Approaches to Modelling Security Scenarios with Domain-Specific Languages

Phillip J. Brooke[1], Richard F. Paige[2], and Christopher Power[2]

[1] School of Computing, Teesside University, Middlesbrough, TS1 3BA, UK
pjb@scm.tees.ac.uk
[2] Department of Computer Science, University of York, York, YO10 5GH, UK
{richard.paige,christopher.power}@york.ac.uk

Abstract. Many security scenarios involve both network and cryptographic protocols and the interactions of different human participants in a real-world environment. Modelling these scenarios is complex, in part due to the imprecision and underspecification of the tasks and properties involved. We present work-in-progress on a domain-specific modelling approach for such scenarios; the approach is intended to support coarse-grained state exploration, and incorporates a classification of elements complementary to computer protocols, such as the creation, personalisation, modification and transport of identity tokens. We propose the construction of a domain-specific language for capturing these elements, which will in turn support domain-specific analyses related to the reliability and modifiability of said scenarios.

1 Introduction

We present a work-in-progress report on modelling security scenarios beyond the level of the network and cryptographic protocols to the level of human interactions with the security components. This work is an application of a coarse-grained state exploration approach that incorporates a classification of elements that are complementary computer protocols, such as the creation, personalisation, modification and transport of identity tokens. The present conceptual work for these attributes is currently being implemented as a domain-specific language (DSL) in a modelling and simulation framework.

Current research has looked at security scenarios in detail regarding the computing aspects; however, there has been low focus on the types of attacks that can happen in a physical environment involving human users. The following scenario demonstrates one such case:

> *Alice applies for, and is issued, an identification card from her local municipality. The card is couriered to Alice's home address. This card will be used by Alice in various businesses and services in the community. With many of the transactions, such as applying for a tax certificate for her car, a clerk, Bob, will take Alice's card, and check it for various pieces of information. Bob the clerk at the tax office uses card as a means of verifying Alice's claims of things about herself (e.g., name, age).*

In this scenario, it is insufficient to consider only attacks on the cryptographic and networking components. For instance, there are potential problems with physical consistency throughout the system. If Alice has an identification card, it must have been created and sent to her. There are points along that value chain where attacks of opportunity can happen. The person issuing the card could make an error about Alice's age, or a courier taking the card to Alice could lose it. It is also possible that the courier could actively try to take and use Alice's card to obtain a service to which he is not entitled.

Within such scenarios, *beliefs* also play a role; this is because in some of the activities within the scenario, two or more actors are involved, and each may have a different set of behaviours *in that context*. For example, while Bob might believe the information on the identification card when Alice uses it, he may be less likely to believe it when the courier is using the card.

All of these situations lead to a complex set of properties that must be represented in any system hoping to model security risks that may arise from execution of a scenario.

This line of work was originally motivated by the UK's Identity Cards Act 2006 (now repealed). The type of question we were interested in trying to answer was: Given a scenario, what difference do particular configurations of that scenario make to the outcome? More concretely, in our example above, does the existence and use of an identification card make enough of a difference to the identification decisions the clerk, Bob, has to make to justify its additional cost?

To help to answer such questions, we would benefit from an infrastructure that can describe and test assertions about identification scenarios. This paper contributes our work-in-progress to modelling the elements of scenarios beyond the network and cryptographic protocols.

2 Modelling security scenarios

We outline the overall approach taken to modelling scenarios like the identification example presented earlier. In general, these are scenarios with (informally) some element of *fuzziness* to them — specifically, scenarios which can be configured and where the decisions within are taken based on probability distributions. We present a high-level overview of the model, starting with its requirements and then presenting its key entities.

2.1 Requirements for the modelling approach

The requirements for our modelling approach are as follows. They are expressed assuming that we wish to exploit, as much as possible, model checking approaches, so as to use proven technology. We have three simulation or exploration requirements concerning automated exploration, probabilistic evaluation and simulation of individual paths through the scenario, as well as a need for a DSL.

Automated exploration We want to analyse every possible path through the scenario. This is akin to model checking (using tools such as FDR2 [8] or SPIN [25]), and is necessary for completeness.

Probabilistic evaluation When exploring every path, we want to assess how likely it is to occur. It is very unlikely that an agent will sell alcohol (under jurisdictional restrictions) to a 5 year old, yet someone who appears to be over 21 is unlikely to be challenged. This makes the scenarios fuzzy, requiring parameterisation. Such parameters might include

- a probability density function of the age of a subject;
- a probability density function giving an agent's perception of the subject's age, given her actual age.

Individual path simulation For validation and understanding of the model, we would like to be able to step through particular paths. We anticipate increasingly complex scenarios, and individual path simulations are a useful counterpoint to the complete automated exploration.

Domain-specific We could build a simulation in any computer language. To make it understandable and usable, we require something specific to the problem domain. It has to understand that there are actors and objects, that actors can show objects (*e.g.*, identity cards) to other actors, *etc.*

Additionally, the set up of a scenario is a distinct and complicated step. Concretely, does the subject remember to take her identity card with her? How were they issued? We could take this a step further and simulate the issue and delivery of such credentials. This bootstrap has to be included in the scenario, as it gives opportunities for people to make mistakes, others to intercept credentials, and so on.

One approach to better enable the construction and validation of scenarios is to provide a domain-specific language, specifically defined and constrained to support exactly and only the concepts and logic for describing scenarios. If we are able to define such a DSL, capturing *just* the concepts of scenarios (and their semantics), we ideally make it easier for end-users (who may be security experts) to specify relevant and meaningful scenarios of interest, and provide greater confidence that said scenarios are acceptable and valid. Moreover, if the DSL is specified and implemented in an appropriate way (*e.g.*, using Eclipse EMF or other similar modelling technologies), automated tools can be developed, such as editors, that end-users can apply in constructing and checking their scenarios. Such *convenience facilities*, while not essential, can add further confidence that well-formed scenarios have been constructed.

2.2 Concepts and definitions

The model we present has been adapted from scenario based design proposed by Carroll and Rosson [24] for the design of interactive systems. Essentially, we are simulating actors (the subject, the agent) interacting with different components of the system, and the events that occur in response to those interactions. These concepts were identified by sketching use cases of a series of scenarios.

People and actors can be classified within a scenario thus

Subject An individual that initiates or triggers a scenario; often a user or customer (*e.g.*, a person being identified). In police or immigration-related scenarios, the subject is a person being identified.

Imposter A person masquerading as a subject (whether the subject actually exists or not).

Cheat A subject who is attempting to convince an agent that they hold a property that in fact they do not. For example, a cheat may try to convince an agent that they are of a particular age when they are not. We distinguish a cheat from an Imposter, who attempts to impersonate a different subject.

Agent A representative (*e.g.*, sales assistant, police officer) of an organisation that interacts with the subject.

Actor A generic term covering all the roles above, as well as others not explicitly noted. For example, in an identification scenario, a particular subject could be using another actor's identification tokens.

We need to model a *memory* for all actors. This represents their prior beliefs, current impressions, their understanding of an encounter so far, and so on. In section 4, we fold these into properties, *i.e.*, the memory of an actor is a property of the actor, simply to reduce the amount of terminology and different fields in the formal definitions. A further possibility is to consider *forgetfulness* as another source of errors or protocol problems.

Agents may act on behalf of an organisation:

Organisations. An organisation owns artefacts of values, and has representatives (agents) that engage in scenarios. Organisations wish to ascertain whether subjects hold properties related to artefacts of value. Some of these organisations will issue identity cards; others may courier or deliver items, cards, *etc.*

Some scenarios involve identity cards, *e.g.*, national identity cards, passports, driving licences, PASScards, which involve the following entity types:

Genuine card A genuine (non-forged) identity card: it may be held by the subject, or by an imposter. Additionally, it may be a card that an imposter has persuaded the issuer to wrongly issue. Genuine cards may be faulty in some sense, *e.g.*, smartcard chips may have been damaged (deliberately/maliciously or accidentally) and not operate.

Forged card An identity card that has not been issued by the issuer.

Tampered card A genuine card that has been tampered with, *e.g.*, to modify the photograph. (Both forging and tampering with credentials is well-known in practice.)

Card terminal A card reader, that incorporates a slot or interface for the genuine or forged card, a key pad for entering PINs and a display to direct the user. Some terminals may be able to take biometric readings.

We could further address the issue of corrupted terminals in this work — such terminals could skim the card, attempt to retain other data such as the subject's PIN or biometric reading, or give misleading information on a

display. There are other problems that can be investigated here, many similar to ATMs, such as whether subjects have good cause to trust terminals. How can they know the terminal is not corrupt? What stops someone 'shoulder-surfing' to obtain a PIN?

Objects A generic term covering all the items above, as well as any others required in a particular scenario. Other services (the two examples immediately below) are encoded into the objects that provide access to those services.

Verification and CRL services Cards can be verified via terminals. These may be on- or off-line, and thus network and cryptographic protocols are required. Some services could perhaps be used for the identification of subjects who are not carrying credentials.

Communication links Terminals must be able to communicate with both an organisation's computers and verification services. In the same way that we need to model the couriering of cards from issuer to subject, communication between computers is mediated by these links.

Within a scenario, actors (*e.g.*, subjects, agents) interact, sometimes using particular artefacts (*e.g.*, items of value such as ID cards). Some interactions are constrained by location: for example, a subject cannot show an identity card to an agent if they are not both in the same location. As such, the model has to support constraints in scenarios.

The sketch use cases that produced the concepts above also produced possible *operations* applicable to concepts and configurations. Operations are interpreted as events resulting in state changes. Specifically, the operations identified are: **pick up** and **drop** an object from or into a location; **give** an object to another actor (presumably in the same location, *e.g.*, a courier); **say** to another actor (*e.g.*, to trigger some transaction); **show** an object to another actor; **set** a property or belief; and, **move** between locations. Many of these are relatively simple in terms of state change and modelling. **say** and **set** are substantially more complicated, as they can be used as "escape hatches" to implement relatively complicated interactions.

There is one further class of entity that we need to describe: attackers.

2.3 Attackers

We have used Casper [15] as an inspiration for some of this work; so we require an analogue of the most general intruder. In one sense, this is partially covered in the Actors described above (*e.g.*, Imposter and Cheat).

A more specialised treatment may allow our analysis to produce emergent, unanticipated results. So an attacker may be involved in

- the interception of credentials in transit, similar to the missing-in-post problem for delivery of credit cards;
- theft and robbery: simply taking items from a location or another actor;
- interacting with other attackers. There may be multiple actors behaving as attackers. This interaction could involve cooperation or even competition

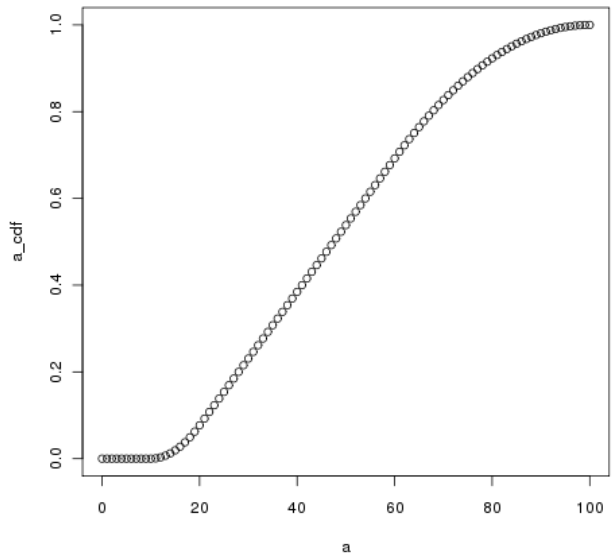

Fig. 1. Example cumulative density function for an off-licence subject's age.

between attackers. Indeed, some systems may not trust their expected subjects (such as ecash services, where the system has to prevent the subjects adding extra value without payment).

3 Fuzziness and probabilities

Nondeterminism is inherent in this work. When an actor sees another actor, they both form impressions about the other. In scenarios where ages are important, an actor will assess another's age as being approximately the correct age, but we assert that this is likely to be a normal distribution. When close to a legal boundary, this can be important. As an example, we might use a simple trapezoid to describe the probability distribution of the *actual* age of subjects attending an off-licence, as illustrated in figure 1. The agent who has to determine the subject's age will have a probability density function giving her *estimate* of the subject's age given the actual age, for example, figure 2. This particular example illustrates the need for different configurations of a scenarios —the likely distributions may vary according to the particular deployment— and the need for validation.

Similarly, when confronted with a photographic credential and an actor asserting to be the person in the photograph, the likelihood of the verifier accepting that identification depends on the quality of the credential, the difference in time, changes in appearance, the diligence of the verifier and the skill of the verifier.

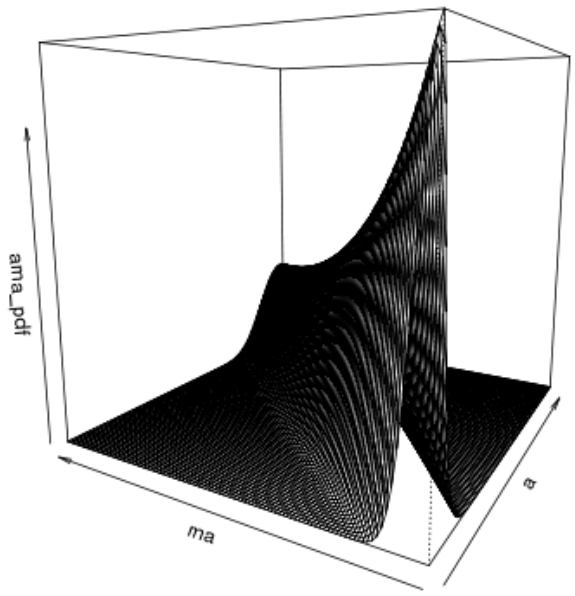

Fig. 2. Example probability density function for an off-licence scenario. a denotes the subject's actual age, ma is the perceived age and ama_pdf is the probability.

Alternatively, an automated device may be involved, such as in the current generation of some eGates at national borders and services such as `face.com`. Such matters can be modelled as nondeterminism.

A final source of nondeterminism in our scenarios is *mistakes*. Humans make errors. Some may simply not care or be less diligent than other actors. But we cannot guarantee that the actors in our scenarios will follow protocols faithfully; indeed, the point of our work is that actors cannot always follow protocols perfectly.

4 Formal definitions

Given the concepts, configurations, notions of constraints, and operations, a formalisation of the model (in, *e.g.*, Z) is straightforward. We now give a more formal description of scenario, state and event.

A *scenario* is a tuple $(A, B, O, L, P, E, s, e_e, e_a)$ where

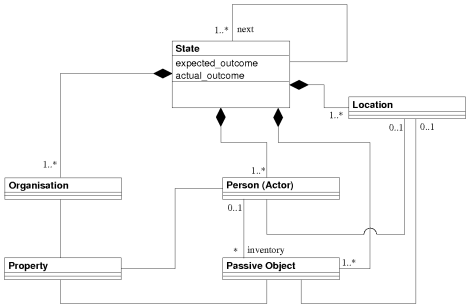

Fig. 3. States model.

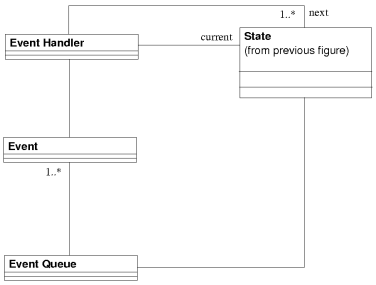

Fig. 4. Events model.

- A is a set of actors;
- B is a set of (passive) objects;
- O is a set of organisations;
- L is a set of locations;
- P is a set of properties (which include, for example, if an identity document has been modified and the beliefs that an actor may form about another actor or their memories);
- E is a set of possible events;
- s is a sequence of events that set up the scenario; and
- e_e and e_a are functions over a state of the simulation evaluating the expected and actual outcomes.

We illustrate this in figures 3 and 4.

A *state* is a tuple (l, i, r, q) where

- $l : A \nrightarrow L$ gives the locations of the actors;
- $i : B \nrightarrow L \cup A$ gives the location containing or actor holding each object;
- $r : A \cup B \cup O \cup L \nrightarrow P$ gives the properties associated with the actors, objects, organisations and locations; and
- q is a priority queue of pending events. They are prioritised as
  immediate contains events that should happen as soon as possible, such as an actor's reaction to another actor's action;
  delayed contains events with a natural delay, such as movement, or posting an item. These happen after all immediate events; and
  deferred contains events relating to the simulation management.

l, i and r are partial functions, which allows for an actor or object to not be present at all and for only desired properties to be selected.

Finally, an *event* is a conventional transition on a state (l, i, r, q) resulting in one or more new states (l', i', r', q'). The operational view of an event is that the head of q is processed according to its definition. The processing of an event may add additional events to the queue; some of these additional events are implicit in the design, for example, the arrival of an actor might cause a second actor to carry out some action, which itself is represented by enqueuing further events.

Where more than one state results, the *transition* is associated with either a fixed probability (*e.g.*, 0.6) or an abstraction of a random variable to handle nondeterminism; again this is defined by the event.

Given a scenario $(A, B, O, L, P, E, s, e_e, e_a)$, we create an initial state $(\emptyset, \emptyset, \emptyset, s)$. Configuration may slightly adjust initial events or probability distributions (*e.g.*, in our running example, identity cards may not be available at all). From the initial state, we select the first event and this results in a subsequent state or subsequent states: thus our model may end up with multiple current states if we are attempting an exhaustive exploration instead of concentrating on only one successor at a time. We repeat this for each subsequent state until the queue of pending events is empty in each current state, at which point we evaluate e_e and e_a to determine the outcome of each terminal state.

One particular issue is that the rich state, in part resulting from modelling actors' memories, turns what might have been a fairly large digraph into a tree for all but the most trivial scenarios. Thus traditional model checking approaches are problematic.

5 Domain-Specific Language

We aim to refine and implement this model using a domain-specific language; this will allow us to capture scenarios (involving actors, etc) and will support end-users —who may be identification specialists, security experts, software engineers or business specialists— in expressing scenarios and validating them. Building a suitable DSL is therefore a way of *implementing* our scenario-based approach. Our rationale for constructing a DSL as opposed to simply implementing the model in, *e.g.*, an object-oriented programming language, is that a DSL-based approach allows us to capture only essential concepts, thus making it easier to validate the scenarios (and underlying model) that we produce.

As is the case with the design of any language, we want our scenario DSL to be sufficiently expressive while practical: it should allow end-users to capture the full scope of a scenario without the underlying model being too large to meaningfully analyse.

The DSL has to capture the actors, objects, etc. It must also deal with constraints, such as an actor can only **give** an object to another actor if they are both in the same location; and responses, such as an actor responding to another **say**ing something. Some of these should be implicit to the simulation and are generic (*e.g.*, **give**) whereas others are likely to be specific to each scenario.

We could consider the separate phases of a scenario. For example, we might view bootstrap and an encounter of a subject and agent as being two phases. Complex scenarios may have more than two phases. Splitting them into distinct phases allows more reusability. Similarly, the configuration must be captured, for example, is a particular type of identify credential available in this run?

We need to describe the outcomes we are testing. At its simplest, the simulation knows what the correct outcome is: it knows whether or not the subject should be able to carry out whatever transaction they are requesting. Thus we

can simple enumerate possible outcomes and compare true/false positive/negative outcomes for different configurations. However, there are further characteristics we might wish to check: a significant one relates to privacy and information leakage. Does a particular configuration make it easier for one actor to obtain (unnecessary) information about another?

Some form of overloading is desirable: some behaviours of actors will be generic, yet the actors will also need scenario-specific and even configuration-specific behaviours and responses.

Lastly, we must describe the attacker(s) behaviour. Can someone misbehave in this scenario? What emergent behaviours might we see?

6 Related work

Much of the motivation for this work concerns simulation or model checking of scenarios involving humans and identity, where identity is a property of entities that allows entities to be distinguished from each other. Sometimes humans are attempting to identify another individual, or are instead carrying out the task of verification. Sometimes a computer or network is involved, perhaps as an intermediary between two humans, or in place of one of the parties.

Precise definitions of 'identity' can be controversial [3]. In this case, it is recognising an individual which degenerates (in our examples) to unique naming. Identification is the process of establishing an entity's identity via a particular mechanism (or set of mechanisms), *e.g.*, comparison to a trusted repository of identities. Of course, different actors will have different assessments of the trust of any particular repository.

Current identification practices are varied, and can involve a vendor making a judgement as to age, or accepting other documents that may never have been intended for this purpose. The notorious example here is the use of utility bills for identification. Government-issue ID such as driving licences (particularly photocard variants) and passports are popular, as are third-party schemes such as the Proof of Age Standards Scheme.

The systematic approach proposed is similar to the use of Casper [15] but involves other aspects such as the actors in the scenario, the physical movement of credentials, and so on. Interruptions and modifications can occur, involving such behaviours as intercepting a document so that a third party can use it, or the intended recipient modifying it to support different claims such as a different age.

At a high level, this work has some relation to what are sometimes called "serious games", such as military simulations and wargames, public order models and fire/evacuation models (as exemplified by Galea's group at Greenwich). Looking more closely at actors, there are classic belief-desire-intention models and competency models (in the context of training, learning, assessment and development) which could have influence on development of our work.

The most closely related work we are aware of is due to Martina and Carlos [16], where they describe the same type of issue, *i.e.*, that security protocols

need to take into account the environment. This necessarily includes the interaction of humans. Subsequent work [6] presents first steps to an Isabelle/HOL model, which distinguishes the work from our attempts at a simulational and model-checking model.

Ideally, we would like to exploit domain-specific validation and analysis toolsets such as Lowe's Casper [15], which is itself based on FDR2 [8]. However we also desire a way to simulate individual runs of a scenario. This is to allow us to explain and further understand how individual results might come about. Thus, our current prototype simply enumerates every possible run (every trace in process algebra terms) by brute force, but also allows us to run this in an interactive mode (as in FDR2 *vs.* Probe [21]). Other notations and tools could also be considered. We could enhance our CSPsim [5], but this will likely involve a significant amount of effort for proof-of-concept, as it would require us to extend the formal model and the model checker, when we are at a stage of evaluating whether the approach to scenario modelling provides benefit.

Tools such as PRISM [13] potentially can be applied to such problems. It has a rich language capturing a range of probabilistic features (from simple probabilities for nondeterministic choice to variants of Markov chains), with a tool that allows exhaustive calculation and path simulation. However, it does not allow potentially expensive exhaustive exploration to be performed and then repeatedly re-calculated for the probabilistic calculations (as in our current prototype). Confusingly, there is another tool called PRISM [1] based on Prolog. Programs for this tool are logic programs in which facts have a parameterised probability distribution: this is very similar to the mechanics of the approach we have adopted in our work. A difference is that it does not support exhaustive exploration of the scenarios. Similar reasoning applies to considerations of tools such as SPIN [25]. However, state-rich formalisms such as Event-B [2] or Alloy [12] may provide a sensible basis for supporting rich simulations, but these require extensions to associate probability distributions with events (and hence interactions). Other more simulation-oriented tools include Anylogic [28] and Demos2k [17]. A useful list of such tools is available from Rizzoli [22]. Monahan's DXM [17] is notable for using a Monte Carlo style of modelling. Lanotte *et al.* [14] outline an analytical approach using probabilistic transition systems. These tools and approaches, with suitable extensions, could support analysis of probabilistic models as described in this paper, but we would then lose the benefit of using model checking infrastructure to support analysis outside of exhaustive state exploration.

The model used in this paper is inspired by fuzzy logic [29] and probabilistic logics. Significant tool support for modelling fuzzy logic-based scenarios was provided by XFuzzy [18], which included a specification language as well as rules for manipulating linguistic variables. XFuzzy focused on analysing block diagrams rather than scenarios, making it unsuitable for problems like those considered in this paper. There has also been extension to logics for reasoning about probability; besides the state machine models underpinning PRISM, extensions to

refinement calculi [20] and formal specification languages (*e.g.*, Event-B [19]) have been proposed.

The work in this paper is related to that of reasoning in the presence of uncertainty, *e.g.*, as exemplified by research on analysis incomplete or inconsistent requirements specifications. Most noteworthy here is work on the XCheck model checker [7] which provides a multi-valued symbolic model checker; such a model checker provides probabilistic support but not support for reasoning with probability distriutions and for manipulating configurations of scenarios.

Substantial work has been carried out on the design and implementation of DSLs; [26] is a comprehensive reference, and many of the important design principles and technical patterns for implementing DSLs are discussed in [9]. DSLs are typically defined to support modelling of a particular problem domain, to address a particular engineering tasks (*e.g.*, a form of analysis), or to enable particular stakeholder groups to more easily express their concerns. DSLs have been constructed in a number of ways, including via embedding of a DSL in another, more general-purpose language, and through abstraction of other languages.

A significant example of the latter approach is MetaBorg [4], which is a transformation-based approach for the definition of embedded textual DSLs implemented based on the Stratego framework. The MetaBorg approach defines new concepts by mapping them to expansions in the host language. MetaBorg was designed for textual languages. This means that the transformations involved can be expressed by means of transformations on abstract syntax trees rather than abstract syntax graphs. Stratego is also used in [10] to develop an external DSL for web development.

The underlying notion of an embedded DSL seems to have been discussed first by Hudak [11]. The idea of *forwarding* that is now a standard technique for defining embedded textual DSLs using attribute grammars was introduced in [27].

7 Preliminary conclusions

We claim that it is possible to construct scenarios involving our desired properties in such as way that they can be explored and produce useful results, at least for comparing different configurations. A major difficulty of the work is to properly set the scope: too small and it's not expressive enough; too large and we can perform no analysis. A research question here is: Can we find a viable scope for interesting scenarios? Routine scenarios are easy: they involve all the actors behaving "correctly" and no attackers. Yet the interesting scenarios are where people make mistakes and others are actively attacking the system.

The DSL is currently being elaborated to provide more detail. Our next step is to present a more elaborate and rigorous definition of the DSL, perhaps using Eclipse-based EMF tools, and implement a suitable model-to-text transformation that enables compilation, analysis and simulation via state-of-the-art simulation tools. A prototype implementation in Python (with unvalidated parameters) suggests a one or two percentage point difference in true/false pos-

itive/negative outcomes between different configurations of an age-related scenario that might have used ID cards. As a result, we suspect that the actors' behaviour will swamp technical aspects of protocols, hence it is valuable to pursue this line of work.

This work is closely related to the model checking of protocols. There are necessarily assumptions about the protocol that the actors follow in particular scenario. Additionally, we have to consider the probabilities associated with some decisions. As well as the general scenario correctness problem —however we formulate a model, it will always be an abstraction of a real system— this inclusion of probabilities adds additional validation problems. One approach is to test the stability of any results under perturbation. If the results are unstable under perturbation of the parameters, the validity of the model is questionable. Automating this perturbation analysis would result in greater computational demands.

Rich state gives us less scope to reduce the size of the state space because the actors know their own history. As this history is part of the state, it is less likely that there will be a significant number of identical states following transitions. Classical model checking approaches (*e.g.*, FDR2 [8], SPIN [25]) or even lazy approaches such as CSPsim [5] each have a restricted set of data types. A richer simulation worsens this problem.

As an alternative, describing the scenario in an interactive fiction environment such as TADS3 [23] would certainly satisfy the domain-specific and individual path simulation requirements, as well as adding rich simulation capabilities. Automated exploration would require some work (*e.g.*, a driver program), along with a way to abstract the probability distributions. Canonicalisation is also harder due to the opaque state within the interactive fiction interpreter.

Acknowledgements

The authors thank the delegates to the Twentieth International Workshop on Security Protocols for their constructive comments on this work.

References

1. PRISM: PRogramming in Statistical Modeling. `http://sato-www.cs.titech.ac.jp/prism/`, Feb 2012.
2. Jean-Raymond Abrial. *Modeling in Event-B: System and Software Engineering.* Cambridge University Press, 2010.
3. Ross J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems.* Wiley, 2nd edition, 2008.
4. Martin Bravenboer and Eelco Visser. Concrete syntax for objects: Domain-specific language embedding and assimilation without restrictions. In *Proc. 19th Annual ACM SIGPLAN Conf. on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'04)*, pages 365–383. ACM Press, 2004.

5. Phillip J. Brooke and Richard F. Paige. Lazy exploration and checking of CSP models with CSPsim. In Alistair A McEwan, Wilson Ifill, and Peter H. Welch, editors, *Communicating Process Architectures 2007*, pages 33–50, February 2007.
6. Marcelo Carlomagno Carlos, Jean Everson Martina, Geraint Price, and Ricardo Felipe Custódio. A proposed framework for analysing security ceremonies. In *Proc. SECRYPT*, 2012.
7. Steve M. Easterbrook and Marsha Chechik. A framework for multi-valued reasoning over inconsistent viewpoints. In *ICSE*, pages 411–420, 2001.
8. FDR2 model checker. `http://www.fsel.com/software.html`. Last visited 12th Jan 2012.
9. Martin Fowler. *Domain-Specific Languages*. Addison-Wesley, 2010.
10. Zef Hemel, Lennart C. L. Kats, and Eelco Visser. Code generation by model transformation: A case study in transformation modularity. In Jeff Gray, Alfonso Pierantonio, and Antonio Vallecillo, editors, *Int'l Conf. on Model Transformation (ICMT'08)*, volume 5063 of *LNCS*. Springer, 2008.
11. Paul Hudak. Modular domain specific languages and tools. In *Proc. 5th Int'l Conf. on Software Reuse*, pages 134–142. IEEE Computer Society Press, 1998.
12. Daniel Jackson. *Software Abstractions*. MIT Press, 2008.
13. Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM: Probabilistic symbolic model checker. In Tony Field, Peter Harrison, Jeremy Bradley, and Uli Harder, editors, *Computer Performance Evaluation: Modelling Techniques and Tools*, volume 2324 of *LNCS*, pages 113–140, 2002.
14. Ruggero Lanotte, Andrew Maggiolo-Schettini, and Angelo Troina. Parametric probabilistic transition systems for system design and analysis. *Formal Aspects of Computing*, 19:93–109, 2006.
15. G. Lowe and B. Roscoe. Using CSP to detect errors in the TMN protocol. *IEEE Transactions on Software Engineering*, 23(10):659–669, 1997.
16. Jean Everson Martina and Marcelo Carlomagno Carlos. Why should we analyse security ceremonies? In *Proc. CryptoForma workshop*, May 2010.
17. Brian Monahan. DXM — Demo2k eXperiments Manager. Technical Report HPL-2008-173, HP Laboratories, 2008.
18. Francisco Jose Moreno-Velo, Iluminada Baturone, Santiago Sánchez-Solano, and Angel Barriga Barros. Xfuzzy 3.0: a development environment for fuzzy systems. In *EUSFLAT Conf.*, pages 93–96, 2001.
19. Carroll Morgan, Thai Son Hoang, and Jean-Raymond Abrial. The challenge of probabilistic Event-B - extended abstract. In *ZB*, pages 162–171, 2005.
20. Carroll Morgan, Annabelle McIver, and Karen Seidel. Probabilistic predicate transformers. *ACM Trans. Program. Lang. Syst.*, 18(3):325–353, 1996.
21. ProBE — CSP animator. `http://www.fsel.com/software.html`. Last visited 2nd Feb 2011.
22. Andrea Emilio Rizzoli. A collection of modelling and simulation resources on the internet. `http://www.idsia.ch/~andrea/sim/simtools.html`, last accessed 6th January 2012.
23. Michael J. Roberts. TADS 3 downloads. `http://www.tads.org/tads3.htm`. Last visited 4th January 2012.
24. Mary Beth Rosson and John Carroll. Scenario-based design. In *The Human-Computer Interaction Handbook*, chapter 53, pages 1032–1050. Lawrence Earlbaum Associates, 2002.
25. SPIN — model checker. `http://spinroot.com/spin/whatispin.html`. Last visited 4th January 2012.

26. Arie van Deursen, Paul Klint, and Joost Visser. Domain-specific languages: an annotated bibliography. *SIGPLAN Not.*, 35(6):26–36, 2000.
27. Eric Van Wyk, Oege de Moor, Kevin Backhouse, and Paul Kwiatkowski. Forwarding in attribute grammars for modular language design. In R. N. Horspool, editor, *Int'l Conf. on Compiler Construction*, volume 2304 of *LNCS*, pages 128–142. Springer, Berlin / Heidelberg, 2002.
28. XJ Technologies. Anylogic. `http://www.xjtek.com/anylogic/why_anylogic/`, last accessed 6th January 2012.
29. L. Zadeh. Fuzzy sets. *Information and Control*, 8(3), 1965.