Matelas: A Predicate Calculus Common Formal Definition for Social Networking

Nestor Catano[1] and Camilo Rueda[2]

[1] Madeira ITI
Funchal, Portugal
ncatano@uma.pt
[2] Pontificia Universidad Javeriana
Cali, Colombia
crueda@cic.puj.edu.co

Abstract. This paper presents Matelas, a B predicate calculus definition for social networking, modelling social-network content, privacy policies, social-networks friendship relations, and how these relations effect users' policies. The work presented in this paper is part of an ongoing work that aims at using several formal methods tools and techniques to develop a full-fledged social-network service implementing stipulated policies. Although we employed Atelier B to write Matelas, plans are to port it to Event B and to use Rodin to implement the social-network application.

1 Introduction

Over the past years we have experienced a huge development in Internet and communication systems. Internet and technology have changed our lives. They have changed the way we perceive the world, the way we build social relations, the way we approach people, the way we are. Today, many people find easier to share interests with people on the opposite side of the world, people who they have never personally met, than with the neighbour from the opposite house. Social-networks services in the form of web-sites, e.g., Facebook, Sapo, MySpace, LinkedIn, Hi5, have revolutionised the way people socialise. They have become popular tools to allow people to share common interests, and keep-up with friends, family and business connections. Facebook, currently the dominant service, reports 250 million active user accounts, roughly half of which include daily activity [14]. A typical social network user profile features personal information (e.g., gender, birthday, family situation), a continuous stream of activity logged from actions taken on the site (such as messages sent, status updated, games played) and media content (e.g., personal photos and videos). The privacy and security of this information is therefore a significant concern [16]. For example, users may upload media (such as photographs) they wish to share with specific friends, but do not wish to be widely distributed to their network as a whole. However, social network services have conflicting goals. Although respecting the privacy of their client base is important, they must also grow and expand the connections between their users in order to be successful. This is

M. Frappier et al. (Eds.): ABZ 2010, LNCS 5977, pp. 259–272, 2010.

typically achieved by exposing content to users through links such as friends-of-friends, in which content relating to individuals known to a user's friends (but not the user) is revealed. Examples of this behaviour include gaining access to a photo album of an unknown user simply because a friend is tagged in one of the images. Back-doors also exist to facilitate casual connections such as allowing an unknown user to gain access to profile information simply by replying to a message he or she has sent.

We argue that mechanisms for users to enforce restricted access to content in social network applications are urgently needed, and propose the use of formal method techniques to build a core social network application enforcing these policies. Formal methods are based on mathematical formalisms whereby social-networks policies can be expressed in logic unambiguously. Formal methods make possible the use of mathematically-based machinery to support the precise reasoning about the logical description of properties. The work presented in this paper is part of an ongoing research work [12] in which social networking web-sites (e.g. Facebook, Twitter, and Hi5) are used as a living testbed in which formal methods [23] coupled with graph theory and Human Computer Interaction (HCI) techniques are employed to develop more dependable, secure, and crucially trustworthy social network systems. In this paper, we present Matelas[1], a predicate calculus abstract specification layer definition for social networking, modelling social-network content, privacy policies, social-network friendship relations, and how these effect the policies with regards to content and other users in the network. Our work builds on Jean-Raymond Abrial's "parachute strategy" of building systems [1] in which a system is first considered from a very abstract and simple point of view, with broad fundamental observations, and then details are added to describe more precise behaviour of the system. As future work, we envision to refine Matelas into a social-network core application that adheres to stipulated policies and definitions. Hence, from our predicate calculus model definition of social networks, a code-level model will be attained while applying successive refinement steps.

In the following, Section 2.1 presents the context of the work presented in this paper. Section 2.2 gives a brief introduction to the B method for software development. Section 3 presents Matelas. Section 4 discusses related work on the use of formal methods for social-networking, and Section 5 presents conclusions and discusses future work and underlying challenges.

2 Preliminaries

2.1 A Formal Framework for Social Networking

The work presented in this paper is part of an ongoing research work in which social networking web-sites are used as a living testbed in which formal methods coupled with graph theory and Human Computer Interaction (HCI) techniques are employed to develop more dependable, secure, and crucially trustworthy

[1] Matelas is the French word for the English word mattress.

social network systems. This ongoing work builds on the correct definition of Matelas. We plan to *refine* Matelas to a social network core application that adheres to stipulated policies [17,18]. The refined core application will serve as a common trunk to which social network features will be plugged-in. While this core is minimal in functionality, it will be considerably extended by incorporating plug-ins. This will be achieved by developing a framework where the plug-ins, written in popular programming languages such as Java or C, can demonstrate their adherence to the policies stipulated by Matelas. This will be achieve by using Proof Carrying Code (PCC) [20]. PCC is a technique in which a code consumer (the social network core application) establishes a set of rules (privacy and security policies) that guarantee that externally produced programs (the plug-ins) can safely be run by the consumer. In addition to the code to be executed, the code producer must provide a proof of adherence to the set of rules defined by the code consumer. This proof is run by the code consumer once, using a proof validator, to check whether the proof is valid and therefore the external program is safe to execute. Hence, the problem of extending our social network core application with plug-ins can be regarded as a producer-consumer problem in which the code producer (the plug-in) must adhere to security and privacy policies specified by Matelas, and as a consequence to the policies of the social network core application.

Furthermore, while a social network application or plug-in may adhere to stipulated policies, these policies might be insufficient to avoid human error. While a plug-in may not access users date of birth without explicit authorisation, it is still possible for users to inadvertently give such authorisation. This may happen either by accident or, most likely, due to the complexity of the settings and preferences interface that the user is asked to interact with. Hence, as part of our whole work on social networking, we will augment our correct social network core and plug-ins with understandable human interfaces that enable end users to express their privacy policies and preferences, as well as to review and modify them.

2.2 The B Method for Software Development

In the *refinement* calculus strategy for software development, the process of going from a system specification to its implementation in a machine goes through a series of stages. Each stage adds more details to the description of a system. Each stage can thus be seen as a model of the system at a particular level of abstraction. Models at each level serve different purposes. At higher levels models are used to state and verify key system properties. At lower levels models are used to implement the system behaviour. It is crucial that models at each stage are coherent with the system specification, *i.e.*, that the simulation obeys the specification properties. A model M_{i+1} at stage $i+1$ is said to be a refinement of a model M_i at stage i when the states computed by M_i and M_{i+1} at each given step obey a so-called "gluing invariant" stating properties for the joint behaviour of both models. A refinement step generates *proof obligations* that must be formally verified in order to assert that a model M_{i+1} is indeed a

refinement of a model M_i. These are sufficient conditions to guarantee that, although at different levels of abstraction, both are models of the same system. Correctness of the whole development process is thus ensured ([3]).

In the B method ([1], [25]) models are so-called *machines* composed of a static part defining observations (variables, constants, parameters, etc) of the system and their invariant properties, and a dynamic part defining operations changing the state of the system. Each operation must maintain the invariant property. In B, the language for stating properties, essentially predicate logic plus set theory, and the language for specifying dynamic behaviour (*i.e.* programs) are seamlessly integrated. A significant feature of the B system modelling approach is the availability of automatic verification tools such as B-Tools [10], or Atelier B [5], and model-checking simulators such as ProB [22].

A derivative of the B method is Event B [3]. Event B models are developments of discrete transition systems. They are composed of *machines* and *contexts*. These correspond, roughly, to a B method *machine* whose static part (except variables and their invariants) is transferred to a different module (the context). B method operations are replaced in Event B machines by *events*. In B method machines, operations are *invoked*, either by a user or by another machine, whereas in Event B, an event can be fired some condition (its *guard*) holds. Three basic relations are used to structure a model. A machine *sees* a context and can *refine* another machine. A context can *extend* another context. Events have two forms, as shown in Table 1. The "when" form of event executes the action A_1 when the current value of the system variables v satisfies the guard G_1. The "any" form of event executes action A_2 when there exists some value of x satisfying the guard G_2. Proof obligations require invariants to hold after executing the actions.

Table 1. Events

any x **where** $G_2(x, v)$ **then** $A_2(x, v)$ **end**	**when** $G_1(v)$ **then** $A_1(v)$ **end**

3 Matelas

Matelas is a B abstract specification for social networking that models social-network content, social networks friendship relations, and privacy on content. Privacy issues have generated a bunch of theories, and approaches [26]. Nonetheless, as stated by Anita L. Allen in [4], "while a no universally accepted definition of privacy exists, definitions in which the concept of access plays a central role have become increasingly commonplace". Following Allen's approach, we model

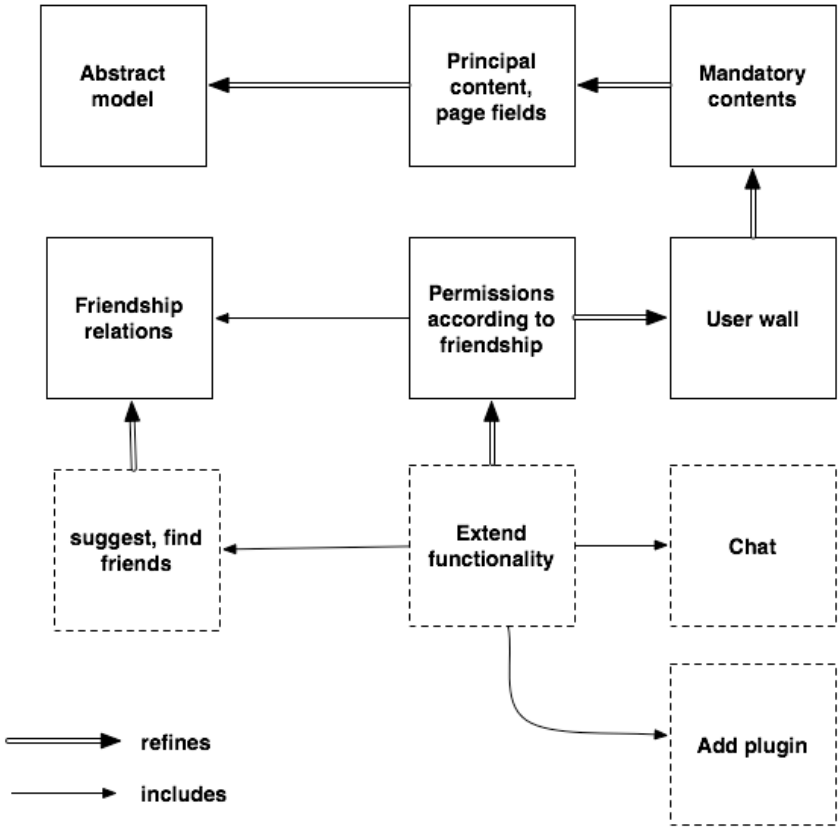

Fig. 1. System architecture. Dashed boxes are components not yet defined

privacy with the aid of a relation that registers users' access privileges on social-network resources, and a content ownership relation.

Matelas distinguishes five rather independent aspects of social networks, namely, user content and privacy issues, friendship relation in social-networks, user content and how it is affected by friendship relations, external plug-ins, and the user interface. At the present stage our model comprises six B (implemented) components: an abstract machine, four refinements and an included machine. We plan to refine Matelas to a social-networking core system. What each implemented machine observes of the system is shown in Table 2. The architecture of the core system is shown in Figure 1, with dashed boxes representing components not yet defined. The fourth refinement in Table 2 includes the *social_friends* machine. A first abstract model views the system as composed of users and "raw content", representing photos, videos, or text that a person has in his personal page. Four relations concerning raw contents are modelled at this level: content, visibility, ownership, and access privileges. The "content" relation associates a person with all raw contents currently in the person's page.

Table 2. System architecture

Machine	Observations
Abstraction	Page content, content visibility, content ownership, access privileges
Refinement 1	Principal content, page fields
Refinement 2	Mandatory content
Refinement 3	User wall, wall visible content, wall access privileges
Social_friends	Friendship relations
Refinement 4	Relations between friendship, visibility and privileges

Each user owns some of the content in his page. The "visible" relation associates a person with visible raw content. Visible raw contents are those raw contents a user is allowed to view at some point. Only those raw contents for which a user has "view" privilege can be visible. The "content" relation contains the "visible" relation. The "view" privilege and other types of privileges (e.g., edition of a particular content) are defined in the access privileges relation *act*. Elements in *act* are triplets (rc, op, pe) stating that person *pe* has *op* privilege on raw content *rc*. In B language notation a triplet (a, b, c) is written $a \mapsto b \mapsto c$.

The owner $owner(rc)$ of a raw content rc is unique. The following invariant properties of the abstract model state that, (1) $owner(rc)$ has all privileges over rc[2], (2) each raw content owned by a user is in the user's page content, (3) a raw content is visible for a user only when the user has "view" privilege over it, and (4) all user's visible raw contents are in the user's page.

(1) $\forall rc.(rc \in rawcontent \Rightarrow (\forall op.op \in OPS \Rightarrow (\ rc \mapsto op \mapsto owner(rc)\ ) \in act))$
(2) $owner^{-1} \subseteq content$
(3) $\forall (rc, pe).(rc \in rawcontent \wedge pe : person \Rightarrow$
$((pe \mapsto rc) \in visible \Rightarrow (rc \mapsto view \mapsto pe) \in act))$
(4) $visible \subseteq content$

The abstract model defines actions (so-called "operations") for creating, transmitting, making visible, hiding, editing, commenting and removing a raw content. All these, of course, are defined so as to maintain all the invariant properties. Code for the operation representing a user removing from his page a raw content owned by some other user is shown in Table 3. The pre-conditions requires the user in question not being the owner. Upper case items refer to types. Lower case, to variables of the system. A user can only remove visible raw contents. The SELECT clause has two cases. The first one is when the *rc* to be eliminated is not the only one present in *pe*'s page. The second one is the opposite. Since the web page of each person in the system must have at least one content, *pe* must be deleted from the system in this case. In B notation, $C \lhd r$ and $r \rhd C$ denote restriction of a relation r to a subset C of its domain and its range respectively. Similarly, $C \lhd\!\!\!- r$ and $r \rhd\!\!\!- C$ denote the restriction of the domain and the range of r to elements *not* belonging to C.

[2] OPS is the set of privilege types in the system.

Table 3. Operation for removing a raw content

remove_rc (rc , pe) =
PRE
$rc \in RAWCONTENT \wedge rc \in rawcontent \wedge$
$pe \in person \wedge$
$pe \mapsto rc \in visible \wedge pe \neq owner(rc)$
THEN
SELECT $pe \in dom(content - \{pe \mapsto rc\})$ THEN
$visible := visible - \{pe \mapsto rc\} \parallel$
$content := content - \{pe \mapsto rc\} \parallel$
$act := act - \{rc \mapsto view \mapsto pe\} \parallel$
WHEN $pe \notin dom(content - \{pe \mapsto rc\})$ THEN
$visible := \{pe\} \lhd visible \parallel$
$content := \{pe\} \lhd content \parallel$
$act := act \rhd \{pe\} \parallel$
$person := person - \{pe\}$
END
END

Table 4. Operation for removing an owned raw content

remove_owned_rc (rc) =
PRE
$rc \in RAWCONTENT \wedge rc \in rawcontent$
THEN
$visible := visible \rhd \{rc\} \parallel$
$content := content \rhd \{rc\} \parallel$
$act := (\{rc\} \times OPS) \lhd act \rhd dom(content \rhd \{rc\}) \parallel$
$owner := \{rc\} \lhd owner \parallel$
$person := dom(content \rhd \{rc\})$
END
END;

The operation for a user removing an owned raw content is shown in Table 4. Notice that in this case the removed content must also be removed from all other user's pages ($content \rhd \{rc\}$). This might leave some users with no raw contents in their pages. The persons remaining in the system must thus be recomputed ($person := dom(content \rhd \{rc\})$).

The first refinement mainly adds the observation of page fields. Each content belongs to some field. The notion of field models the fact that users perform different actions, such as commenting, dealing with some given content. Page fields are defined as $field \in (rawcontent - principal) \rightarrow principal$. The various raw contents of a given field are thus thought to be related (e.g. as a comment, or as being part of a photo album) to a unique *principal* raw content. Removing a principal raw content entails removing all its "comment" contents in all user

pages. A principal raw content can only be removed by its owner. The following selected actions from the *remove_owned_rc* operation show this behaviour (for the case $rc \in principal$). Expression $field^{-1}[\{rc\}]$ gives all secondary raw contents whose primary is rc. These have to be removed together with rc.

$$rawcontent := rawcontent - (field^{-1}[\{rc\}] \cup \{rc\}) \parallel$$
$$content := content \rhd (field^{-1}[\{rc\}] \cup \{rc\}) \parallel$$
$$act := (\{rc\} \times OPS) \lhd act \rhd dom(content \rhd \{rc\})$$

The second refinement models the fact that each user page must always keep some predefined minimum information. This is represented as a set of special predefined contents (referred to as *prawcontent*) in each page that cannot be removed. This predefined information must be present in a page before any other content is added, as stated in the invariant property, $prawcontent \subset rawcontent \Rightarrow prawcontent \neq \varnothing$. Remove operations are refined to ensure that all these special raw contents are always kept in a user's page.

The third refinement models the notion of *wall*, common in social network systems. A wall is modelled as a relation associating a user with some raw contents *different* from those in her web page: $wall \in person \leftrightarrow rawwall \wedge (rawwall \cap rawcontent = \varnothing)$. Each wall owner gives others some particular visibility and access privileges to his wall. Operations for adding/removing/hiding comments to/from the wall are included at this level.

Machine *Social_friends* provides definitions for types of friendship relations in social networks. The machine models *acquaintance, social* and *best friend* relations, with operations to add/remove users to/from each type of friendship relation of a given user. This machine is parametrised with a set modelling a type (that of "friends"). Some invariant properties of this machine are shown below, where $id(friend)$ is the identity relation over $friend$, and $ran(friendship)$ is the range of the $friendship$ relation. The third property states that a user is not a friend of himself. The fourth one states that all friends are involved in some friendship relation. Other friendship types are defined similarly. Notice that friendship relations are *not* defined to be transitive.

$$friend \subseteq FRIEND$$
$$friendship \in friend \leftrightarrow friend$$
$$id(friend) \cap friendship = \varnothing$$
$$friend = dom(friendship) \cup ran(friendship)$$
$$best\_friends \in friend \leftrightarrow friend$$
$$best\_friends \subseteq friendship$$

The fourth refinement includes the *Social_friends* machine. It models how access privileges relate to friendship relations, namely, $best\_friends$, $social\_friends$, and $acquaintances$. The relation $best\_friends$ models the highest level of friendship of people in the social network, and $acquaintances$ the lowest. In general, a

lower friendship level cannot have any access privilege a higher level does not also have, as stated in the following invariant properties:

$$\begin{array}{|l|}\hline \forall pe.pe \in dom(friendship) \Rightarrow \\ \quad \forall bs.bs \in best\_friends[\{pe\}] \Rightarrow \\ \qquad (owner^{-1}[\{pe\}] \times OPS) \cap act^{-1}[social\_friends[\{pe\}]] \\ \qquad\quad \subseteq \\ \qquad (owner^{-1}[\{pe\}] \times OPS) \cap act^{-1}[\{bs\}] \\ \hline \forall pe.pe \in dom(friendship) \Rightarrow \\ \quad \forall sf.sf \in social\_friends[\{pe\}] \Rightarrow \\ \qquad (owner^{-1}[\{pe\}] \times OPS) \cap act^{-1}[acquaintances[\{pe\}]] \\ \qquad\quad \subseteq \\ \qquad (owner^{-1}[\{pe\}] \times OPS) \cap act^{-1}[\{sf\}] \\ \hline \end{array}$$

Similar properties are stated for wall access privileges. All these properties only relate to each user's raw contents. That is, for any rc of a given user pe (i.e. $owner^{-1}[\{pe\}]$), the privileges of her social friends with respect to rc cannot include something that any pe's best friend does not also have. In this fourth refinement, the *remove_owned_rc* operation adds the action

$$restrict\_friends(dom(content \rhd \{rc\}))$$

where *restrict_friends* is an operation of the *Social_friends* machine restricting the friendship relation to the supplied set (see Table 5).

Table 5. Restricting friendship relations to a supplied set

$$\begin{array}{|l|}\hline \text{restrict\_friends(frs)} = \\ \quad \text{PRE } frs \subseteq FRIEND \\ \quad \text{THEN} \\ \qquad friendship := frs \lhd friendship \rhd frs \parallel \\ \qquad best\_friends := frs \lhd best\_friends \rhd frs \parallel \\ \qquad social\_friends := frs \lhd social\_friends \rhd frs \parallel \\ \qquad acquaintances := frs \lhd acquaintances \rhd frs \parallel \\ \qquad friend := friend \cap frs \\ \text{END} \\ \hline \end{array}$$

The operations distinguish between commenting a particular raw content in some user's page or doing so in the wall. Commenting a wall is done as shown in table 6. Variable *wall* records all contents present in each person's wall. Variable $vinwall \subseteq wall$ keeps track of visible wall contents for each person, *wallowner* the owner of each content in a wall and *wallaccess* defines, for each wall owner, the persons allowed to comment her wall. In the operation in table 6, when a comment is added to the wall of person *ow*, the added comment is put in the wall of each person having access to the wall of *ow* (expression $(wallaccess[\{ow\}] \times \{cmt\})$) and is also defined to be visible in those walls.

Table 6. Operation for commenting in a wall

comment_wall (cmt, ow, pe) =
PRE
$pe \in person \wedge ow \in person$
$\wedge cmt \in RAWCONTENT \wedge cmt \notin rawcontent$
THEN
SELECT $ow \mapsto pe \in wallaccess \wedge cmt \notin rawcanvas$
THEN
$rawwall := rawwall \cup \{cmt\} \parallel$
$rawcanvas := rawcanvas \cup \{cmt\} \parallel$
$vinwall := vinwall \cup (wallaccess[\{ow\}] \times \{cmt\}) \parallel$
$wall := wall \cup (wallaccess[\{ow\}] \times \{cmt\}) \parallel$
$canvas := canvas \cup (wallaccess[ow] \times \{cmt\}) \parallel$
$wallowner := wallowner \cup \{cmt \mapsto ow\}$
END
END

3.1 Publishing Content

A common operation to social-networking web-sites is publishing content to people in the network. Publishing a social-network content rc to a user pe can be regarded as a process of transmitting rc from the page of $owner(rc)$ to the page of pe. The abstract machine code for transmitting a raw content in a social-network is shown in Table 7. The pre-condition of $transmit\_rc$ requires that pe is different than ow, and rc is not already in the page of pe. To transmit raw content rc, the triplet $rc \mapsto view \mapsto pe$ is added to act so as to grant the $view$ permission on raw content rc to user pe, and raw content rc is made visible to pe by adding $pe \mapsto rc$ to $visible$.

In a complementary direction, user pe can request permission to operate raw-content rc. The abstract machine code for requesting a particular permission on a raw content rc is shown in Table 8, where op is the permission being requested.

Table 7. Transmitting page content

transmit_rc (rc , ow , pe) =
PRE
$rc \in RAWCONTENT \wedge rc \in rawcontent \wedge ow \in person \wedge$
$pe \in person \wedge ow = owner(rc) \wedge$
$ow \neq pe \wedge pe \mapsto rc \notin content \wedge rc \mapsto view \mapsto pe \notin act$
THEN
$visible := visible \cup pe \mapsto rc \parallel$
$content := content \cup pe \mapsto rc \parallel$
$act := act \cup rc \mapsto view \mapsto pe$
END

Operation *request_permission* can either grant *pe* permission *op* over raw content *rc*, or deny the permission. If the permission is granted, $rc \mapsto op \mapsto pe$ is added to *act*, *rc* is added to $content(pe)$, and the result variable *res* is set to $TRUE$ so as to communicate the success of the operation. Otherwise, when the permission is denied, *res* is set to $FALSE$. The pre-condition of requesting a permission requires that *pe* is different than $owner(rc)$.

Table 8. Requesting Content Permissions

res ⟵ request_permission (rc , pe , op) =
PRE
$rc \in RAWCONTENT \wedge rc \in rawcontent \wedge pe \in person \wedge$
$op \in OPS \wedge pe \neq owner(rc)$
THEN
CHOICE
$act := act \cup rc \mapsto op \mapsto pe \parallel$
$content := content \cup pe \mapsto rc \parallel$
$res := TRUE$
OR
$res := FALSE$
END
END

4 Related Work

P3P, the Platform for Privacy Preferences (`http://www.w3.org/P3P/`), an effort of the World Wide Web Consortium (W3C), encompasses a standard XML markup language for expressing privacy policies so as to enable user agent tools (e.g. Web browsers, electronic wallets, mobile phones, stand-alone applications, or social network applications) to read them and take appropriate actions. A P3P Policy is primary a set of boolean answers to multiple-choice questions about name and contact information, the kind of access that is provided, the kind of data collected, the way the collected data will be used, and whether the data will be shared with third parties or not. Though P3P policies are precisely scoped [13], they are not expressive enough to model general privacy properties on content. They are not based on mathematical formalisms either, e.g., predicate calculus, so that it is not possible to reason about the truths derivable from policies expressed in P3P standard language.

In [7], N. Sadeh et al. develop a theory that relates expressiveness and efficiency in a domain-independent manner. Authors derive an upper bound on the expected efficiency of a given mechanism. The expected efficiency depends on the mechanism's expressiveness only. Using predicate calculus to write users' privacy policies on content improves the expressiveness of mechanisms modelling policies. We plan to build on Sadeh et al.'s work to study how this higher expressiveness of predicate calculus based privacy policies comes down to a higher

efficiency of the agent mechanisms allowing social-network users to set their privacy preferences.

In B language the expression of temporal logic constraints is notably missing. In [15], J. Groslambert proposes a method to verify temporal logical properties of Event B systems. We will build on Groslambert's work, and J-R Abrial's work in [2], to verify temporal logic properties about Matelas.

In [19], Vijay Saraswat et al. propose a policy language for access control, and a policy algebra in the timed constraint programming paradigm. Based on Saraswat's work, we plan to extend our work on modelling privacy on content with a relation that registers users' access privileges on social-network content with a relation that registers role-based access permissions on content.

5 Conclusion

We presented Matelas, a B model for social networking, describing social-network content, privacy policies, social-networks friendship relations, and how these effect the policies with regards to content and other users in the network. We used Atelier B [5] to write Matelas. We found the B method particularly useful in two aspects. One is the expressivity of the generalised substitution language that makes it possible to construct a very simple abstract model of the system, yet containing all fundamental security and privacy properties. The second is that proof obligations are easy to interpret as "before-after" predicates of each operation assignments which makes it easy to discover possible errors and their correction by just looking at the statement of the proofs. A minor drawback, at least for this application, is that some useful operations are discovered as the refinement process leads to more detailed components which requires to change all previous models to include these operations as empty statements. This inconvenience could, of course, be circumvented by using Event B. Our decision of using the Atelier B tool to undertake the development of the social-network core was based on the authors' previous experience with the tool. We envision to port Matelas machines to Event B models and to use Rodin [24] to refine Matelas so as to produced the social-network core system.

We have a positive impression on the use of Atelier B as tool to develop relatively complex software systems, yet have some recommendations on how the tool can be improved. For Matelas' abstraction, the four refinements and the *social_friends* machine, the B method software tool Atelier B generates 658 proof obligations. About 60% of them are discharged automatically. Most of the others (handling up to 90% of all obligations) are discharged in *Atelier B* by just doing *modus ponens* followed by invocation of one of the available provers. Some proofs, especially those involving equality of assignments in abstract and concrete machines, are somewhat tricky. We found this might be due to some limitations of the provers for handling predicates of the form $A \vee B$, for A, B complex expressions with A false and B true, even when B is supplied as a hypothesis and proved first or, similarly, $\neg A$ is added as hypothesis and proved first.

The work presented in this paper is part of an ongoing work that aims at developing a full-fledged social network core that implements stipulated privacy

policies. To the best of our knowledge, this is the first effort on using the B method to formally develop a social-network web-site. We plan to refine Matelas to a social-network core, and use Proof Carrying Code [20,21] to build Java plug-ins that extend its features. The policies for Java plug-ins can be written in JML, which allows the use of different formal methods tools to check program correctness [9,11]. JML specifications have the advantage over predicate calculus based models in that they are close to Java, and thus are closer to average programmers. We envisage to investigate on systematic ways B Machines can be translated into JML specifications. Work has already been done in the other direction [8], that is, to transform JML specifications into B machines to check the specifications for flaws. Alternatively, plug-ins can be written in C language, and formal specifications using the ACSL (ANSI/ISO C Specification Language) specification language [6], a JML-like specification language for C programs. Altogether, our work falls within the scope of the Grand Challenge in "Dependable System Evolution" (`http://vsr.sourceforge.net/introduction.htm`) set forth by the U.K, and Tony Hoare's Grand Challenge in Verified Software. The challenge is to create a toolset that would guarantee that programs meet given specifications.

References

1. Abrial, J.R.: The B-Book: Assigning Programs to Meanings. Cambridge University Press, Cambridge (1996)
2. Abrial, J.-R., Mussat, L.: Introducing dynamic constraints in B. In: Bert, D. (ed.) B 1998. LNCS, vol. 1393, pp. 83–128. Springer, Heidelberg (1998)
3. Abrial, J.R., Hallerstede, S.: Refinement, decomposition and instantiation of discrete models: Application to Event-B. Fundamentae Informatica 77(1,2), 1–24 (2007)
4. Allen, A.L.: Uneasy Access: Privacy for Women in a Free Society. Rowman and Littlefield (1988)
5. Atelier b, `http://www.atelierb.eu/index_en.html`
6. Baudin, P., Filliâtre, J.-C., Marché, C., Monate, B., Moy, Y., Prevosto, V.: ACSL: ANSI/ISO C specification language, `http://frama-c.cea.fr/download/-plug-indevelopmentguide.pdf`
7. Benisch, M., Sadeh, N., Sandholm, T.: A theory of expressiveness in mechanisms. In: Proceeding of the 23rd Conference on Artificial Intelligence (July 2008)
8. Bouquet, F., Dadeau, F., Julien, J.: JML2B: Checking JML specifications with B machines. In: The 7th International B Conference, pp. 285–288 (2007)
9. Breunesse, C., Catano, N., Huisman, M., Jacobs, B.: Formal methods for smart cards: An experience report. Science of Computer Programming 55(1-3), 53–80 (2005)
10. B Tools, `http://www.b-core.com/btool.html`
11. Burdy, L., Cheon, Y., Cok, D., Ernst, M., Kiniry, J., Leavens, G.T., Leino, K.R.M., Poll, E.: An overview of JML tools and applications. International Journal on Software Tools for Technology Transfer (STTT) 7(3), 212–232 (2005)
12. Catano, N., Kostakos, V., Oakley, I.: Poporo: A formal framework for social networking. In: 3rd International Workshop on Formal Methods for Interactive Systems (FMIS), Eindhoven, The Netherlands (November 2009) (to appear)

13. Cranor, L., Lessig, L.: Web Privacy with P3p. O'Reilly & Associates, Inc., Sebastopol (2002)
14. Facebook's statistics, http://www.facebook.com/press/info.php?statistics
15. Groslambert, J.: Verification of LTL on B event systems. In: Julliand, J., Kouchnarenko, O. (eds.) B 2007. LNCS, vol. 4355, pp. 109–124. Springer, Heidelberg (2006)
16. Gross, R., Acquisti, A.: Information revelation and privacy in online social networks. In: Workshop on Privacy in the Electronic Society (WPES), pp. 71–80 (2005)
17. He, J., Hoare, C.A.R., Sanders, J.W.: Data refinement refined. In: Robinet, B., Wilhelm, R. (eds.) ESOP 1986. LNCS, vol. 213, pp. 187–196. Springer, Heidelberg (1986)
18. Hoare, C.A.R.: Proof of correctness of data representations. Acta Informatica 1, 271–281 (1972)
19. Jagadeesan, R., Marrero, W., Pitcher, C., Saraswat, V.A.: Timed constraint programming: a declarative approach to usage control. In: Proceeding of Principles and Practice of Declarative Programming (PPDP), pp. 164–175 (2005)
20. Necula, G.C.: Proof-carrying code. In: Symposium on Principles of Programming Languages (POPL), Paris, January 1997, p. 106119 (1997)
21. Necula, G., Lee, P.: Research on proof-carrying code for untrusted-code security. In: Proceedings of the 1997 IEEE Symposium on Security and Privacy, p. 204 (1997)
22. ProB, http://users.ecs.soton.ac.uk/mal/systems/prob.html
23. Robinson, A., Voronkov, A.: Handbook of Automated Reasoning. MIT Press, Cambridge (2001)
24. Rodin, http://www.event-b.org/platform.html
25. Schneider, S.: The B-Method: An Introduction. Palgrave (2001)
26. Schoeman, F.D.: Philosophical Dimensions of Privacy: An Anthology. Cambridge University Press, Cambridge (1984)