Available online at www.sciencedirect.com
ScienceDirect
ELSEVIER
Electronic Notes in Theoretical Computer Science 183 (2007) 3–20
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Model-checking Driven Design of Interactive Systems

Antonio Cerone[1] and Norzima Elbegbayan[2]

International Institute for Software Technology
United Nations University
Macau SAR China

Abstract

This paper describes a model-checking based methodology to detect systematic errors commonly made by non-expert users. The human and computer components of the systems are modelled separately. The human component consists of a general model of the user's cognitively plausible behaviour, which can be then refined into specific instances of behaviour that reflect relevant aspects of users' personalities and skills. We consider, as a case study, a formal model of an online interactive tool that enables conference attendees to share thoughts and reactions and select matching attendees to start communication with. Starting from the initial system design, a model-checking technique is used to highlight system vulnerabilities that arise from interactions with non-expert users and may lead to security violations. The results of the analysis are exploited to improve the design by introducing safeguards that reduce or even prevent security violations.

Keywords: formal verification, human behaviour, usability, user error, social computing, process calculi, model-checking.

1 Introduction

The widespread use of computers in safety-critical and security systems increases the need for human-computer interaction to be designed in a way that reduces the likelihood of human errors.

Human Reliability Assessment (HRA) techniques, which mostly emerged in the 1980's, have been widely used in the analysis of safety-critical systems but have shown little success when applied to the safety assessment of user interface design [11]. In the 1990's increasing use of formal methods has yielded more objective analysis techniques [6], which, however, mainly addressed safety-critical aspects of Interactive Systems (IS) where the human component is represented by operators with expected expertise and skills. Moreover, the operator behaviour was often

[1] Email: antonio@iist.unu.edu
[2] Email: norzima@iist.unu.edu

doi:10.1016/j.entcs.2007.01.058

modelled as defined by the interface requirements. In reality, however, the user interacting with the system does not necessarily behave as it was expected while designing the interface, and errors are actually the very result of an unexpected user behaviour that emerge through the interaction. To best capture such an emergent behaviour, a model of the operator must specify the *cognitively plausible behaviour*, that is all the possible behaviours that can occur, and that involve different cognitive processes [1]. The model must take into account all relationships between user's actions, user's goals and the environment.

A number of researchers have explored the use of formal models to understand how cognitive errors can affect user performance. Rushby [12] models the behaviour of a forgetful operator who follows a warning display light or a non-forgetful operator without warning lights and checks for an emergent *mode confusion*. Curzon and Blandford [5] focus on goal-based interactions and model the behaviour of a user who assumes all tasks completed when the goal is achieved, but forgets to complete some important subsidiary tasks *(post-completion error)*.

In this paper we focus on IS intended for use among large groups of people, with communication, collaboration, information exchange and interest matching as the main goals [8,9]. In such a context there is no concern about safety, but the system must be easy to use regarding learnability and efficiency as well as guarantee confidentiality and other security properties, as required by legal and community policies. The complexity of the interface should be acceptable for any level of user's skill, and the system should not discourage users with its deficient or inconvenient operability. Moreover, there is little *a priori* knowledge about the user's behaviour and experience in using the system, and about more general user's skills in dealing with computers and the Internet. It is important, therefore, to set the most pessimistic scenario, in which the user is *non-expert*, with minimal skills, and to explore alternative user behaviours corresponding to a variety of attitudes and personalities. Mode confusion and post-completion errors must be considered very likely to occur. On the other hand, the system must be designed to guide user's actions and decisions by offering appropriate options in stages of interaction.

We use modelling techniques developed in previous work [2], which are based on the CSP process algebra [10] and on temporal logic, first to define the user's goals and the interface as separate processes, and then to compose such processes in parallel and analyse the emergent interaction, looking for security vulnerabilities. Analysis is carried out by specifiying the properties in temporal logic and using the Concurrency Workbench of the New Century (CWB–NC) [4] model-checker to verify properties against the model. We illustrate the approach on a simple case study based on a web-based online interactive tool that enables conference attendees to communicate and share their thoughts and reactions to a shared event [7].

Section 2 introduces a scenario that may occur during a workshop or conference, in which conference attendees share thoughts and opinions through a web interface motivated by various goals (Section 2.1). We also assume a basic structure of the web interface accessible through a login mechanism (Section 2.2).

Section 3 briefly introduces the CSP notation used throughout the paper and

describes the model of the user behaviour in terms of three possible goals that may motivate and lead the user in interacting with the system: expressing own ideas in the forum (Section 3.1), establishing contact with a matching user (Section 3.2) and gathering information about other users (Section 3.3).

Section 4 presents an initial design of the system as the parallel composition of two CSP processes (Section 4.3), one modelling the user privileges (Section 4.1) and the other modelling the web pages (Section 4.2).

Section 5 identifies possible security threats (Section 5.1), defines a security property which aims to prevent security violations (Section 5.2), introduces assumptions on the expertise and forgetfulness of a typical user (Section 5.3), shows how to analyse the system design under the given assumptions with respect to the security property (Section 5.4), and describes possible improvements of the design to reduce (Section 5.5), or even completely overcome (Section 5.6), the vulnerabilities of the initial design.

2 Case Study: A Conference System

This tool consists of a web-based interface which could be a part of a bigger system, that features a simple discussion forum and a member list. Through web pages users gather information on a conference (or some other events) and find/contact other users who are likely to match their interests. The tool, however, does not feature a *dating service* [9]. *Matching decisions* are instead explicitly made by the user.

2.1 Scenario

We start considering a common scenario that may occur at a workshop or conference. We use the word *user* to identify the main subject of our scenario. After a lecture or speech a user would like to meet other attendees to discuss impressions or reactions to the attended presentation. Such attendees might either be working on similar projects as our user or have similar thoughts about the topic of the presentation. A lecture usually involves a large number of attendees and every single attendee could have a different opinion about the topic. In series of lectures attendees do not have many chances to communicate with one another and ask opinions. Therefore it is important to allow the user to search and initiate communications before attending the conference and in order to make in advance plans and appointments for meetings to be held while being at the conference.

A conference has a website dedicated to sharing and discussing ideas and reflections about talks and seminars. On such a website users can set up their own profiles and browse other users' profiles in order to decide whether to contact them. The purpose of the system is to help people to meet and share their opinions and reactions to lectures or other events that will be (or were) held during the conference.

The conference web site contains all information about the conference, including lists of lectures and events. We assume that users have already started sharing their

thoughts and opinions well before the beginning of the conference. In a typical situation, the user accesses the web site and scans through the lists of events and lectures, and reads the lecture notes and abstracts of papers and presentations. After **logging in** the user can **set up** a profile, which includes the choice of some keywords to represent the user's professional and research interests as well as ideas and thoughts. The user can also:

read messages posted by other users to get a general view of other people's reaction to a recent event/presentation as well as to look for users with matching thoughts and opinions;

post messages to share thoughts and opinions on a recent event/presentation;

reply to messages posted by other users to support or try to confute their thoughts and opinions;

read profiles of other users to understand whether they have matching interests, ideas and thoughts;

contact users who are believed to represent good matches;

logout from the system.

We assume that every post and reply has a link where all related posts and replies are listed.

The user may have different motivations to use the conference system. These motivations, in general, depend on the user's personality, social skills, familiarity with the topics, research and professional interests, as well as practical issues such as availability of time.

Motivations determine the users' goals in using the conference system. Depending on the specific goals, the user may exploit different services provided by the system. In this paper we analyse the behaviour of the user in relation to three specific goals:

gathering information by just browsing through posted messages and user profiles;

establishing contacts with users who represent good matches;

expressing ideas by just posting messages and replying to messages (after reading them).

2.2 Web Interface

The informal description of the user's interaction with the system presented in Section 2.1 highlights three basic user statuses:

- the user has **not logged in**;
- the user has **logged in**, but has **not set up a profile**;
- the user has **set up a profile**.

We assume that

(1) only logged-in users can set up their profiles and read a user profile and a message;

(2) only users who have set up a profile can post or reply to a message and contact other users.

Therefore, the user status changes when the user logs in, sets up a profile and logs out.

The system interface consists of three main web pages

- a **Home** page to set up the user profile and browse general information about the conference;
- a page to browse **User Profiles** set up by other users;
- a **Forum** page to post own messages and browse messages posted by other users;

which are linked to two additional pages. The **User Profiles** page is linked to

- the **Profile** page that allows the user to view and analyse a specific profile and, when a matching is found, contact the corresponding user.

The **Forum** page is linked to

- the **Message** page that allows the user to read a specifc message and, if this is found interesting, to reply to it.

These two additional pages are mutually linked because every message has an author who must have set up a profile, due to assumption (2) above. Therefore, messages are linked to the author's profile. Similarly, a profile may be linked to message(s), if the corresponding user has already posted (or replied to) any.

3 Modelling User Behaviour

The notation that we are going to use throughout the paper is based upon Hoare's CSP notation for describing Communicating Sequential Processes [10]. We use the CWB–NC [4] syntax for CSP:

- "`a -> X`" means that action `a` occurs and then process `X` starts;
- "`X [] Y`" is the choice between processes `X` and `Y`;
- "`X [| S |] Y`" is the parallel composition of processes `X` and `Y` with synchronisation set `S`.

The *synchronisation set* defines the set of actions that must synchronise within the parallel composition.

Our model of the user behaviour focusses on the three user goals introduced in Section 2.1. The specific goal the user has in mind will drive the choice of the appropriate actions, among those allowed by the web interface. For example, if the goal is *gathering information*, the user will just need to browse through posted messages and other user profiles, whereas if the goal is *establishing contacts*, the user will eventually need to explicitly contact another user. To achieve any goal the user always needs to login in the system (assumption (1) in Section 2.2). This

ensures that only authorised users may enter the system initially.

```
proc User = ( login -> AuthorisedUser ) [] UnauthorisedUser

proc AuthorisedUser = goal -> ( ( gather_info -> GatherInfo )
                              [] ( establish_contact -> EstablishContact )
                              [] ( express_ideas -> ExpressIdeas ) )
```

After achieving the goal, the authorised user can either logout or choose a new goal and continue the interaction session with the system. In principle, a *cognitively plausible behaviour* [1] must include the situation in which the user may leave the interaction session unattended at any time, independently of whether the goal is achieved or not. However, such a situation is unlikely to occur when the user focusses on achieving the goal, but it is much more plausible after the goal is achieved. It is actually common that the user assumes all tasks completed when the goal is achieved, but forgets to complete some important subsidiary tasks (*post-completion error*) [5], such as logging out of the system. Therefore we assume that

(3) the user will not logout and will not leave the interaction session unattended while trying to achieve a goal, unless failing to perform an action needed to achieve the goal;

(4) after achieving the goal, the user may forget to logout and leave the interaction session unattended.

```
proc GoalAchieved = AuthorisedUser [] Leave

proc Leave = unattended -> ( ( short_delay -> UnauthorisedUser )
                           [] ( long_delay -> UnauthorisedUser ) )
          [] ( logout -> User )

proc UnauthorisedUser = ( try_setup -> UnauthorisedUser )
                      [] ( try_contact -> UnauthorisedUser )
                      [] ( try_reply -> UnauthorisedUser )
                      [] ( try_post_a_message -> UnauthorisedUser )
                      [] ( try_read_a_message -> UnauthorisedUser )
                      [] ( try_read_a_profile -> UnauthorisedUser )
                      [] ( success -> UnauthorisedUser )
                      [] ( failure -> UnauthorisedUser )
                      [] ( logout -> User)
```

After achieving the goal, the authorised user may pursue a new goal (state `AuthorisedUser`) or leave the session (state `Leave`). In the latter case, the user may either logout (action `logout`) or leave the session unattended (action `unattended`) without logging-out. An unattended open session, after a certain time, which may be short (action `short_delay`) or long (action `long_delay`), may be taken over by an unauthorised user (in state `UnauthorisedUser`), who can try to perform any action.

Note that an (authorised or unauthorised) user may just try to perform the intended action; whether such an action will succeed (action `success`) or fail (action `failure`) depends on the system with which the user interacts.

3.1 Expressing Ideas

Posting messages in the forum and replying to already posted messages are ways of expressing one's ideas.

In general, the goal can be achieved (action `goal_achieved`) by either posting a message (action `try_post_a_message` followed by `success`) or replying (action

`try_reply` followed by `success`) to a message, possibly after reading such message (action `try_read_a_message` followed by `success`). Due to assumption (2) in Section 2.2 action `try_setup` may also need to be performed.

Note that, in general, even if we assume that the user has the intention to read a message before replying to it, we cannot assume that such intention will be always implemented in the right sequence of actions. It may happen that the user reads several messages before replying to any of them and then, intending to reply to some of them, may erroneuosly reply to a message which was not previously read.

```
proc ExpressIdeas = ( try_setup -> success -> ExpressIdeas
                              [] failure -> ( ExpressIdeas [] Leave ) )
                  [] ( try_post_a_message -> ( success -> goal_achieved -> GoalAchieved
                                             [] failure -> ( ExpressIdeas [] Leave ) ) )
                  [] ( try_read_a_message -> ( success -> ExpressIdeas
                                             [] failure -> ( ExpressIdeas [] Leave ) ) )
                  [] ( try_reply -> ( success -> goal_achieved -> GoalAchieved )
                                 [] failure -> ( ExpressIdeas [] Leave ) )
```

At any stage of the interaction, the user may fail (action `failure`) to perform an action. There are four possible ways in which the user may react to such a failure:

(i) try to repeat the failed action (remaining in state `ExpressIdeas` and repeating the same action);

(ii) try an alternative action (remaining in state `ExpressIdeas` and performing a different action);

(iii) leave the interaction session unattended (moving to state `Leave` and performing `unattended`);

(iv) log out from the system (moving to state `Leave` and performing `logout`).

In general the choice of reaction depends on the user's personality and familiarity with computer systems, as well as time availability.

3.2 Establishing contacts

In order to establish contact with a matching user it is necessary to explicitly contact that user. In general, the user who wishes to establish contact may have already collected outside the system all necessary information to select a matching user and use the system just to contact such a matching user.

```
proc EstablishContact = ( try_setup -> ( success -> EstablishContact
                                       [] failure -> ( EstablishContact [] Leave ) ) )
                      [] ( try_read_a_profile -> ( success -> EstablishContact
                                                 [] failure -> ( EstablishContact [] Leave ) ) )
                      [] ( try_read_a_message -> ( success -> EstablishContact
                                                 [] failure -> ( EstablishContact [] Leave ) ) )
                      [] ( try_contact -> ( success -> goal_achieved -> GoalAchieved
                                          [] failure -> ( EstablishContact [] Leave ) ) )
```

There are two ways for gathering information to help select a matching user: reading other user's profiles (action `try_read_a_profile`) and reading messages (action `try_read_a_message`). The user who wishes to establish contact will keep reading profiles and messages (remaining in state `EstablishContact`) until a matching user is found, before trying to contact such a matching user (action `try_contact`). However, action `try_contact` may be immediately performed without any iteration

of the information gathering loop, if the information gathering process has been performed outside the system. Obviously, the `goal_achieved` action must be preceeded by the `try_contact` action. Due to assumption (2) in Section 2.2 action `try_setup` may also need to be performed.

As for the previous goal, at any stage of the interaction, the user may succeed or fail in performing an action.

3.3 Gathering Information

Gathering information about other users may not only be a means to select a matching user but also be the final goal to achieve. If gathering information is the actual user's goal, then each of the `try_read_a_profile` and `try_read_a_message` actions may either be an iteration of the information gathering loop (in state `GatherInfo`) or lead to the `goal_achieved` action, which is performed when the user has collected all needed information.

```
proc GatherInfo = ( try_read_a_profile -> ( success -> ( GatherInfo
                                                  [] goal_achieved -> GoalAchieved )
                                          [] failure -> ( GatherInfo [] Leave ) ) )
                [] ( try_read_a_message -> ( success -> ( GatherInfo
                                                  [] goal_achieved -> GoalAchieved )
                                          [] failure -> ( GatherInfo [] Leave ) ) )
```

4 Initial System Design

4.1 Model of User Privileges

The three basic user statuses highlighted in Section 2.2 can be formalised by three CSP processes as follows.

```
proc Priviliges = ( login -> (( noprofile -> enter -> NonMember )
                        [] ( profile -> enter -> Member ) ) )

proc NonMember = ( try_setup -> success -> Member )
               [] ( logout -> Priviliges )
               [] ( try_read_a_profile -> NonMember )
               [] ( try_read_a_message -> NonMember )
               [] ( success -> NonMember )

proc Member = ( try_read_a_profile -> Member )
            [] ( try_read_a_message -> Member )
            [] ( try_post_a_message -> Member )
            [] ( try_reply -> Member )
            [] ( try_contact -> Member )
            [] ( logout -> Priviliges )
            [] ( success -> Member )
```

Process `Priviliges` defines the initial state, in which the user has not logged in yet. After the user logs in (action `login`), the system checks whether the user has already set a profile (action `profile`) or not (action `noprofile`). If the user has not set a profile yet, the state changes to `NonMember`, otherwise it changes to `Member`. These two states define the two user privileges that correspond to assumptions (1) and (2) in Section 2.2. The purpose of action `enter` is to move to the state corresponding to the appropriate user privilege and to activate the web interface described in Section 4.2. User privileges can be changed by the successful execution of action `try_setup` (from `NonMember` to `Member`). Action `logout` leads back to the initial state (`Priviliges`).

Note that setting up a profile is necessary to achieve the expressing ideas and establishing contacts goals, but not to achive the gathering information goal.

4.2 Model of the Web Interface

The model of the web interface consists of six states.

```
proc Interface1 = ( enter -> Home1 )

proc Home1 = ( users -> UserProfiles1 )
          [] ( forum -> Forum1 )
          [] ( try_setup -> success -> Home1 )
          [] ( logout -> Interface1 )

proc UserProfiles1 = ( forum -> Forum1 )
                  [] ( try_read_a_profile -> success -> AProfile1 )
                  [] ( home -> Home1 )

proc AProfile1 = ( back_to_users -> UserProfiles1 )
              [] ( try_read_a_message -> success -> AMessage1 )
              [] ( try_contact -> success -> AProfile1 )

proc Forum1 = ( try_read_a_message -> success -> AMessage1 )
           [] ( users -> UserProfiles1 )
           [] ( try_post_a_message -> success -> Forum1 )
           [] ( home -> Home1 )

proc AMessage1 = ( try_read_a_profile -> success -> AProfile1 )
              [] ( back_to_forum -> Forum1 )
              [] ( try_reply -> success -> AMessage1 )
```

In the initial state (`Interface1`) the home page of the web interface is activated by action `enter` (and the subsequent change to state `Home1`), which ends the procedure to check the user privileges described in Section 4.1.

The other states model the five web pages described in Section 2.2. Actions `users`, `forum`, `home`, `back_to_users` and `back_to_forum` allow the user to freely navigate through the five web pages. Note that action `logout` is only possible from state `Home1`. This means that the user has always to go back to the home page in order to be able to logout.

Since logout and navigation are determined by just clicking on buttons, we implicitly assume that the user will never fail to perform such actions. Therefore, we do not need to express logout and pure navigation actions as attempts (by prefixing their names with "`try_`"), neither to associate them with success or failure actions.

4.3 Overall System Model

The overall system is expressed by process `SYSTEM1` given by the parallel composition of the user privileges (initially in state `Priviliges`), the web interface (initially in state `Interface1`) and the user (process `User`)

```
proc SYSTEM1 = ( ( Priviliges [| {enter, try_read_a_profile, try_read_a_message,
try_post_a_message, try_reply, try_contact, try_setup, success, logout} |] Interface1 )
[| {login, try_setup, try_read_a_message, try_read_a_profile, try_contact,
try_post_a_message, try_reply, logout, success, failure} |] User )
```

The `Priviliges` and `Interface1` processes must synchronise on all actions that can be eventually performed by the `Priviliges` process apart from the `profile` and `noprofile`, which define the checking procedure modelled by the `Priviliges` process (they are internal actions of `Priviliges`).

The process originated by the parallel composition of processes `Priviliges` and

`Interface1` is then composed with the `User` process. In this second parallel composition, the synchronisation must include all user actions that define interactions with the interface. Note that we have also included the `failure` action, which does not occur in the `Priviliges` and `Interface1` processes (and therefore neither in their parallel composition) in the sychronisation set. This prevents the overall system from performing the `failure` action, so modelling the following assumption

(5) the user never fails to perform an intended action that is immediately available on the current web page.

Such an assumption was implicitly made for logout and navigation actions in Section 4.2; it is here explicitly extended to all user actions. The purpose of this assumption is to show that the design weaknesses captured by the model-checking analysis presented in Section 5.4 are independent of the ability of the user in successfully performing a single interaction with the interface.

5 Improving the System Design

In this section we use the CWB–NC to analyse the interaction between the user behaviour model defined in Section 3 and various versions of the web interface defined in Section 4. The analysis of the original web interface defined in Section 4 highlights security vulnerabilities, which are then partly or entirely overcome in the next versions.

5.1 Security Threats and Safeguards

According to assumptions (3) and (4) in Section 3 the user will not logout and will not leave the interface unattended before achieving the goal, unless failing to perform an action needed to achieve the goal. However, after achieving the goal, the user may forget to logout and leave the interface unattended without coming back to use it. Such a situation may lead to *security violations*. The interface is supposed to be designed with the aim to minimise the likelihood that an unattended session is exploited by a non-authorised user to access profiles (*confidentiality violation*) or to pretend to be the logged-in user (*masquerading*).

Ideally, we would like an unattended session to automatically logout *on time* to prevent security violations. However, in practice, we can just introduce *safeguards* that minimise the likelihood of security violations, in a way that does not introduce much degradation in the quality and performance of the services provided to the user. In order to find the right balance between security and the quality and performance of services, it is important to analyse the user attitudes and behaviours while interacting with the system. Specific attitudes and behaviours may actually reduce the likelihood of some threats and increase the likelihood of others. For example, panicking when the planned action does not appear immediately available on the current web page is an attitude that may lead to the behaviour of leaving the session unattended, so causing a security threat. On the other hand, the attitude of always checking that all tasks have been completed after achieving a goal reduces

the likelihood of forgetting to logout before leaving the session.

It is therefore sensible to introduce safeguards only to prevent the most likely threats, that is, those threats that are more likely to occur given specific assumptions about user attitudes and behaviours.

5.2 Security Properties

An important requirement that our system should meet is that an open session is never left unattended. Meeting such a requirement would definitely prevent security violation from occurring.

However, it would be useless to state a property that heavily depends on the user's limitations, in terms of memory, attention, etc., and can be only partly affected by the way the interface constrains the user behaviour. Assuming that the session can actually be left unattended in the way expressed by assumptions (3) and (4) in Section 3, we have to ensure that no unauthorised user is able to exploit the situation by performing actions (in state `UnauthorisedUser`) that may generate security violations. In particular, we would like to ensure that an unauthorised user cannot access other users' profiles and messages (*confidentiality violation*), by successfully performing actions in

$$\mathcal{C} = \{\texttt{try\_read\_a\_profile}, \texttt{try\_read\_a\_message}\},$$

or pretend to be the logged-in user (*masquerading*) by successfully performing actions in

$$\mathcal{M} = \{\texttt{try\_setup}, \texttt{try\_post\_a\_message}, \texttt{try\_reply}, \texttt{try\_contact}\}.$$

Set $\mathcal{S} = \mathcal{C} \cup \mathcal{M}$ consists of exactly all actions that may lead to security violations.

According to assumptions (3), (4) and (5), a session can only be left unattended after achieving the goal. Since our model implicitly assumes that goals are determined only by authorised users, the critical time interval during which security violations may occur starts at the `goal_achieved` action and terminates at action `goal`, when the next goal is determined by an authorised user, or at action `logout`. In fact, determining a new goal cannot occur in the `UnauthorisedUser` state, and action `logout` exits the `UnauthorisedUser` state.

Using CWB–NC syntax [4], a property asserting that no security violation may occur between the achievement of the goal and the next logout or determined goal can be formalised as follows.

```
prop secure = ( A G ( {goal_achieved} -> ( {-success} W {goal,logout} ) ) )
```

In CWB-NC syntax atomic formulae have form `{`*action_list*`}` or `{-`*action_list*`}`; the former is satisfied by any action that appears in *action_list*, whereas the latter is satified by any action that does not appear in *action_list*. Note that a deadlock process satisfies `~{`*action_list*`}`, where `~` is the negation connective, but does not satisfy `{-`*action_list*`}`. The `W` temporal operator (*weak until*) ensures that formula `pWq` is true if and only if `p` is continuously true forever or *until* `q` is true.

Property `security` prevents action `success` from occuring between achieving a goal and pursuing a new goal or logging-out, that is, it prevents any action in

$\mathcal{S}$ from being successfully performed between achieving a goal and pursuing a new goal or logging-out. Therefore property `security` guarantees that

- *the authorised user* will always be able to logout or set a new goal;
- if the session is left unattended (action `unattended`) after achieving a goal (action `goal_achieved`), then *no unauthorised user* who may take over the session can successfully perform actions in $\mathcal{S}$.

Note that using `{-success}` rather than `~{success}` in formula `secure` ensures that a deadlock occurring after the goal is achieved falsifies the `secure` formula. We will exploit this in Section 5.3 by checking whether some constrained user behaviours are supported by the interface.

5.3 Constraining the User Behaviour

The users of our system are not supposed to be expert in using interactive systems. In fact, some of them might have very low familiarity with computers. The user behavior model defined in section 3 is a very general one and needs to be restricted to capture specific attitudes and behaviours of non-expert users.

A typical behaviour of a non-expert user after achieving a goal is to look for a way to logout but, if no way is found after a reasonable time, to eventually leave the session unattended. Such a behaviour may be enforced by a process defined as follows.

```
proc NonExpert = ( goal -> NonExpert )
               [] ( home -> NonExpert )
               [] ( users -> NonExpert )
               [] ( forum -> NonExpert )
               [] ( back_to_users -> NonExpert )
               [] ( back_to_forum -> NonExpert )
               [] ( unattended -> NonExpert )
               [] ( logout -> NonExpert )
               [] ( goal_achieved -> Finished)

proc Finished = ( logout -> NonExpert )
              [] ( unattended -> NonExpert )
```

This process has then to be composed in parallel with the system as follows.

```
proc SYSTEM1N = ( SYSTEM1 [| {goal, home, users, forum, back_to_users, back_to_forum,
                              unattended, logout, goal_achieved} |] NonExpert )
```

Apart from `unattended` and `logout`, any other action on which the two processes synchronise cannot occur after the `goal_achieved` action, consistently with the behaviour of a non-expert user described above.

The user defined by the `SYSTEM1N` process may, however, forget to logout and leave the session unattended even if there is a logout mechanism (e.g. a logout button) promptly available on the current web page. This occurs when actions `unattended` and `logout` are both available but `unattended` is chosen. We model an extreme case of a non-forgetful user, who will always choose a `logout` action when available after `goal_achieved`, even when intending afterwards to pursue another goal, and will never leave the session unattended. Such a non-forgetful user may be defined by appropriately synchronising the system with the `NonForgetful` process defined as follows.

```
proc NonForgetful =  ( goal_achieved -> logout -> NonForgetful )
```

```
                [] ( goal -> NonForgetful )
                [] ( logout -> NonForgetful )
```

The appropriate synchronisation is achieved by composing this process, which works as a *constraint*, in parallel with `SYSTEM1N` as follows.

```
proc SYSTEM1NR = ( SYSTEM1N [| {goal_achieved, unattended, logout, goal} |] NonForgetful )
```

Since the two components must also synchronise on `unattended`, and this does not occur in the behaviour of the `NonForgetful` process, the `unattended` action can never occur in the composite process. Note that this would cause a deadlock when composing the `NonForgetful` constraint with a system that does not allow action `logout` to be performed immediately after `goal_achieved`.

5.4 Analysis of the Initial Design

Using the CWB–NC we can verify that `security` does not hold for the `SYSTEM1NR` system. The system appears to be unsecure in spite of having introduced a constraint to ensure that the user will always remember to logout before terminating the interaction with the system. Constraints `NonForgetful` and `NonExpert` allow only action `logout` to be performed immediately after `goal_achieved`, but `logout` is only available in state `Home1` of process `Interface1`, which cannot be the state in which `Interface1` is immediately after `goal_achieved` is performed. This cause a deadlock immediately after `goal_achieved`, which falsifies `security`.

The fact that `security` does not hold for the `SYSTEM1NR` system, in which the non-expert user is constrained to immediately logout, shows that the security vulnerability cannot be due to the forgetfulness of the user and the web interface needs to be improved to address non-expert users.

The problem is that the logout is not available on each web page, but just on the Home page. The users have to properly navigate back to the Home page from the page where the goal has been achieved. This might be quite challenging for a non-expert user. In addition, the presence of a logout button on each web page would be a reminder for the user to logout, so addressing also expert but forgetful users.

We therefore modify the interface by introducing a `logout` action in every state as follows.

```
proc Interface2 = ( enter -> Home2 )

proc Home2 = ( users -> UserProfiles2 )
          [] ( forum -> Forum2 )
          [] ( try_setup -> success -> Home2 )
          [] ( logout -> Interface2 )

proc UserProfiles2 = ( forum -> Forum2 )
                  [] ( try_read_a_profile -> success -> AProfile2 )
                  [] ( home -> Home2 )
                  [] ( logout -> Interface2 )

proc AProfile2 = ( back_to_users -> UserProfiles2 )
              [] ( try_read_a_message -> success -> AMessage2 )
              [] ( try_contact -> success -> AProfile2 )
              [] ( logout -> Interface2 )

proc Forum2 = ( try_read_a_message -> success -> AMessage2 )
           [] ( users -> UserProfiles2 )
           [] ( try_post_a_message -> success -> Forum2 )
```

```
          [] ( home -> Home2 )
          [] ( logout -> Interface2 )

proc AMessage2 = ( try_read_a_profile -> success -> AProfile2 )
            [] ( back_to_forum -> Forum2 )
            [] ( try_reply -> success -> AMessage2 )
            [] ( logout -> Interface2 )
```

The processes that define the composite system are then defined as follows.

```
proc SYSTEM2 = ( ( Priviliges [| {enter, try_read_a_profile, try_read_a_message,
try_post_a_message, try_reply, try_contact, try_setup, success, logout} |] Interface2)
[| {login, try_setup, try_read_a_message, try_read_a_profile, try_contact,
try_post_a_message, try_reply, logout, success, failure} |] User)

proc SYSTEM2N = ( SYSTEM2 [| {goal, home, users, forum, back_to_users, back_to_forum,
                              unattended, logout, goal_achieved} |] NonExpert )

proc SYSTEM2NR = ( SYSTEM2N [| {goal_achieved, unattended, logout, goal} |] NonForgetful )
```

We can now verify, using the CWB–NC, that `security` holds for the `SYSTEM2NR` system.

5.5 Introducing a timeout

A problem with the interface defined by `SYSTEM2` is the lack of any protection for forgetful users. Although adding a direct logout mechanism to each web page may work as a reminder to the user to logout, users might still forget to logout. Property `security` does not actually hold for `SYSTEM2N`.

A way of improving the situation is the introduction of a *timeout* in the interface to force the system to automatically logout if there is no action by the on-line user, within a given time. We modify the interface as follows

```
proc Interface3 = ( enter -> Home3 )

proc Home3 = ( users -> UserProfiles3 )
          [] ( forum -> Forum3 )
          [] ( try_setup -> success -> Home3 )
          [] ( logout -> Interface3 )
          [] ( short_delay -> Home3 )
          [] ( long_delay -> timeout -> logout -> Interface3 )

proc UserProfiles3 = ( forum -> Forum3 )
                  [] ( try_read_a_profile -> success -> AProfile3 )
                  [] ( home -> Home3 )
                  [] ( logout -> Interface3 )
                  [] ( short_delay -> UserProfiles3 )
                  [] ( long_delay -> timeout -> logout -> Interface3 )

proc AProfile3 = ( back_to_users -> UserProfiles3 )
              [] ( try_read_a_message -> success -> AMessage3 )
              [] ( try_contact -> success -> AProfile3 )
              [] ( logout -> Interface3 )
              [] ( short_delay -> AProfile3 )
              [] ( long_delay -> timeout -> logout -> Interface3 )

proc Forum3 = ( try_read_a_message -> success -> AMessage3 )
           [] ( users -> UserProfiles3 )
           [] ( try_post_a_message -> success -> Forum3 )
           [] ( home -> Home3 )
           [] ( logout -> Interface3 )
           [] ( short_delay -> Forum3 )
           [] ( long_delay -> timeout -> logout -> Interface3 )

proc AMessage3 = ( try_read_a_profile -> success -> AProfile3 )
              [] ( back_to_forum -> Forum3 )
              [] ( try_reply -> success -> AMessage3 )
              [] ( logout -> Interface3 )
              [] ( short_delay -> AMessage3 )
              [] ( long_delay -> timeout -> logout -> Interface3 )
```

In every state, apart from the initial `Interface3` state, we have inserted a choice

```
[] ( long_delay -> timeout -> logout -> Interface3 )
```

which implements a timeout occuring only after some time (`long_delay`), which is long enough not to disrupt the short idling periods that users normally have during sessions, and a choice

```
[] ( short_delay -> ... )
```

which implements such a short idling period.

In general, this safeguard does not fully solve the problem of unauthorised accesses. In fact, an unauthorised user can still enter an unattended session before the timeout expires. Let us consider the composite system defined as follows.

```
proc SYSTEM3 = ( ( Priviliges [| {enter, try_read_a_profile, try_read_a_message,
try_post_a_message, try_reply, try_contact, try_setup, success, logout} |] Interface3)
[| {login, try_setup, try_read_a_message, try_read_a_profile, try_contact,
try_post_a_message, try_reply, logout, short_delay, long_delay, success, failure} |] User)

proc SYSTEM3N = ( SYSTEM3 [| {goal, home, users, forum, back_to_users, back_to_forum,
                              goal_achieved, leave, logout} |] NonExpert )
```

We can actually verify using the CWB–NC that `security` does not hold for the `SYSTEM3N` system. However, we can assume that

(6) no authorised user may enter an unattended session within a time period shorter (`short_delay`) than the delay (`long_delay`) that triggers the timeout.

Such an assumption may be modelled as follows.

```
proc QuickTimeOut = ( home -> QuickTimeOut )
                  [] ( users -> QuickTimeOut )
                  [] ( forum -> QuickTimeOut )
                  [] ( back_to_users -> QuickTimeOut )
                  [] ( back_to_forum -> QuickTimeOut )
                  [] ( logout -> QuickTimeOut )
                  [] ( long_delay -> timeout -> logout -> QuickTimeOut )
```

The process that incorporates this assumption is defined as follows.

```
proc SYSTEM3NQ = ( SYSTEM3N [| {home, users, forum, back_to_users, back_to_forum,
                                timeout, logout, short_delay, long_delay} |] QuickTimeOut  )
```

We can verify that `security` holds for the `SYSTEM3NQ` system.

5.6 Introducing authentication

All safeguards introduced in previous sections contribute to reduce the likelihood of security violations, but they do not guarantee the absence of such violations. Those safeguards aim actually to reduce the likelihood that an unauthorised user enters an unattended open session, but there are no explicit mechanisms in the system to prevent an unauthorised user, who has actually entered the session, from performing interactions with the system.

We could modify the web interface by introducing a protection mechanism that requires the user to provide authentication (i.e., to input a password) before performing a specific critical action. For example, we would like to avoid *masquerading threats* by requiring authentication to users who wish to contact other users.

We modify the interface as follows.

```
proc Interface4 = ( enter -> Home4 )

proc Home4 = ( users -> UserProfiles4 )
          [] ( forum -> Forum4 )
          [] ( try_setup ->
                    ( ( authenticated -> setup -> success -> Home4 )
                    [] ( failure -> Home4 ) ) )
          [] ( logout -> Interface4 )
          [] ( short_delay -> Home4 )
          [] ( long_delay -> timeout -> logout -> Interface4 )
          [] ( failure -> Home4 )

proc UserProfiles4 = ( forum -> Forum4 )
                  [] ( try_read_a_profile -> read_a_profile -> success -> AProfile4 )
                  [] ( home -> Home4 )
                  [] ( logout -> Interface4 )
                  [] ( short_delay -> UserProfiles4 )
                  [] ( long_delay -> timeout -> logout -> Interface4 )
                  [] ( failure -> UserProfiles4 )

proc AProfile4 = ( back_to_users -> UserProfiles4 )
              [] ( try_read_a_message -> read_a_message -> success -> AMessage4 )
              [] ( try_contact ->
                       (  ( authenticated -> contact -> success -> AProfile4 )
                       [] ( failure -> AProfile4 ) ) )
              [] ( logout -> Interface4 )
              [] ( short_delay -> AProfile4 )
              [] ( long_delay -> timeout -> logout -> Interface4 )
              [] ( failure -> AProfile4 )

proc Forum4 = ( try_read_a_message -> read_a_message -> success -> AMessage4 )
           [] ( users -> UserProfiles4 )
           [] ( try_post_a_message ->
                    ( ( authenticated -> post_a_message -> success -> Forum4 )
                    [] ( failure -> Forum4 ) ) )
           [] ( home -> Home4 )
           [] ( logout -> Interface4 )
           [] ( short_delay -> Forum4 )
           [] ( long_delay -> timeout -> logout -> Interface4 )
           [] ( failure -> Forum4 )

proc AMessage4 = ( try_read_a_profile -> read_a_profile -> success -> AProfile4 )
             [] ( back_to_forum -> Forum4 )
             [] ( try_reply ->
                     ( ( authenticated -> reply -> success -> AMessage4 )
                     [] ( failure -> AMessage4 ) ) )
             [] ( logout -> Interface4 )
             [] ( short_delay -> AMessage4 )
             [] ( long_delay -> timeout -> logout -> Interface4 )
             [] ( failure -> AMessage4 )
```

We have now introduced new actions to characterise the success of the user in performing each specific "try" action. For example, the success in performing `try_contact` is characterised by action `contact`. Setting up a profile, replying to or posting a message and contacting another user are now allowed only upon successful authentication. For example action `contact` may occur only after action `authenticated`. If action `failure` occurs instead (the authentication fails), then action `contact` cannot occur (no contact can be established).

We assume that

(7) only authorised users can be authenticated.

This is a reasonable assumption, which in password-based authentication corresponds to the assumption that the password is kept secret. Assumption (7) may be defined by a constraint as follows.

```
proc Authorised = ( authenticated -> Authorised )
               [] ( logout -> Authorised )
               [] ( unattended -> UnAuthorised )
```

```
proc UnAuthorised = ( failure -> UnAuthorised )
                []  ( logout -> Authorised )
```

This process models the presence of an authorised user until action `unattended` occurs, and then the presence of an unauthorised user, with no `authenticated` action allowed, until action `logout` occurs. Note that this constraint restricts assumption (5) in Section 4.3 to authorised users only.

The processes that define the composite system are defined as follows.

```
proc SYSTEM4 = ( (Priviliges [| {enter, try_read_a_profile, try_read_a_message,
try_post_a_message, try_reply, try_contact, try_setup, success, logout} |] Interface4)
[| {login, try_setup, try_read_a_message, try_read_a_profile, try_contact,
try_post_a_message, try_reply, logout, short_delay, long_delay, success, failure} |] User)

proc SYSTEM4A = ( SYSTEM4 [| {failure, authenticated, unattended, logout} |] Authorised )
```

We want to prove that if an open session is left unattended (action `unattended`), then a profile cannot be set up (action `setup`), no contact is established (action `contact`), no message is posted (action `post_a_message`), no reply is sent (action `reply`), until at some time during the unattended session a logout (action `logout`) occurs. Such a property can be formalised as follows.

```
prop never_masquerading = A G ( {unattended} ->
   ( ( {-setup,contact,post_a_message,reply} ) W {logout} ) )
```

We can verify that `prop never_masquerading` holds for the `SYSTEM4A` system but does not on hold for `SYSTEM4`.

The web interface might be modified in a similar way to obtain a system protected from confidentiality threats. In this case the request for authentication would be associated with actions `read_a_profile` and `read_a_message` and the property to be verified would be

```
prop confidentiality =  A G ( {unattended} ->
   ( ( {-read_a_profile,read_a_message} ) W {logout} ) )
```

Finally, the web interface could be modified to protect from both confidentiality violation and masquerading. Although these mechanisms guarantee perfect security under assumption (7), it is unlikely that they would all be implemented in a real system similar to the case study we have analysed. However, if it is likely that some sensitive information may be posted on the website, privacy policies might require confidentiality just for that particular information. Then the solution above would guarantee that no unauthorised user can read confidential profiles or messages of other users. Although masquerading is still possible and some users might be driven by the attacker to post confidential information, such information could not be read by the attacker.

6 Conclusion

We have used modelling techniques developed in previous work [2] to separately model the user goals and web interface components of an interactive tool for communication, information exchange and interest matching [7].

The composition step has been carried out using the *constraint-based modelling style* [13,3], in which the behavior of a process is restricted by composing it with a

special process that works as a contraint. This approach has been exploited to model alternative aspects of the behaviour of non-expert users (processes `NonExpert`, `NonForgetful`) as well as assumptions that are needed to set the contexts in which security properties can be expressed (processes `QuickTimeOut`, `Authorised`).

Analysis has been carried out by specifying properties in temporal logic and using several iterations of model-checking to discover security vulnerabilities of the initial design and verify the correctness of the safeguards introduced.

Acknowledgements

The authors would like to thank Dang Van Hung for helpful comments.

References

[1] R. Butterworth, Ann E. Blandford, and D. Duke. Demonstrating the cognitive plausability of interactive systems. *Formal Aspects of Computing*, 12:237–259, 2000.

[2] Antonio Cerone, Peter Lindsay, and Simon Connelly. Formal analysis of human-computer interaction using model-checking. In Bernhard Aichernig and Bernhard Beckert, editors, *Proceedings of the 3rd IEEE International Conference on Software Engineering and Formal Methods*, pages 352–361. IEEE Comp. Soc., 2005.

[3] Antonio Cerone and George Milne. Property verification of asynchronous systems. *Innovations in System and Software Engineering*, 1(1):25–40, April 2005.

[4] Rance Cleaveland, Tan Li, and Steve Sims. The concurrency workbench of the new century. User's manual, SUNY at Stony Brook, Stony Brooke, NY, USA, 2000.
URL: `http://www.cs.sunysb.edu/~{}cwb`.

[5] Paul Curzon and Ann E. Blandford. Formally justifying user-centred design rules: a case study on post-completion errors. In E.A. Boiten, J. Derrick, and G. Smith, editors, *Integrated Formal Methods*, volume 2999 of *Lecture Notes in Computer Science*, pages 461–480. Springer-Verlag, Berlin, Germany, 2004.

[6] Alan John Dix. *Formal Methods for Interactive Systems*. Academic Press, 1991.

[7] Norzima Elbegbayan. Shared reflection — case study on conference support website. Technical Report 342, UNU-IIST, 2006.
URL: `http://www.iist.unu.edu/`.

[8] Joseph F. McCarthy *et al.* Digital backchannels in shared physical spaces: attention, intention and contention. In *CSCW '04: Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work*, pages 550–553, New York, NY, USA, 2004. ACM Press.

[9] Andrew Fiore and Judith S. Donath. Online personals: An overview. In *ACM Computer-Human Interaction*, Vienna, 2004.
URL: `http://smg.media.mit.edu/papers/atf/chi2004_personals_short.pdf`.

[10] C.A.R Hoare. *Communicating Sequential Processes*. International Series in Computer Science. Prentice Hall, 1985.

[11] B. Kirwan. Human error identification in human reliability assessment. Part 1: Overview of approaches. *Applied Ergonomics*, 25(5):299–318, 1992.

[12] John Rushby. Using model-checking to help discover mode confusions and other automation surprises. *Reliability Engineering and System Safety*, 75(2):167–177, February 2002.

[13] C. A. Vissers, G. Scollo, M. van Sinderen, and E. Brinksma. Specification styles in distributed systems design and verification. *Theoretical Computer Science*, 89:179–206, 1991.