDownvoteQuestion() == (

        -- requirements
        stackOverflow.clear();
        stackOverflow.signup("user1", "pass1");
        stackOverflow.login("user1", "pass1");
        stackOverflow.addQuestion("title1","question1");

        -- test
        stackOverflow.downvoteQuestion(1);
--    stackOverflow.downvoteAnswer(2);

        assertTrue(
                exists1 q in set elems stackOverflow.questions & (
                        q.id = 1 and
                        q.score = -1
                )
        );
);


/**
* Test unvoteQuestion
*/
private testUnvoteQuestion: () ==> ()
testUnvoteQuestion() == (

        -- requirements #1
        stackOverflow.clear();
        stackOverflow.signup("user1", "pass1");
        stackOverflow.login("user1", "pass1");
        stackOverflow.addQuestion("title1","question1");
        stackOverflow.upvoteQuestion(1);
        stackOverflow.downvoteQuestion(1);
        stackOverflow.upvoteQuestion(1);

        -- test #1
        stackOverflow.unvoteQuestion(1);
--    stackOverflow.unvoteQuestion(2);

        -- requirements #2
        stackOverflow.downvoteQuestion(1);

        -- test #2
        stackOverflow.unvoteQuestion(1);

        assertTrue(
               exists1 q in set elems stackOverflow.questions & (
                      q.id = 1 and
                      q.score = 0
               )
        );
);


/**
* Test showQuestionsAndAnswers
*/
private testShowQuestionsAndAnswers: () ==> ()
testShowQuestionsAndAnswers() == (

        -- variables
        dcl num: nat;

        -- requirements
        stackOverflow.clear();
        stackOverflow.signup("user1", "pass1");
        stackOverflow.login("user1", "pass1");
        stackOverflow.addQuestion("title1","question1");
        stackOverflow.addAnswer(1,"text1");
        stackOverflow.addAnswer(1,"text2");
        stackOverflow.upvoteQuestion(1);
        stackOverflow.upvoteAnswer(1);
        stackOverflow.downvoteAnswer(2);

        -- test
        num := stackOverflow.showQuestionsAndAnswers();

        assertTrue(
               num = 3
        );
);


/**
* Test showQuestion
*/
private testShowQuestion: () ==> ()
testShowQuestion() == (

        -- variables
        dcl num: nat;

        -- requirements
        stackOverflow.clear();
        stackOverflow.signup("user1", "pass1");
        stackOverflow.login("user1", "pass1");
        stackOverflow.addQuestion("title1","question1");
        stackOverflow.addAnswer(1,"text1");
        stackOverflow.addAnswer(1,"text2");

        -- test
        num := stackOverflow.showQuestion(1);

        assertTrue(
               num = 3
        );
```

```
);


/**
* Run all tests
*/
public static main: () ==> ()
main() ==
(
	dcl soTest: StackOverflowTest := new StackOverflowTest();

	IO`print("\n--- TESTING ---\n");

	IO`print("testDefaultConstructors: ");
	soTest.testDefaultConstructors();
	IO`print("Success!");

	IO`print("\ntestSignup: ");
	soTest.testSignup();
	IO`print("Success!");

	IO`print("\ntestLogin: ");
	soTest.testLogin();
	IO`print("Success!");

	IO`print("\ntestLogout: ");
	soTest.testLogout();
	IO`print("Success!");

	IO`print("\ntestAddQuestion: ");
	soTest.testAddQuestion();
	IO`print("Success!");

	IO`print("\ntestAddAnswer: ");
	soTest.testAddAnswer();
	IO`print("Success!");

	IO`print("\ntestUpvoteAnswer: ");
	soTest.testUpvoteAnswer();
	IO`print("Success!");

	IO`print("\ntestDownvoteAnswer: ");
	soTest.testDownvoteAnswer();
	IO`print("Success!");

	IO`print("\ntestUnvoteAnswer: ");
	soTest.testUnvoteAnswer();
	IO`print("Success!");

	IO`print("\ntestUpvoteQuestion: ");
	soTest.testUpvoteQuestion();
	IO`print("Success!");

	IO`print("\ntestDownvoteQuestion: ");
	soTest.testDownvoteQuestion();
	IO`print("Success!");

	IO`print("\ntestUnvoteQuestion: ");
	soTest.testUnvoteQuestion();
	IO`print("Success!");

	IO`print("\ntestShowQuestionsAndAnswers: ");
	soTest.testShowQuestionsAndAnswers();
	IO`print("Success!");

	IO`print("\ntestShowQuestion: ");
```

```
                        soTest.testShowQuestion();
                        IO`print("Success!");

                        IO`print("\n--- TESTING ---\n");
                );

end StackOverflowTest
```

5. Model verification

5.1 Example of domain verification

One of the proof obligations generated by Overture is:

No.	PO Name	Type
11	Answers`unvote(User)	legal map application

The code under analysis (with the relevant map application underlined) is:

```
atomic (
            -- add user to set of upvoters
            upvoters := upvoters \ {user};

            -- update score
            score := score + 1;
         );
```

In this case the proof is trivial because the of the precondition '**user in set upvoters or user in set downvoters**'.

5.2 Example of invariant verification

Another proof obligation generated by Overture is:

No.	PO Name	Type
3	Answer`Answer(User,Answer`String)	state invariant holds

The code under analysis (with the relevant state changes underlined) is:

```
   public Answer: User * String ==> Answer
      Answer(user, text1) == (

         author  := user;
         text := text1;
         id := nextid;
         nextid := nextid + 1;

         return self;
      )
      post (
         text = text1 and nextid = id + 1
      );
```

The relevant invariant under analysis is:

```
   inv author <> nil;
```

6. Java Code Generation

The java code was generated through overture and no changes were necessary in order to compile or run the new java project.

7. Conclusions

The model that was developed covers all the requirements, but some improvements that could be made are: the ability to categorize questions or give reputation to users who answer questions with the best answers.

Group Participation:

José Costa: 50%

Tiago Silva: 50%

8. References

1. Validated Designs for Object-oriented Systems, J. Fitzgerald, P.G. Larsen, P. Mukherjee, N. Plat, M. Verhoef, Springer, 2005
2. VDM-10 Language Manual, Peter Gorm Larsen et al, Overture Technical Report Series No. TR-001, March 2014
3. Overture tool web site, https://overturetool.org
4. Modelio tool web site, https://www.modelio.org