INFORMATION SOCIETY TECHNOLOGIES
(IST)
PROGRAMME

Project IST-2001-33562 MoWGLI

Report n. D6.b
Validation 2: Smart Card Security

Eduardo Gimenez

Project Acronym: MoWGLI
Project full title: Mathematics On the Web: Get it by Logic and Interfaces
Proposal/Contract no.: IST-2001-33562 MoWGLI

Contents

1 Introduction

Trusted Logic explored the use of Mowgli's prototype as a support for the certification of IT products based on the Common Criteria standard (CC). This international standard is presently recognized in several countries of the European Union, like France, Germany, Spain and the UK.

The use case is the certification of a security IT product, like a smart credit card, an electronic passport, or an e-government signature creation device based on the CC standard. That standard requires the developer of the product to produce several kinds of documents describing the software embedded on such devices. Examples of that documents are:

- The Security Target, which specifies the security requirements on the product
- The Security Policy Model, which details the security policies that the product meets
- The Functional Specification, which describes the external interfaces of the product
- The High Level Design, which describes the products architecture
- The Low Level Design, which introduces the implementation details

Depending on the Evaluation Assurance Level claimed by the sponsor of the evaluation, some of those documents must be presented using an informal, semi-formal or completely formal languages. Informal documents are just text in English. Semi-formal documents are usually based on graphical specification languages, like for instance the Unified Modeling Language (UML). Formal documents are written using a language based on a formal calculus, like Coq.

On that use scenario, Mowgli's prototype may support the task of the CC Evaluator on three possible axes:

1. Providing an Security Policy Model with semantic contents. This means a document that contains both English explanations but also the logical structure of the formal definitions of the security policy, that the Evaluator may directly check using a proof assistant.
2. As a tool for explaining the formal description of the product, that provides support for the exploration (hyperlinks, search tools) and the appropriate rendering of the formal definitions.
3. As a neutral exchange format for transforming formal models into semi-formal ones (composite evaluations) and to communicate with development teams.

Trusted Logic explored the three axes above on the formal models developed to evaluate a smart card platform based on the Java Card Technology, a raising standard in the smart card's world. Three main contribution to the Mowgli's project resulted from these experiments.

2 Extracting meta-data from Coq source files

Mowgli's prototype was tested on the Java Card formal model, which contains more than 3750 formal definitions and 2000 theorems along 4MB of Coq source code. The intended purpose was to test Mowgli as a support for explaining those models to the evaluator. Three main observation arisen from this experiment:

1. The installation of Mowgli's prototype is not a simple task. This is the consequence of at least three implementation choices. First, it is based on an heterogeneous implementation based on several different languages: ocaml, Perl, MySql, XSLT, PXP, etc. Second, it depends on several (unstable) Linux packages developed by the free software community. Third, it is based on a completely open architecture, where information can be exchanged and accessed with no restrictions, a vision that cannot be easily integrated to firewalls and other control access functions.

2. The rendering that is actually available in Mowgli's prototype is better suited for explaining Mathematics than Software. For instance, there is no support for the correct rendering of record types, functions are presented from the lamba-term perspective (and not as programs) and indentation is not always as expected. More important, there is no support for simplifying rendering modification, like an intermediate language of "boxes".

3. Mowgli's prototype takes as input the compiled format of Coq files, so all comments present in the Coq sources are lost. This is annoying, because the CC standard requires formal definitions to be accompanied by explanatory text.

4. Other information present in Coq sources that could improve readability and be exploited by rendering tools is also missed, like the coercions and the implicit arguments.

As a contribution to the point (1) above, Trusted Logic collected and packaged all the libraries that are required for compiling Mowgli's prototype an write a short installation guide to help other users. Concerning the points (3) and (4), Trusted Logic developed a tool for retrieving source information as meta-data. This includes several kinds of information that present in the sources but that is not part of logical terms, like comments on the formal definitions included in the source files, coercions, implicit arguments, derived constants, etc.

In order to be recognized by the meta-data extractor, source comments must be written in a style close to the one used by literate programming tools, like javadoc: tags inside the comment preceeding a top-level definition (an inductive type, an axiom, a theorem, etc) are recognized, collected, and associated to the definiens. The output of the extractor is an RDF file that can be entered in the data base of Mowgli's prototype. The information inside the comment can be structured using tags. A tag is any word starting with the character `@`, like for instance `@param`. The paragraph following the tag is classified as the description of one of the definition's parameter and displayed as such by Mowgli's prototype.

Another RDF file is produced with the logical information that cannot dumped in the XML format of the terms because it is computed on-the-fly by Coq's compiler without leaving any trace in the terms. This presently includes coercions, implicit arguments, derived constants. This RDF file is produced by a modified version of Coq's compiler.

The meta-data extractor for CIC terms is available on-line at
`http://mowgli.cs.unibo.it/html_no_frames/software/index.html`

3 Security Policy document

The second contribution that Trusted Logic developed is an SPM document of GlobalPlatform, a widely used card manager standard for smart cards. This document describes the models of three different security policies of GlobalPlatform: the control of the actions enabled in each life cycle state of the smart card, the enforcing of the life cycle transitions, and the integrity of the executable files downloaded on the card. Each security policy is modeled as an abstract state machine which process the requests that the smart card receives. The state of the abstract machine is made of the security attributes of each subject and object under the control of the security policy. The transitions of the abstract machine describes the premises under which a given subject may process a request on a given object, as well as the side effects on the security attributes that the operation entails. The SPM document contains XML directives that are processed by the XSLT transformations of Mowgli's prototype. Those directives enable to inline Coq definitions in the text flow and to introduce hyperlinks for navigating across the formal definitions of the model.

One of the main conclusions from this experience is that Mowgli's prototype could be enriched with an XSLT transformation for displaying formal definitions in English. Actually, when writing an SPM model (or

a scientific article describing a proof in Coq), most of the authors re-phrase in English the formal definitions in Coq with the aim of clarifying its meaning to the reader. Similarly to what Mowgli's prototype actually does with formal proofs, definitions could also be translated into structured English. This idea is illustrated by the following formal definition in Coq and its possible translation into English:

```
Inductive le (n:nat) : nat -> Prop :=
  | le_n : le n n
  | le_S : forall m:nat, le n m -> le n (S m).
```

Definition 1 ***: less than****. The natural number n is less than another natural number if and only if one of the following rules is satisfied:*

- *Rule le_n: n is less than n*
- *Rule le_s: given a natural number m, n is less than (S m) provided that n is less than m.*

If the symbol *nat* is associated to the phrase *natural number* and the symbol `less` is associated to the notation ... *is less than* ..., then the definition above could be produced by automated means from the formal definition in Coq (like, for instance, an XSLT stylesheet). In this way, the task of the author of the SPM could be rather concentrate on describing the modeling choices, rather than rephrasing the definitions themselves.

The xml files containing the SPM document is available on-line at
`http://mowgli.cs.unibo.it/html_no_frames/software/index.html`

4 A translator form Coq to UML.

The third contribution that Trusted Logic developed is related to the use of Mowgli's XML output as an exchange format for formal models. That format can be used as a "neutral" format, as opposite to the "proprietary" one used by the Coq team (the so-called `vo` format). The translator enables to communicate formal models to development teams that use CASE tools based on UML or to link formal models to other UML models provided by the clients. The tool enables the automatic integration of formal models in Coq into the Rational Rose UML environment.

Starting from the XML format for Coq terms introduced in Mowgli, Coq's data structures, predicates and comments are translated into XMI, an XML format for describing UML models. The XMI file is then imported into Rational Rose using a special add-in provided by the environment. To resulting model contains class diagrams describing those data structures and three kinds of UML relations: associations, generalizations and dependencies.

Concerning informative objects, the translation process applies the following rules:

- Each directory containing Coq modules is recursively translated into an UML package. This enables to group together several files, so respecting the structure of the Coq model.
- Coq module M is translated into a root class M.
- Each Coq type $T : Set$ is represented in UML as a class T.
- Each dependent type $T : (A : Set)Set$ is represented as a parameterized class $T[A]$
- Each constant of type c:T:Set is represented as a static final field of the class T.
- Each record type (ie, inductive type with a single constructor) $R\{x_1, \ldots x_n\}$: Set is represented as a class R with instance fields $x_1, \ldots x_n$

- Each function $f\{x_1 : T_1, \ldots x_n : T_n\}$ is represented as an instance method f of the class C if a suitable class C is found for f. The following heuristic is used to find a suitable class: if T_1 is close enough to f, that is, if it is defined in the same module than f, then C is defined as T_1. Otherwise, f is classified as an operation of the root class M associated to the Coq module where f is defined. This heuristic could be easily replaced by an explicit classification of f that the used could provide as part of the formatted comment describing f. Such information could be easily exploited by the translator using the meta-data extractor from Coq sources.

- Each coercion $f : A \to B$ is represented as a subclass relation arrow from class A to class B.

- The diagram obtained using the rules above is enriched with dependency associations between classes in order to show the relationship between parametric classes and their instances: if $T : Set$ is defined as an instance of the parametric type $F[A]$, then a definition arrow linking T to F is created.

The class diagram so obtained is completed by the introduction of predicates relating the informative classes. Such predicates are also represented as classes, using the following translation rules:

- Each predicate $P\{x_1 : T_1, \ldots x_n : T_n\} : Prop$ is viewed as a sub-class of the Cartesian product $T_1 \times \ldots T_n$. This classical interpretation of predicates is represented in UML introducing a class P with n instance fields $x_1 : T_1, \ldots x_n : T_n$. An instance of that class is therefore a tuple $x_1 : T_1, \ldots x_n : T_n$ representing a proof of the set $P \leq T_1 \times \ldots T_n$. As usual in the UML class diagrams, the instance field $x : T$ of the class C is represented a directed association relation from C to T labeled in its origin with x.

- If the predicate P is an inductive one, each constructor is represented as a constructor of the class P constructing a tuple in $T_1 \times \ldots T_n$.

- More in general, in this interpretation a theorem would become an operation producing an instance (proof) in some class (type).

The following example illustrates how an inductive predicate in Coq is translated into UML. Consider the following inductive predicate P:

```
Inductive P : nat -> bool -> Prop :=
          C : forall x : nat . 1<x -> (P (S x) true).
```

Assuming that the the predicate *less* associated to the notation "1¡x" has been already translated, the predicate P give raise to the following UML definition:

```
public class P {
  public nat n;
  public bool b;
  public C(nat x; less p){
         n:=S(x);
         b:=true
  }
}
```

Using this technique, UML diagrams can be used to have an overview (actually, a picture) of all the data-structures of the Coq model and the different relations linking those data-structures. Even tough the general interpretation provided above enables the interpretation of all the theorems of the Coq model as well, we preferred to restrict it to data-structures and predicates. The reason of this choice is that too much information would rend the diagrams incomprehensible. Statements like theorems, axioms, etc are

integrated into the model as html links to the root class containing the statement. When clicking in one of those links, Rational Rose launches the default web browser and requests Mowgli's server to display the rendering of the statement of the theorem and its proof in English. Therefore, the translator takes advantage of the XML format for Coq models on which Mowgli's prototype lays on, but it also uses the UWOBO as a database of theorems and as a rendering tool for statements and proofs. Moreover, the translation process itself uses Mowgli's prototype to retrieve the coercion meta-data used to represent subclass relationships between UML classes. Meta-data comments on each Coq definition (source comments) are also integrated into the UML model.

When developing UML diagrams through automatic generation, one important concern is to generate readable pictures, containing a reasonable amount of information and relations. Several axes have been explored on this issue. First, in addition to the whole diagram containing all the data-structures used in the model, local diagrams by Coq modules are also derived. These latter diagrams only contain the types and predicates defined in one given Coq module and the definitions from the imported modules that are directly related to them. This is usually the kind of information that one likes to see in order to have a global picture of one system in the model architecture. Second, some implicit functions that are included in the XML description of the model are filtered during the generation of the UML one. For instance, the induction principles and the recursion combinators that Coq automatically derives for each inductive type are removed from the UML model, as they only provide *logical* information. Similarly, the projection functions associated to a record type, which Coq also automatically derives from the record, are also removed from the UML model, as they are already represented by the instance fields of each UML class.

The sources of the translator from Coq to UML and some screen-shots of its output in Rational Rose are available on-line at `http://mowgli.cs.unibo.it/html_no_frames/software/index.html`