Universität Augsburg

SecureMDD: Transformation of a UML application model to a formal specification

Nina Moebius, Marian Borek, Kurt Stenzel and Wolfgang Reif

Report 2012-10 November 2012

Institut für Informatik
D-86135 Augsburg

Abstract

The SecureMDD project provides a software engineering approach to develop secure smart card applications. The approach is model-driven and integrates formal verification to guarantee the security of the application under development. Furthermore, based on a platform-independent UML model of the application under development, the approach is able to generate executable source code for the smart cards and terminals of the application. The whole approach is fully supported by tools and all model-to-model- as well as model-to-text-transformations are fully implemented. This paper contains the implementation of the transformations that generate a formal specification out of the platform-independent UML model of an application. The formal specification is based on algebraic specifications and Abstract State Machines (ASM). The formal model can be loaded into the interactive theorem prover KIV and is used to verify security properties for the modeled application.

Contents

Chapter 1

Approach

The SecureMDD approach supports the development of applications that use web services, smart cards and terminals. Terminals are devices that consist of at least one smart card reader and can communicate with a smart card. Moreover a terminal usually has a user interface to communicate with the user of the application and can also call service operations. For example, a terminal could be a Home-PC or an automat.

In this Section we shortly summarize the approach. It has been described in more detail in [5]. Fig. 1.1 contains an overview.

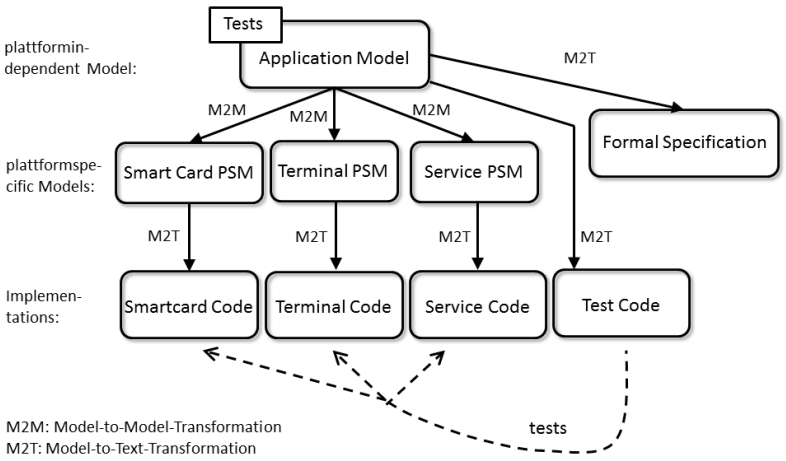


Figure 1.1: Overview of the SecureMDD approach

The development of an application starts with the creation of a platform-independent UML model. This is an abstract view of a system, omitting implementation details. To be able to model security-critical applications, UML was tailored to this domain by defining a UML profile. To support the modeling of the dynamic part of an application, i.e. the communication as well as the processing of messages, we defined a domain-specific language called Model Extension Language (MEL) which is used in UML activity diagrams. With this language it is possible to make assignments to the attributes of component classes, to create objects or to call predefined cryptographic operations. The platform-independent UML model of an application consists of all information that are needed to generate executable code as well as a formal model of the whole

application automatically.

The platform-independent model is used to generate three platform-specific models (PSM), one for the modeled smart card components, one for the terminals and one for the modeled services. The platform-specific models contain technical information about the implementation. Their aim is to minimize the gap between the platform-independent model and the generated code. For example, the Smart Card PSM contains classes with operations for the serialization and deserialization of messages. Since the communication with smart cards in the implementation is based on byte arrays (called APDUs [3]), these operations are used to transform a message into a byte array before sending it.

The platform-specific model is transformed into code using model-to-model and model-to--text transformations. For terminals and web services we generate Java code. Smart cards are programmed with Java Card ([3]), which is a subset of Java and is tailored for the use on resource-constraint devices. Thus, for smart cards we generate Java Card code.

The modeled application and especially the cryptographic protocols may contain functional and security errors. We integrated model-based testing into the approach that allows to design a test case with UML and generate test cases automatically from this model (see [4]). Then, these test cases can be executed on the generated code. Currently, tests to find protocol logic flaws as well as security tests are supported. To test the security, possible attacks are modeled with UML. The generated test cases check if these attacks are executable on the generated code.

To guarantee the security of an application, the approach integrates the formal verification of application-specific security properties (see [6]). Application-specific properties are e.g. that an electronic prescription that is stored on an electronic health card cannot be filled twice in a pharmacy. Other properties are that only genuine prescriptions can be filled, i.e. only a doctor is able to issue a prescription, and that the attacker does not get to know any prescriptions. In our opinion, for many applications application-specific security properties give better guarantees to the security of an application than standard properties like secrecy, integrity or authenticity. However, standard properties are often prerequisites for proving application-specific properties and thus, have to be verified as well. Our approach generates a formal specification based on algebraic specifications and abstract state machines (ASM) [2] for the modeled application. The generated formal model is loaded into the theorem prover KIV [1] and used for interactive verification of application-specific security properties.

The security properties that are proved to hold on the formal model also have to hold on code level. To guarantee this, the generated code has to be a refinement of the generated formal model. A solution that proves the refinement for any application that is developed with the SecureMDD approach is work in progress. A first result is the definition of a calculus for QVT (the language used to implement the model-to-model transformations) in KIV. This calculus can be used to prove the correctness of QVT transformations (see [7]).

The SecureMDD approach is fully supported by tools. For creation of the platform-independent UML model we make use of the UML modeling tool Magic Draw[1]. The transformations of the UML model into code as well as into the formal specification are executable using Eclipse plugins. The model-to-model transformations are implemented in Operational QVT[2]. The model-to-text transformations are implemented in Xpand[3]. As development environment as well as to execute the transformations we use the Eclipse Modeling Project[4]. The parsing and annotation of the MEL expressions is implemented in Java.

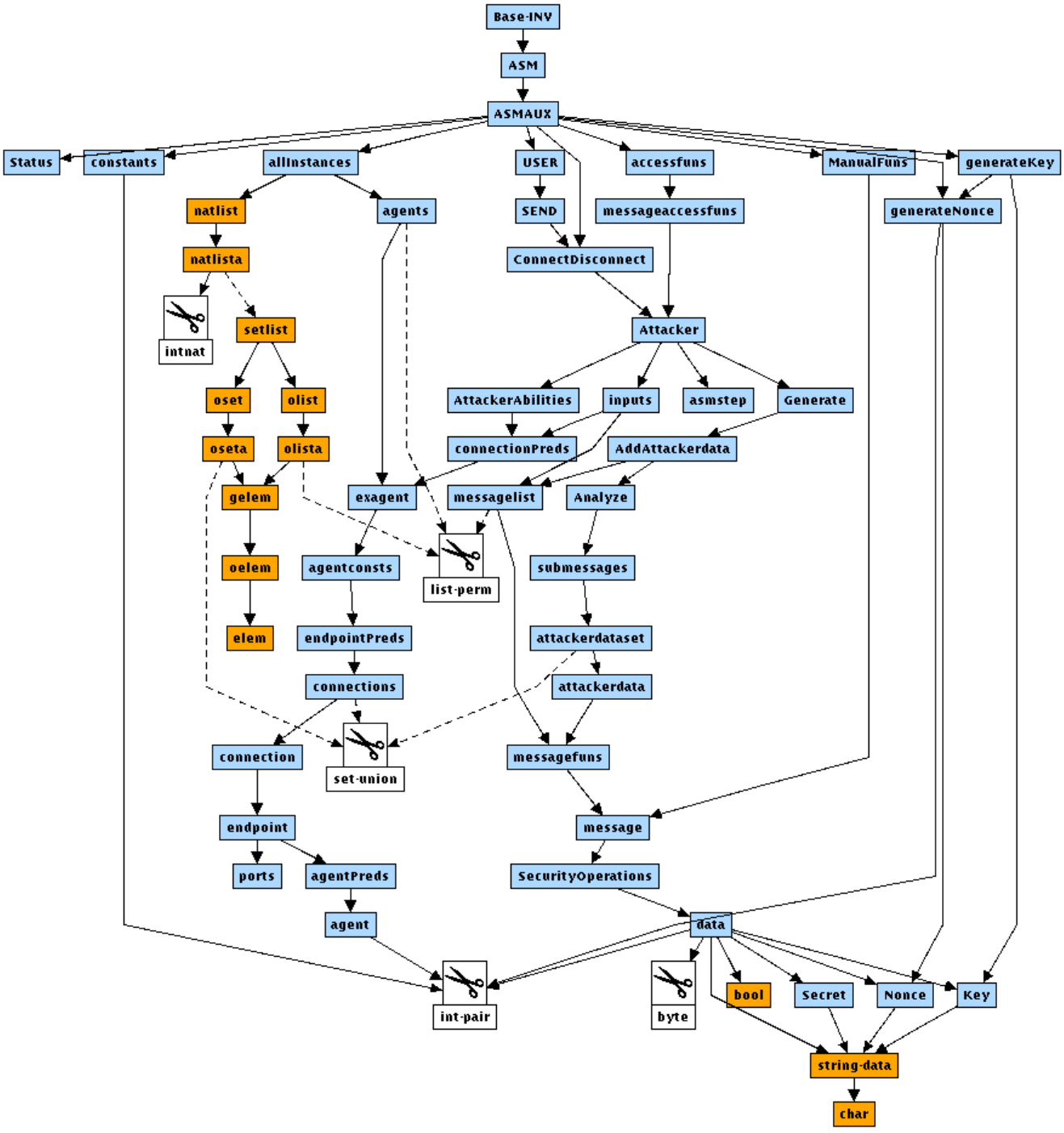
This paper contains the implementation of the model-to-text-transformations that generate a formal specification for smart card and terminal applications out of the platform-independent UML model. The generation of a formal model for web services is not supported yet. The following chapters show the model-transformations that generate the formal specification from the UML model of an application.

[1] http://www.nomagic.com/
[2] http://www.eclipse.org/m2m/
[3] http://www.eclipse.org/modeling/m2t/?project=xpand
[4] http://www.eclipse.org/modeling/

Chapter 2

Generation of a formal specification

The following graphics shows a generated formal specification. Orange nodes and clipped nodes are library nodes that are imported into this project. Blue nodes show specifications that are generated by XPand transformations.


Base-INV
ASM
ASMAUX
Status
constants
allInstances
USER
accessfuns
ManualFuns
generateKey
natlist
agents
SEND
messageaccessfuns
generateNonce
natlista
ConnectDisconnect
intnat
setlist
Attacker
oset
olist
AttackerAbilities
inputs
asmstep
Generate
oseta
olista
connectionPreds
AddAttackerdata
gelem
exagent
messagelist
Analyze
oelem
agentconsts
list-perm
submessages
elem
endpointPreds
attackerdataset
connections
attackerdata
connection
set-union
messagefuns
endpoint
message
ports
agentPreds
SecurityOperations
agent
data
int-pair
byte
bool
Secret
Nonce
Key
string-data
char

Chapter 3

Transformations for the ASM

```
«IMPORT uml»
«EXTENSION templates::parse»
«EXTENSION templates::getter»
«EXTENSION templates::deployment»

«DEFINE Root FOR uml::Model»
«this.setContext()»
«EXPAND generateASM FOR this»
«EXPAND generateASMAUX FOR this»
«ENDDEFINE»

«DEFINE generateASMAUX FOR Model»
«FILE "ASMAUX/specification"»
enrich USER, allInstances, ConnectDisconnect
  , ManualFuns, accessfuns
«IF this
  .existsFakeCardlet()», FakeCardletSimple
    «ENDIF-»
«IF this.constantsExist()-», constants
  «ENDIF-»
«IF this.statusExists()-», Status
  «ENDIF-»
«IF this.usesSymmKey()», generateKey
  «ENDIF-»
«IF this.existsNonce()», generateNonce
  «ENDIF-»
«FOREACH this.getListAssociations()
  AS a», listof«a.type.name»funs
    «ENDFOREACH»
with
«getASMAUXSpec()»
end enrich
«ENDFILE»
«ENDDEFINE»

«DEFINE generateASM FOR Model»
«FILE "ASM/specification"-»
asm specification ASM
using ASMAUX
```

```
≪getASMVariablesDef()≫
≪EXPAND generateSTEPRule FOR this–≫
≪activities2ASM()≫
end asm specification
≪ENDFILE–≫
≪ENDDEFINE≫

≪DEFINE generateSTEPRule FOR uml::Model≫
asm rule STEP
declaration
STEP : STEP
{
 let asm-step = [?], exception_occurred = false
 in
 {
  if asm-step = connect then
  { CONNECT(; connections, inputs) }
  else if asm-step = disconnect then
  { DISCONNECT(; connections, inputs) }
  else if asm-step = attacker-agent-step then
  { ATTACKER(connections, attacker-known; inputs) }
  else if asm-step = user-agent-step then
  { choose ag with (is_user(ag)
    ≪convertSymbol("and")≫ exagent
      (ag)) in USER
        (ag, connections; attacker-known, inputs
          ) ifnone skip }
  ≪FOREACH this.getAllSmartcards() AS scclass–≫
  else if asm-step =
    ≪scclass.name≫-agent-step then
  { choose ag with (is_≪scclass.name≫(ag)
    ≪convertSymbol("and")≫ exagent
      (ag)) in
        ≪scclass.name
          .toUpperCase()≫STEP ifnone skip }
  ≪ENDFOREACH–≫
  ≪FOREACH this.getAllTerminals()
    AS terminalclass–≫
  else if asm-step =
    ≪terminalclass.name≫-agent-step then
  { choose ag with (is_
    ≪terminalclass.name≫(ag)
      ≪convertSymbol("and")≫ exagent
        (ag)) in
          ≪terminalclass.name
            .toUpperCase()≫STEP ifnone skip }
  ≪ENDFOREACH–≫
  ≪FOREACH this.getAllServices()
    AS serviceclass–≫
  else if asm-step =
    ≪serviceclass.name≫-agent-step then
  { choose ag with (is_
    ≪serviceclass.name≫(ag)
      ≪convertSymbol("and")≫ exagent
```

```
        (ag)) in
          ≪serviceclass.name
            .toUpperCase()≫STEP ifnone skip }
  ≪ENDFOREACH-≫
  ≪IF this.existsFakeCardlet()-≫
  else if asm-step = forgeable-cardlet-agent-step then
  { choose ag with (is_fakecardlet
    (ag)) in SIMPLE-FAKE-CARDLET
      (ag, connections; fake-cardlet-known, inputs
        ) ifnone skip }
  ≪ENDIF-≫
 };
  stop := [?];
};
≪ENDDEFINE≫
```

Chapter 4

Transformations for agents and messages

```
«IMPORT uml»
«EXTENSION templates::getter»
«EXTENSION templates::attacker»
«EXTENSION templates::deployment»

«DEFINE Root FOR uml::Model»
«EXPAND generateFromClassDiagram»
«ENDDEFINE»

«DEFINE generateFromClassDiagram FOR Model»
«EXPAND generateAgent FOR this»
«EXPAND generateAgentPreds FOR this»
«EXPAND generateAgentList FOR this»
«EXPAND generateData FOR this»
«EXPAND generateDatatypesList FOR this»
«EXPAND generatePlainDataList FOR this»
«EXPAND generateHashDataList FOR this»
«EXPAND generateSignDataList FOR this»
«EXPAND generateMessages FOR this»
«EXPAND generateMessagefuns FOR this»
«EXPAND generateMessagelist FOR this»
«EXPAND generateConstants FOR this»
«EXPAND generateStatus FOR this»
«ENDDEFINE»

«DEFINE generateAgent FOR Model»
«FILE "agent/specification"-»
data specification
using int-pair
agent =
«FOREACH this.getAllTerminals()
  AS terminalclass-»
«terminalclass.name»( .
  .name : nat) with is_«terminalclass.name» |
«ENDFOREACH-»
«FOREACH this.getAllSmartcards() AS cardclass-»
```

```
≪cardclass.name≫( . .name : nat) with is_
  ≪cardclass.name≫ |
≪ENDFOREACH–≫
≪FOREACH this.getAllServices()
  AS serviceclass–≫
≪serviceclass.name≫( .
  .name : nat) with is_≪serviceclass.name≫ |
≪ENDFOREACH–≫
≪IF this.existsFakeCardlet()–≫
fake–cardlet( . .name : nat) with is_fakecardlet |
≪ENDIF–≫
user( . .name : nat) with is_user |
attacker with is_attacker;
variables ag, ag0, ag1 : agent;
end data specification
≪ENDFILE≫
≪ENDDEFINE≫

≪DEFINE generateAgentPreds FOR Model≫
≪FILE "agentPreds/specification"–≫
enrich agent with
predicates
is_statefulService : agent;
axioms
≪LET this.getAllServices().select(e|e
  .isStateful()) AS statefulclasses–≫
≪IF statefulclasses.isEmpty–≫
is_statefulService : not is_statefulService
  (ag); used for : s,ls;
≪ELSE–≫
is_statefulService : is_statefulService(ag) <–>
 (
 ≪FOREACH statefulclasses
   AS serviceclass SEPARATOR convertSymbol("or")–≫
 is_≪serviceclass.name≫(ag)
 ≪ENDFOREACH–≫
 );
≪ENDIF–≫
≪ENDLET–≫
end enrich
≪ENDFILE≫
≪ENDDEFINE≫

≪DEFINE generateData FOR Model≫
≪FILE "data/specification"–≫
≪LET this.getNormalClasses() AS dataclasses–≫
≪LET dataclasses.isEmpty AS noData≫
≪IF noData≫enrich
  ≪ELSE≫data specification
using
≪ENDIF≫
bool, int–pair, string–data, byte
  ≪IF this.existsSecret() || this
    .usesSymmKey()≫, Secret≪ENDIF–≫
```

```
«IF this.existsNonce() || this
  .usesSymmKey()», Nonce«ENDIF-»
«IF this.usesSymmKey() || this.uses
  ASymmKey()», Key«ENDIF-»
«IF ! noData»
«FOREACH dataclasses AS c-»
«c.name» = mk«c.name»
  «c.getArguments()»;
«ENDFOREACH»
«IF ! dataclasses.isEmpty»
data =
  «FOREACH dataclasses AS c SEPARATOR " | "-»
   «c.WRAP2("data")-»(. .
     «c.name.toFirstLower()-» :
       «c.name-») with is«c.name»
   «ENDFOREACH-»;
«ENDIF-»
«LET this.getPlainClasses() AS plainclasses-»
«IF !plainclasses.isEmpty-»
PlainData =
  «FOREACH plainclasses AS c SEPARATOR " | "-»
          «c.WRAP2("PlainData")-»(. .
            «c.name.toFirstLower()-» :
              «c.name-») with is
                «c.name»
«ENDFOREACH-»;
«ENDIF-»
«ENDLET-»
«LET this.getHashClasses() AS hashclasses»
«IF !hashclasses.isEmpty-»
HashData =
  «FOREACH hashclasses AS c SEPARATOR " | "-»
          «c.WRAP2("HashData")-»(. .
            «c.name.toFirstLower()-» :
              «c.name-») with is
                «c.name»
«ENDFOREACH-»;
«ENDIF-»
«ENDLET-»
«LET this.getSignClasses() AS signclasses-»
«IF !signclasses.isEmpty-»
SignData =
  «FOREACH signclasses AS c SEPARATOR " |"-»
          «c.WRAP2("SignData")-»(. .
            «c.name.toFirstLower()-» :
              «c.name-») with is
                «c.name»
«ENDFOREACH-»;
«ENDIF-»
«ENDLET-»
«IF this.usesEncryption()-»
EncData = mkEncData( . .key : SymmKey; .
  .plain : PlainData);
«ENDIF-»
```

```
≪IF this.usesAsymmEncryption()-≫
EncDataAsymm = mkEncDataAsymm( . .key : PublicKey; .
  .plain : PlainData);
≪ENDIF-≫
≪IF this.usesHashing()-≫
HashedData = mkHashedData( . .hash : HashData);
≪ENDIF-≫
≪IF this.usesSigning()-≫
SignedData = mkSignedData( . .key : PrivateKey; .
  .signdata : SignData);
≪ENDIF-≫
variables
≪IF !dataclasses.isEmpty-≫
≪varname("data")≫,
  ≪varname("data")≫0,
    ≪varname("data")≫1 : data;
≪ENDIF-≫
≪IF !this.getPlainClasses().isEmpty-≫
≪varname("PlainData")≫,
  ≪varname("PlainData")≫0,
    ≪varname("PlainData")≫1 : PlainData;
≪ENDIF-≫
≪IF !this.getHashClasses().isEmpty-≫
≪varname("HashData")≫,
  ≪varname("HashData")≫0,
    ≪varname("HashData")≫1 : HashData;
≪ENDIF-≫
≪IF !this.getSignClasses().isEmpty-≫
≪varname("SignData")≫,
  ≪varname("SignData")≫0,
    ≪varname("SignData")≫1 : SignData;
≪ENDIF-≫
(: variables for data, plain, hash, sign classes :)
≪FOREACH this.getNormalClasses() AS c-≫
≪c.varname()≫, ≪c.varname()≫0,
  ≪c.varname()≫1 : ≪c.name≫;
≪ENDFOREACH-≫
≪IF this.usesEncryption()-≫
≪varname("EncData")≫,
  ≪varname("EncData")≫0,
    ≪varname("EncData")≫1 : EncData;
≪ENDIF-≫
≪IF this.usesAsymmEncryption()-≫
≪varname("EncDataAsymm")≫,
  ≪varname("EncDataAsymm")≫0,
    ≪varname("EncDataAsymm")≫1 : EncDataAsymm;
≪ENDIF-≫
≪IF this.usesHashing()-≫
≪varname("HashedData")≫,
  ≪varname("HashedData")≫0,
    ≪varname("HashedData")≫1 : HashedData;
≪ENDIF-≫
≪IF this.usesSigning()-≫
≪varname("SignedData")≫,
```

```
  «varname("SignedData")»0,
    «varname("SignedData")»1 : SignedData;
«ENDIF-»
end data specification
«ELSE»
with
end enrich
«ENDIF»«ENDLET-»«ENDLET»
«ENDFILE-»
«ENDDEFINE»

«DEFINE generateMessages FOR Model»
«FILE "message/specification"-»
data specification
«LET this.getAllMessageAndUserMessageClasses()
  AS msgs»
using SecurityOperations
message =
  «FOREACH msgs AS c SEPARATOR "| "-» mk
    «c.name»
      «c.getArguments()» with is
        «c.name»
«ENDFOREACH-»;
variables
msg, msg0, msg1 : message;
«ENDLET-»
end data specification
«ENDFILE»
«ENDDEFINE»

«DEFINE generateMessagefuns FOR Model»
«FILE "messagefuns/specification"»
enrich message with
predicates
   is_user_message : message;
   is_valid_card_message : message;
   comment:true if not a user message
     and all numbers fit into a short value;
      «IF !this.getPlainClasses().isEmpty-»
   is_valid_card_PlainData : PlainData; comment
     : true if all numbers fit into a short value.;
      «ENDIF-»
      . \in . : int , message;
      comment:true if i occurs as a clear text number
      anywhere in the message. I.e is recursive,
      but does ignore encrypted/signed/hashed stuff;
      «IF !this.getPlainClasses().isEmpty-»
      . \in . : int, PlainData;
      «ENDIF-»
      «FOREACH this.getNormalClasses() AS c-»
      . \in . : int, «c.name»;
      «ENDFOREACH-»
axioms
(
```

```
  is_valid_card_message-def :
  is_valid_card_message(msg) <-> not is_user_message
  (msg) and all i. i \in msg -> i \inshort;

«FOREACH this.getAllMessageClasses() AS c-»
«c.generateIntInAxiom()-»
«ENDFOREACH-»

«FOREACH this.getNormalClasses() AS c-»
«c.generateIntInAxiom()-»
«ENDFOREACH-»
«LET this.getPlainClasses() AS plainclasses-»
«IF !plainclasses.isEmpty»

is_valid_card_PlainData-def :
is_valid_card_PlainData(a_PlainData) <-> all i
  . i \in a_PlainData -> i \inshort;
«ENDIF»
«FOREACH plainclasses AS c-»
«c.generatePlainDataIntInAxiom()-»
«ENDFOREACH-»
«ENDLET»

is_user_message-def : is_user_message(msg)
  «convertSymbol("equivalent")»
«LET this.getAllUserMessageClasses() AS umsgs-»
«FOREACH umsgs
  AS umsg SEPARATOR convertSymbol("or")-»
 is«umsg.name»(msg)
«ENDFOREACH»«ENDLET»;
end enrich
«ENDFILE»
«ENDDEFINE»

«DEFINE generateConstants FOR Model»
«FILE "constants/specification"-»
enrich int-pair with
«LET this.getAllConstants() AS classes-»
«IF classes.size > 0»
constants
«FOREACH classes AS c-»
«FOREACH c.getAllAttributes() AS att-»
«IF att.type != null-»
«att.name» : «att.translateType()»;
«ELSE-»
«att.name» : int;
«ENDIF-»
«ENDFOREACH-»
«ENDFOREACH-»
axioms
«FOREACH classes AS c-»
«FOREACH c.getAllAttributes() AS att-»
«IF att.defaultValue != null-»
«att.name» : «att.name» =
```

```
  ≪att.defaultValue.stringValue()−≫;
≪ELSE−≫
≪att.name≫ : ≪att.name≫ =
  ≪c.getAllAttributes().indexOf(att) + 1≫;
≪ENDIF−≫
≪ENDFOREACH−≫
≪ENDFOREACH−≫
≪classes.generateConstantsDiffAxioms()≫
≪ENDIF≫
≪ENDLET−≫
end enrich
≪ENDFILE−≫
≪ENDDEFINE≫

≪DEFINE generateStatus FOR Model≫
≪FILE "Status/specification"−≫
data specification
≪FOREACH this.getClassDiagramEnumerations()
  AS statusclass−≫
≪LET statusclass.name AS classname−≫
≪classname≫ =
  ≪FOREACH statusclass.ownedLiteral
    AS att SEPARATOR " | "≫≪att.name≫
      ≪ENDFOREACH−≫;
≪ENDLET−≫
≪ENDFOREACH−≫
variables
≪FOREACH this.getClassDiagramEnumerations()
  AS statusclass−≫
≪statusclass.varname()≫ :
  ≪statusclass.name−≫;
≪ENDFOREACH−≫
end data specification
≪ENDFILE−≫
≪ENDDEFINE≫

≪DEFINE generateMessagelist FOR Model≫
≪FILE "messagelist/specification"−≫
actualize list−perm with messagefuns
by morphism
elem ≪convertSymbol("implies")≫ message;
list ≪convertSymbol("implies")≫ messagelist;
a ≪convertSymbol("implies")≫ msg; a0
  ≪convertSymbol("implies")≫ msg0; b
    ≪convertSymbol("implies")≫ msg1; c
      ≪convertSymbol("implies")≫ msg2;
x ≪convertSymbol("implies")≫ msgs; x0
  ≪convertSymbol("implies")≫ msgs0; y
    ≪convertSymbol("implies")≫ msgs1; z
      ≪convertSymbol
        ("implies")≫ msgs2; y0
          ≪convertSymbol("implies")≫ msgs3;
z0 ≪convertSymbol("implies")≫ msgs4; x1
  ≪convertSymbol("implies")≫ msgs5; y1
```

```
    ≪convertSymbol("implies")≫ msgs6; z1
      ≪convertSymbol
        ("implies")≫ msgs7; x2
          ≪convertSymbol("implies")≫ msgs8;
y2 ≪convertSymbol("implies")≫ msgs9; z2
  ≪convertSymbol("implies")≫ msgs10;
end actualize
≪ENDFILE≫
≪ENDDEFINE≫

≪DEFINE generateDatatypesList FOR Model≫
≪IF !this.getNormalClasses().isEmpty–≫
≪FOREACH this.getNormalClasses()
  .calcListActualizations() AS list–≫
≪FILE list.type.name + "list/specification"–≫
actualize list–perm with ≪list.type.name≫
by morphism
elem ≪convertSymbol("implies")≫
  ≪list.type.name≫; list
    ≪convertSymbol("implies")≫
      ≪list.type.name≫s;
a ≪convertSymbol("implies")≫ a
  ≪list.type.name≫; a0
    ≪convertSymbol("implies")≫ a0
      ≪list.type.name≫; b
        ≪convertSymbol("implies")≫ b
          ≪list.type.name≫; c
            ≪convertSymbol("implies")≫ c
              ≪list.type.name≫;
x ≪convertSymbol("implies")≫ x
  ≪list.type.name≫s; x0
    ≪convertSymbol("implies")≫ x0
      ≪list.type.name≫s; y
        ≪convertSymbol("implies")≫ y
          ≪list.type.name≫s; z
            ≪convertSymbol("implies")≫ z
              ≪list.type.name≫s; y0
                ≪convertSymbol
                  ("implies")≫ y0
                    ≪list.type.name≫s;
z0 ≪convertSymbol("implies")≫ z0
  ≪list.type.name≫s; x1
    ≪convertSymbol("implies")≫ x1
      ≪list.type.name≫s; y1
        ≪convertSymbol("implies")≫ y1
          ≪list.type.name≫s; z1
            ≪convertSymbol("implies")≫ z1
              ≪list.type.name≫s; x2
                ≪convertSymbol
                  ("implies")≫ x2
                    ≪list.type.name≫s;
y2 ≪convertSymbol("implies")≫ y2
  ≪list.type.name≫s; z2
    ≪convertSymbol("implies")≫ z2
```

```
      ≪list.type.name≫s;
end actualize
≪ENDFILE≫
≪ENDFOREACH≫
≪ENDIF≫
≪ENDDEFINE≫

≪DEFINE generatePlainDataList FOR Model≫
≪IF !this.getPlainClasses().isEmpty≫
≪FOREACH this.getPlainClasses()
  .calcListActualizations() AS list-≫
≪FILE list.type.name + "list/specification"-≫
actualize list-perm with ≪list.type.name≫
by morphism
elem ≪convertSymbol("implies")≫
  ≪list.type.name≫; list
    ≪convertSymbol("implies")≫
      ≪list.type.name≫s;
a ≪convertSymbol("implies")≫ a
  ≪list.type.name≫; a0
    ≪convertSymbol("implies")≫ a0
      ≪list.type.name≫; b
        ≪convertSymbol("implies")≫ b
          ≪list.type.name≫; c
            ≪convertSymbol("implies")≫ c
              ≪list.type.name≫;
x ≪convertSymbol("implies")≫ x
  ≪list.type.name≫s; x0
    ≪convertSymbol("implies")≫ x0
      ≪list.type.name≫s; y
        ≪convertSymbol("implies")≫ y
          ≪list.type.name≫s; z
            ≪convertSymbol("implies")≫ z
              ≪list.type.name≫s; y0
                ≪convertSymbol
                  ("implies")≫ y0
                    ≪list.type.name≫s;
z0 ≪convertSymbol("implies")≫ z0
  ≪list.type.name≫s; x1
    ≪convertSymbol("implies")≫ x1
      ≪list.type.name≫s; y1
        ≪convertSymbol("implies")≫ y1
          ≪list.type.name≫s; z1
            ≪convertSymbol("implies")≫ z1
              ≪list.type.name≫s; x2
                ≪convertSymbol
                  ("implies")≫ x2
                    ≪list.type.name≫s;
y2 ≪convertSymbol("implies")≫ y2
  ≪list.type.name≫s; z2
    ≪convertSymbol("implies")≫ z2
      ≪list.type.name≫s;
end actualize
≪ENDFILE≫
```

```
«ENDFOREACH»
«ENDIF»
«ENDDEFINE»

«DEFINE generateHashDataList FOR Model»
«IF !this.getHashClasses().isEmpty»
«FOREACH this.getHashClasses()
  .calcListActualizations() AS list-»
«FILE list.type.name + "list/specification"-»
actualize list-perm with «list.type.name»
by morphism
elem «convertSymbol("implies")»
  «list.type.name»; list
    «convertSymbol("implies")»
      «list.type.name»s;
a «convertSymbol("implies")» a
  «list.type.name»; a0
    «convertSymbol("implies")» a0
      «list.type.name»; b
        «convertSymbol("implies")» b
          «list.type.name»; c
            «convertSymbol("implies")» c
              «list.type.name»;
x «convertSymbol("implies")» x
  «list.type.name»s; x0
    «convertSymbol("implies")» x0
      «list.type.name»s; y
        «convertSymbol("implies")» y
          «list.type.name»s; z
            «convertSymbol("implies")» z
              «list.type.name»s; y0
                «convertSymbol
                  ("implies")» y0
                    «list.type.name»s;
z0 «convertSymbol("implies")» z0
  «list.type.name»s; x1
    «convertSymbol("implies")» x1
      «list.type.name»s; y1
        «convertSymbol("implies")» y1
          «list.type.name»s; z1
            «convertSymbol("implies")» z1
              «list.type.name»s; x2
                «convertSymbol
                  ("implies")» x2
                    «list.type.name»s;
y2 «convertSymbol("implies")» y2
  «list.type.name»s; z2
    «convertSymbol("implies")» z2
      «list.type.name»s;
end actualize
«ENDFILE»
«ENDFOREACH»
«ENDIF»
«ENDDEFINE»
```

```
«DEFINE generateSignDataList FOR Model»
«IF !this.getSignClasses().isEmpty»
«FOREACH this.getSignClasses()
  .calcListActualizations() AS list-»
«FILE list.type.name + "list/specification"-»
actualize list-perm with «list.type.name»
by morphism
elem «convertSymbol("implies")»
  «list.type.name»; list
    «convertSymbol("implies")»
      «list.type.name»s;
a «convertSymbol("implies")» a
  «list.type.name»; a0
    «convertSymbol("implies")» a0
      «list.type.name»; b
        «convertSymbol("implies")» b
          «list.type.name»; c
            «convertSymbol("implies")» c
              «list.type.name»;
x «convertSymbol("implies")» x
  «list.type.name»s; x0
    «convertSymbol("implies")» x0
      «list.type.name»s; y
        «convertSymbol("implies")» y
          «list.type.name»s; z
            «convertSymbol("implies")» z
              «list.type.name»s; y0
                «convertSymbol
                  ("implies")» y0
                    «list.type.name»s;
z0 «convertSymbol("implies")» z0
  «list.type.name»s; x1
    «convertSymbol("implies")» x1
      «list.type.name»s; y1
        «convertSymbol("implies")» y1
          «list.type.name»s; z1
            «convertSymbol("implies")» z1
              «list.type.name»s; x2
                «convertSymbol
                  ("implies")» x2
                    «list.type.name»s;
y2 «convertSymbol("implies")» y2
  «list.type.name»s; z2
    «convertSymbol("implies")» z2
      «list.type.name»s;
end actualize
«ENDFILE»
«ENDFOREACH»
«ENDIF»
«ENDDEFINE»

«DEFINE generateAgentList FOR Model»
«FILE "agents/specification"-»
actualize list-perm with exagent
```

```
by morphism
elem
  ≪convertSymbol("implies")≫ agent; list
    ≪convertSymbol("implies")≫ agents;
a ≪convertSymbol("implies")≫ ag; a0
  ≪convertSymbol("implies")≫ ag0; b
    ≪convertSymbol("implies")≫ ag1; c
      ≪convertSymbol("implies")≫ ag2;
x ≪convertSymbol("implies")≫ agents; x0
  ≪convertSymbol("implies")≫ agents0; y
    ≪convertSymbol("implies")≫ agents1; z
      ≪convertSymbol
        ("implies")≫ agents2; y0
          ≪convertSymbol("implies")≫ agents3;
z0 ≪convertSymbol("implies")≫ agents4; x1
  ≪convertSymbol("implies")≫ agents5; y1
    ≪convertSymbol
      ("implies")≫ agents6; z1
        ≪convertSymbol
          ("implies")≫ agents7; x2
            ≪convertSymbol
              ("implies")≫ agents8;
y2 ≪convertSymbol("implies")≫ agents9; z2
  ≪convertSymbol("implies")≫ agents10;
end actualize
≪ENDFILE≫
≪ENDDEFINE≫
```

Chapter 5

Transformations for the attacker knowledge

```
import uml;
extension templates::getter;
extension templates::deployment;
extension templates::parse;

String calcAttackerReadToStrings(Model mo) :
 let p = mo.getDeploymentDiagram() :
 let ps = p.getPathsWithRead() :
 (ps.isEmpty
   ? "not eavesdrop(conn)" : "eavesdrop(conn)
     <-> " + ps.generateThreatStrings());

String calcAttackerSendToStrings(Model mo) :
 let p = mo.getDeploymentDiagram() :
 let ps = p.getPathsWithSend() :
  (ps.isEmpty ? "not send(conn)" : "send(conn)
    <-> " + ps.generateThreatStrings());

String calcAttackerSuppressToStrings(Model mo) :
 let p = mo.getDeploymentDiagram() :
 let ps = p.getPathsWithSuppress() :
  (ps.isEmpty
    ? "not suppress(conn)" : "suppress(conn)
      <-> " + ps.generateThreatStrings());

String concat(List[String] lstring) :
 (lstring.isEmpty)
   ?(""):(lstring.first() + lstring.withoutFirst
     ().concat());

List[CommunicationPath] getPathsWithRead(Package p) :
 (p.packagedElement.typeSelect
   (CommunicationPath).select(e|e.hasStereotype
     ("SecureMDD::Threat") &&
  e.getValue((e.getAppliedStereotype
     ("SecureMDD::Threat")),"read") == true &&
```

```
  !e.hasStereotype
    ("SecureMDD::TransportLayerSecurity")));

List[CommunicationPath] getPathsWithSend(Package p) :
 (p.packagedElement.typeSelect
   (CommunicationPath).select(e|e.hasStereotype
     ("SecureMDD::Threat") &&
 e.getValue((e.getAppliedStereotype
   ("SecureMDD::Threat")),"send") == true &&
 !e.hasStereotype
   ("SecureMDD::TransportLayerSecurity")));

List[CommunicationPath] getPathsWithSuppress
  (Package p) :
 (p.packagedElement.typeSelect
   (CommunicationPath).select(e|e.hasStereotype
     ("SecureMDD::Threat") &&
 e.getValue((e.getAppliedStereotype
   ("SecureMDD::Threat")),"suppress") == true &&
 !e.hasStereotype
   ("SecureMDD::TransportLayerSecurity")));

String generateThreatStrings
  (List[CommunicationPath] lcp) :
 let strs = lcp.collect(e|e.generateThreatString()) :
 ppList(strs, " or ");

String generateThreatString(CommunicationPath cp) :
  let p1 = cp.memberEnd.first() :
  let p2 = cp.memberEnd.get(1) :
  if (p1.isNavigable() && p2.isNavigable()) then
   "(conn.endpoint1.port = " + p1.portName
     () + " " + convertSymbol
       ("and") + " conn.endpoint2.port = " + p2
         .portName() + ")"
   + convertSymbol("or") +
     "(conn.endpoint1.port = " + p2.portName
       () + " " + convertSymbol
         ("and") + " conn.endpoint2.port = " + p1
           .portName() + ")"
  else if(p1.isNavigable() && !p2.isNavigable()) then
   "(conn.endpoint1.port = " + p1.portName
     () + " " + convertSymbol
       ("and") + " conn.endpoint2.port = " + p2
         .portName() + ")"
  else if(!p1.isNavigable() && p2.isNavigable()) then
    "(conn.endpoint1.port = " + p2.portName
      () + " " + convertSymbol
        ("and") + " conn.endpoint2.port = " + p1
          .portName() + ")"
  else ""
 ;

String calcPortString(Class n, CommunicationPath cp) :
 ( ((Node)n).getPort(cp).name);
```

```
String getAddKnowledgeMessage(Class c):
 let atts = c.getAttributesInCorrectOrder() :
 let vars = atts.getUniqueVariables4Properties() :
 let pref = "sub"+convertSymbol("integral")+"(" :
 let post = ", adset)" :
 let subs = getAddKnowledgeRec
   (atts, vars, pref, post) :
 let betw = " "+convertSymbol("union")+" " :
 let right = (subs.isEmpty
   ? convertSymbol("emptyset") : subs.ppList(betw)) :
 "sub-"+c.name+" : sub"+convertSymbol("integral")+"
   (amessage(" + c.getMK
     () +"), adset) = " + right + "; used for: s,ls;\n"
 ;

List[String] getAddKnowledgeSelectors
  (Class c, String what):
 let v = c.varname() :
 let atts = c.ownedAttribute :
 let pref = "sub"+convertSymbol("integral")+"(" :
 let post = ", adset)" :
 let strs = atts.collect(a|a.addKnowledge
   (v+"."+a.name, pref, post)).select(s| s != "") :
 let betw = " "+convertSymbol("union")+" " :
 let right = (strs.isEmpty
   ? convertSymbol("emptyset") : strs.ppList(betw)) :
 let ato = "a"+what.toLower() :
 "sub-"+c.name+"-"+what+" : sub"+convertSymbol
   ("integral")+"("+ato+"("+c.WRAP2(what)+"
     ("+v+")), adset) = "+right+"; used for : s,ls;\n"
 ;

List[String] getAddKnowledgeRec
  (List[Property] atts, List[String] vars, String pref
    , String post) :
 (atts.isEmpty ? {} :
 (let str = addKnowledge(atts.first(), vars.first
   (), pref, post) :
 (let rest = getAddKnowledgeRec(atts.withoutFirst
   (), vars.withoutFirst(), pref, post) :
 ( str == "" ? rest : { str }.addAll(rest)
 ))));

String addKnowledge
  (Property att, String v, String pref, String post) :
  (att.type.name
    == "Secret")      ?("{ asecret("      + v + ") }"):
  ((att.type.name
    == "Nonce")       ?("{ anonce("       + v + ") }"):
  ((att.type.name
    == "SymmKey")     ?("{ asymmkey("     + v + ") }"):
  ((att.type.name
    == "PublicKey") ?("{ apublickey("  + v + ") }"):
  ((att.type.name
```

```
    == "PrivateKey")?("{ aprivatekey(" + v + ") }"):
  ((att.hasHashedStereotype())
    ?("{ ahasheddata(" + v + ") }"):
  ((att.hasSignedStereotype())
    ?("{ asigneddata(" + v + ") }"):
  ((att.hasEncryptedStereotype())
    ?("{ aencdata("     + v + ") }"):
  ((att.hasEncryptedAsymmStereotype())
    ?("{ aencdataasymm(" + v + ") }"):
  ((!(att.type.name == "Number") &&
    !(att.type.name == "String") &&
    !(att.type.name == "Boolean")) ?
      (pref+"adata(" + att.type.WRAP2("data")+"
        (" + v + "))"+post):
      (""))))))))));

String toAttackerData(Property att, String v) :
  (att.type.name
    == "Secret")      ?("asecret("      + v + ")"):
  ((att.type.name
    == "Nonce")       ?("anonce("       + v + ")"):
  ((att.type.name
    == "SymmKey")     ?("asymmkey("     + v + ")"):
  ((att.type.name
    == "PublicKey") ?("apublickey("  + v + ")"):
  ((att.type.name
    == "PrivateKey")?("aprivatekey(" + v + ")"):
  ((att.hasHashedStereotype())
    ?("ahasheddata(" + v + ")"):
  ((att.hasSignedStereotype())
    ?("asigneddata(" + v + ")"):
  ((att.hasEncryptedStereotype())
    ?("aencdata("     + v + ")"):
  ((att.hasEncryptedAsymmStereotype())
    ?("aencdataasymm(" + v + ")"):
  ((!(att.type.name == "Number") &&
    !(att.type.name == "String") &&
    !(att.type.name == "Boolean")) ?
      ("adata(" + att.type.WRAP2("data")+"
        (" + v + "))"):
      (""))))))))));

String calculateGenerateData(Class c) :
 calculateGenerateData4data(c) +
 calculateGenerateData4SignData(c) +
 calculateGenerateData4PlainData(c) +
 calculateGenerateData4HashData(c) ;

String calculateGenerateData4data(Class c) :
     c.name.toFirstLower
       () + "4data : adset " + convertSymbol("grgr") +
         " adata(" + c.WRAP2("data") + "(" + c.getMK
           () +"))" + calculateGenerateCondition
             (c) + "; used for: s,ls; \n"
```

```
    ;

String calculateGenerateData4SignData(Class c) :
 (c.hasSignDataStereotype() ?
     c.name.toFirstLower
       () + "4SignData : adset " + convertSymbol
         ("grgr") +
         " asigndata(" + c.WRAP2("SignData") + "
           (" + c.getMK
             () +"))" + calculateGenerateCondition
               (c) + "; used for: s,ls; \n" :
     "");

String calculateGenerateData4PlainData(Class c) :
 (c.hasPlainDataStereotype() ?
     c.name.toFirstLower
       () + "4PlainData : adset " + convertSymbol
         ("grgr") +
         " aplaindata(" + c.WRAP2("PlainData") + "
           (" + c.getMK
             () +"))" + calculateGenerateCondition
               (c) + "; used for: s,ls; \n" :
     "");

String calculateGenerateData4HashData(Class c) :
 (c.hasHashDataStereotype() ?
     c.name.toFirstLower
       () + "4HashData : adset " + convertSymbol
         ("grgr") +
         " ahashdata(" + c.WRAP2("HashData") + "
           (" + c.getMK
             () +"))" + calculateGenerateCondition
               (c) + "; used for: s,ls; \n" :
     "");

String calculateGenerateMessage(Class c) :
 "generate-"+c.name+" : adset "+convertSymbol
   ("grgr")+" amessage(" + c.getMK
     () +")" + calculateGenerateCondition
       (c) + "; used for: s,ls;\n";

String calculateGenerateCondition(Class c) :
 let atts = c.getAttributesInCorrectOrder() :
  (atts.forAll(a | a.type.isPrimitiveType()) ? "" :
  (" " + convertSymbol
    ("equivalent") + " " + atts
      .calculateGenerateCondition2
        (atts.getUniqueVariables4Properties(), c))
 );

String calculateGenerateCondition2
  (List[Property] lp, List[String] vars, Class c) :
    let strs = lp.calculateGenerateCondition2rec
      (vars, c, {}) :
```

```
    let goodstrs = strs.reject(s | s == "") :
    ppList(goodstrs, " and ");

List[String] calculateGenerateCondition2rec
  (List[Property] lp, List[String] vars, Class c
    , List[String] done) :
 (lp.isEmpty ? done :
 (let propstr = (lp.first().type.isPrimitiveType()
   ? "" :
                    "adset " + convertSymbol
                      ("grgr") + " " + (lp.first
                        ().toAttackerData(vars.first
                          ()))) :
 lp.withoutFirst().calculateGenerateCondition2rec
   (vars.withoutFirst(), c, (List[String])done.add
     (propstr))
 ));

String generateIntInAxiom(Class c) :
 let nam = "int-in-"+c.name+" : " :
 let fma = " j "+convertSymbol("in") + " " + c.getMK
   () + " " :
 let rest = "; used for : s,ls;\n" :
 let atts = c.getAttributesInCorrectOrder() :
 let vars = atts.getUniqueVariables4Properties() :
 let subs = intInAxiomRec(atts, vars) :
 (subs.isEmpty
   ? nam + convertSymbol("not") + fma + rest :
 (subs.size
   == 1
     ? nam + fma + convertSymbol
       ("equivalent") + " " + subs.first() + rest :
 nam + fma + convertSymbol("equivalent") + " not not
   (" + subs.ppList(" or ") + ")" + rest
 ))
 ;

String generatePlainDataIntInAxiom(Class c) :
 let nam = "int-in-"+c.name+"-PlainData : " :
 let fma = "j "+convertSymbol("in") + " " + c.WRAP2
   ("PlainData") + "(" + varname(c.name) + ") " :
 let right = convertSymbol
   ("equivalent") + " j "+convertSymbol
     ("in") + " " + varname(c.name) :
 let rest = "; used for : s,ls;\n" :
 nam + fma + right + rest
 ;

List[String] intInAxiomRec
  (List[Property] atts, List[String] vars) :
 (atts.isEmpty ? {} :
 (let su = intInAxiom(atts.first(), vars.first()) :
 (let rest = intInAxiomRec(atts.withoutFirst
   (), vars.withoutFirst()) :
```

```
 (su == "" ? rest : { su }.addAll(rest)
 ))));

String intInAxiom(Property p, String v) :
 (p.type.name == "Number" ? "j = " + v :
 (p.isCryptoAtt() ? "" :
 (p.isPrimitiveType() ? "" :
 "j "+convertSymbol("in") + " " + v
 )));

String generateConstantsDiffAxioms
  (List[Class] classes) :
 (let atts = classes.simpleConstantAttributes() :
  atts.generateConstantsDiffs());

List[Property] simpleConstantAttributes
  (List[Class] classes) :
 (classes.size == 0 ? {} :
 (classes.first().ownedAttribute.select
   (a | a.defaultValue == null)
   .addAll(classes.withoutFirst
     ().simpleConstantAttributes())
   ));

String generateConstantsDiffs(List[Property] atts) :
 (atts.size < 2 ? "" :
  generateConstantsDiffsOne(atts.first
    (), atts.withoutFirst()) +
     atts.withoutFirst().generateConstantsDiffs() );

String generateConstantsDiffsOne
  (Property a, List[Property] atts) :
 (atts.size == 0 ? "" :
  "diff : " + a.name + " \\neq " + atts.first
    ().name + "; used for : s,ls,generated;\n" +
  generateConstantsDiffsOne(a, atts.withoutFirst())
  );
```

Chapter 6

Transformations for the attacker abilities

```
«IMPORT uml»
«EXTENSION templates::getter»
«EXTENSION templates::attacker»
«EXTENSION templates::deployment»

«DEFINE Root FOR uml::Model»
«EXPAND generateFromClassDiagram»
«IF this.getDeploymentDiagram() != null»
 «EXPAND generateAttackerAbilities»
«ENDIF»
«ENDDEFINE»

«DEFINE generateFromClassDiagram FOR Model»
«EXPAND generateAttackerData FOR this»
«EXPAND generateAttackerDataSet FOR this»
«EXPAND generateAttackerknowledge FOR this»
«EXPAND generateAnalyze FOR this»
«EXPAND generateAddAttackerdata FOR this»
«EXPAND generateGenerate FOR this»
«ENDDEFINE»

«DEFINE generateAttackerAbilities FOR Model»
«FILE "AttackerAbilities/specification"»
enrich connectionPreds with
predicates
  eavesdrop : connection;
  suppress : connection;
  send : connection;
  attacker-can-read : connection;
  attacker-can-send : connections;
  attacker-can-send : endpoint
    «convertSymbol("times")» connections;
  attacker-can-suppress : endpoint
    «convertSymbol("times")» connections;
  attacker-can-suppress : connection
    «convertSymbol("times")» connections;
```

```
  attacker-can-suppress : connections;
axioms
      attacker-can-read
      : ≪convertSymbol("follows")≫
          attacker-can-read(conn)
          ≪convertSymbol
            ("equivalent")≫ is-endpoint
              (attacker, conn)
                ≪convertSymbol
                  ("or")≫ eavesdrop(conn)
      ;
      attacker-can-send-conn
      : ≪convertSymbol("follows")≫
          attacker-can-send(connections)
        ≪convertSymbol("equivalent")≫ (
          ≪convertSymbol("exists")≫ endp
            . attacker-can-send(endp, connections))
      ;
      attacker-can-send
      : ≪convertSymbol("follows")≫
          attacker-can-send(endp, connections)
        ≪convertSymbol("equivalent")≫
          ≪convertSymbol("not")≫ is_attacker
            (endp .agent)
          ≪convertSymbol("and")≫ (
            ≪convertSymbol("exists")≫ conn.
                conn
                  ≪convertSymbol
                    ("in")≫ connections
                      ≪convertSymbol
                        ("and")≫ is-endpoint
                          (endp, conn)
                            ≪convertSymbol
                              ("and")≫ send(conn))
      ;
      attacker-can-suppress-endpoint
      : ≪convertSymbol("follows")≫
          attacker-can-suppress(endp, connections)
        ≪convertSymbol("equivalent")≫ (
          ≪convertSymbol("exists")≫ conn.
              conn
                ≪convertSymbol
                  ("in")≫ connections
                    ≪convertSymbol
                      ("and")≫ is-endpoint
                        (endp, conn)
                          ≪convertSymbol
                            ("and")≫ suppress
                              (conn))
      ;
      attacker-can-suppress-conn
      : ≪convertSymbol("follows")≫
          attacker-can-suppress(conn, connections)
        ≪convertSymbol
```

```
            ("equivalent")≫ conn
              ≪convertSymbol
                ("in")≫ connections
                  ≪convertSymbol
                    ("and")≫ suppress(conn)
    ;
        attacker-can-suppress-any
        : ≪convertSymbol("follows")≫
            attacker-can-suppress(connections)
          ≪convertSymbol("equivalent")≫ (
            ≪convertSymbol("exists")≫ endp
              . attacker-can-suppress(endp, connections))
        ;
        eavesdrop:
          ≪convertSymbol("follows")≫
            ≪this
              .calcAttackerReadToStrings
                ()≫; used for : s, ls;
        suppress :
          ≪convertSymbol("follows")≫
            ≪this
              .calcAttackerSuppressToStrings
                ()≫; used for : s, ls;
        send :  ≪convertSymbol("follows")≫
          ≪this
            .calcAttackerSendToStrings
              ()≫; used for : s, ls;
end enrich
≪ENDFILE≫
≪ENDDEFINE≫

≪DEFINE generateAttackerData FOR Model≫
≪FILE "attackerdata/specification"-≫
data specification
 using messagefuns
attackerdata = amessage(.
  .message : message) with is_amessage
≪IF this.usesSymmKey()-≫
      | asymmkey(. .key : SymmKey) with is_asymmkey
≪ENDIF-≫
≪IF this.usesASymmKey()-≫
      | apublickey(.
        .key : PublicKey) with is_apublickey
      | aprivatekey(.
        .key : PrivateKey) with is_aprivatekey
≪ENDIF-≫
≪IF this.existsSecret() || this.usesSymmKey()-≫
      | asecret(. .secret : Secret) with is_asecret
≪ENDIF-≫
≪IF this.existsNonce() || this.usesSymmKey()-≫
      | anonce(. .nonce : Nonce) with is_anonce
≪ENDIF-≫
≪IF ! this.getNormalClasses().isEmpty-≫
      | adata(. .data : data) with is_adata
```

```
«ENDIF-»
«IF !this.getPlainClasses().isEmpty-»
     | aplaindata(.
       .plaindata : PlainData) with is_aplaindata
«ENDIF-»
«IF !this.getHashClasses().isEmpty-»
     | ahashdata(.
       .hashdata : HashData) with is_ahashdata
«ENDIF-»
«IF !this.getSignClasses().isEmpty-»
     | asigndata(.
       .signdata : SignData) with is_asigndata
«ENDIF-»
«IF this.usesEncryption()-»
     | aencdata(. .encdata : EncData) with is_aencdata
«ENDIF-»
«IF this.usesAsymmEncryption()-»
     | aencdataasymm(.
       .encdataasymm : EncDataAsymm)
       with is_aencdataasymm
«ENDIF-»
«IF this.usesHashing()-»
     | ahasheddata(.
       .hasheddata : HashedData) with is_ahasheddata
«ENDIF-»
«IF this.usesSigning()-»
     | asigneddata(.
       .signeddata : SignedData) with is_asigneddata
«ENDIF-»
   ;
variables
ad : attackerdata;
end data specification
«ENDFILE»
«ENDDEFINE»

«DEFINE generateAttackerDataSet FOR Model»
«FILE "attackerdataset/specification"-»
actualize set-union with attackerdata by morphism
  elem
    «convertSymbol
      ("implies")» attackerdata ; set
        «convertSymbol
          ("implies")» attackerdataset ; a
            «convertSymbol
              ("implies")» ad ; b
                «convertSymbol
                  ("implies")» ad0; c
                    «convertSymbol
                      ("implies")» ad1;
  s «convertSymbol("implies")» adset; s0
    «convertSymbol("implies")» adset0; s1
      «convertSymbol
        ("implies")» adset1; s2
```

```
          ≪convertSymbol("implies")≫ adset2;
end actualize
≪ENDFILE≫
≪ENDDEFINE≫

≪DEFINE generateAttackerknowledge FOR Model≫
≪FILE "submessages/specification"–≫
≪LET "sub"+convertSymbol("integral") AS subi≫
enrich attackerdataset with
functions
    ≪subi≫ : attackerdata
      ≪convertSymbol
        ("times")≫ attackerdataset
          ≪convertSymbol
            ("implies"
              )≫ attackerdataset; comment
                : known direct subdata;
≪IF this.usesEncryption() || this
  .usesAsymmEncryption() || this.usesSymmKey() || this
    .usesASymmKey()–≫predicates≪ENDIF≫
≪IF this.usesEncryption()–≫
    can_decrypt : EncData
      ≪convertSymbol("times")≫ attackerdataset;
≪ENDIF–≫
≪IF this.usesAsymmEncryption()–≫
    can_decrypt : EncDataAsymm
      ≪convertSymbol("times")≫ attackerdataset;
≪ENDIF–≫
≪IF this.usesSymmKey() || this.usesASymmKey()–≫
 no-keys : attackerdataset;
≪ENDIF–≫
axioms
(: messages :)
≪FOREACH this
  .getAllMessageAndUserMessageClasses()
    AS msg–≫
      ≪msg.getAddKnowledgeMessage()–≫
        ≪ENDFOREACH–≫
(: content of messages :)
≪FOREACH this.getNormalClasses() AS c–≫
  ≪c.getAddKnowledgeSelectors("data")–≫
    ≪ENDFOREACH–≫
(: crypto data :)
≪FOREACH this.getPlainClasses() AS c–≫
  ≪c
    .getAddKnowledgeSelectors("PlainData")–≫
      ≪ENDFOREACH–≫
≪FOREACH this.getHashClasses() AS c–≫
  ≪c
    .getAddKnowledgeSelectors("HashData")–≫
      ≪ENDFOREACH–≫
≪FOREACH this.getSignClasses() AS c–≫
  ≪c
    .getAddKnowledgeSelectors("SignData")–≫
```

```
      ≪ENDFOREACH–≫
(: predefined/special data :)
≪IF this.usesSymmKey()–≫
sub–SymmKey : sub
  ≪convertSymbol("integral")≫(asymmkey(
    ≪varname("SymmKey")≫), adset) =
      ≪convertSymbol
        ("emptyset")≫; used for : s,ls;
≪ENDIF–≫
≪IF this.usesASymmKey()–≫
sub–PublicKey : sub
  ≪convertSymbol("integral")≫(apublickey(
    ≪varname("PublicKey")≫), adset) =
      ≪convertSymbol
        ("emptyset")≫; used for : s,ls;
sub–PrivateKey : sub
  ≪convertSymbol("integral")≫(aprivatekey(
    ≪varname("PrivateKey")≫), adset) =
      ≪convertSymbol
        ("emptyset")≫; used for : s,ls;
≪ENDIF–≫
≪IF this.usesEncryption()–≫
can_decrypt : can_decrypt(
  ≪varname("EncData")≫, adset)
    ≪convertSymbol("equivalent")≫
      ≪convertSymbol("exists")≫
        ≪varname("SymmKey")≫. asymmkey(
          ≪varname("SymmKey")≫)
            ≪convertSymbol("in")≫ adset
              ≪convertSymbol
                ("and")≫ can_decrypt(
                  ≪varname("SymmKey")≫,
                    ≪varname("EncData")≫);
subEncData–yes : can_decrypt(
  ≪varname("EncData")≫, adset)
    ≪convertSymbol("implies")≫ sub
      ≪convertSymbol("integral")≫(aencdata
        (≪varname
            ("EncData")≫), adset) = sub
              ≪convertSymbol("integral")≫
                (aplaindata(
                  ≪varname("EncData")≫
                    .plain), adset); used for : s,ls;
subEncData–no  :
  ≪convertSymbol("not")≫ can_decrypt(
    ≪varname("EncData")≫, adset)
      ≪convertSymbol("implies")≫ sub
        ≪convertSymbol("integral")≫(aencdata
          (≪varname
              ("EncData")≫), adset) =
                ≪convertSymbol
                  ("emptyset")≫; used for : s,ls;
≪ENDIF–≫
≪IF this.usesAsymmEncryption()–≫
```

```
can_decrypt_asymm : can_decrypt(
  ≪varname("EncDataAsymm")≫, adset)
    ≪convertSymbol("equivalent")≫
      ≪convertSymbol("exists")≫
        ≪varname("PrivateKey")≫
          . aprivatekey(
            ≪varname("PrivateKey")≫)
              ≪convertSymbol("in")≫ adset
                ≪convertSymbol
                  ("and")≫ can_decrypt(
                    ≪varname
                      ("PrivateKey")≫,
                        ≪varname
                          ("EncDataAsymm")≫);
subEncDataAsymm-yes : can_decrypt(
  ≪varname("EncDataAsymm")≫, adset)
    ≪convertSymbol("implies")≫ sub
      ≪convertSymbol("integral")≫
        (aencdataasymm(
          ≪varname
            ("EncDataAsymm")≫), adset) = sub
              ≪convertSymbol("integral")≫
                (aplaindata(
                  ≪varname("EncDataAsymm")≫
                    .plain), adset); used for : s,ls;
subEncDataAsymm-no  :
  ≪convertSymbol("not")≫ can_decrypt(
    ≪varname("EncDataAsymm")≫, adset)
      ≪convertSymbol("implies")≫ sub
        ≪convertSymbol("integral")≫
          (aencdataasymm(
            ≪varname
              ("EncDataAsymm")≫), adset) =
                ≪convertSymbol
                  ("emptyset")≫; used for : s,ls;
≪ENDIF-≫
≪IF this.existsSecret() || this.usesSymmKey()-≫
sub-Secret : sub
  ≪convertSymbol("integral")≫(asecret(
    ≪varname("Secret")≫), adset) =
      ≪convertSymbol
        ("emptyset")≫; used for : s,ls;
≪ENDIF-≫
≪IF this.existsNonce() || this.usesSymmKey()-≫
sub-Nonce : sub
  ≪convertSymbol("integral")≫(anonce(
    ≪varname("Nonce")≫), adset) =
      ≪convertSymbol
        ("emptyset")≫; used for : s,ls;
≪ENDIF-≫
≪IF this.usesHashing()-≫
sub-hashed : sub
  ≪convertSymbol("integral")≫(ahasheddata(
    ≪varname("HashedData")≫), adset) =
```

```
      ≪convertSymbol
        ("emptyset")≫; used for : s,ls;
≪ENDIF–≫
≪IF this.usesSigning()–≫
sub–signed : sub
  ≪convertSymbol("integral")≫(asigneddata(
    ≪varname("SignedData")≫), adset) =
      ≪convertSymbol
        ("emptyset")≫; used for : s,ls;
≪ENDIF–≫
≪IF this.usesSymmKey() || this.usesASymmKey()–≫
(: other axioms :)
 no–keys–def : no–keys(adset) <–>
   ≪convertSymbol("all")≫ ad. ad
     ≪convertSymbol("in")≫ adset –>
       ≪IF this.usesSymmKey() && this.uses
         ASymmKey()–≫
 not is_asymmkey(ad) and not is_aprivatekey(ad);
   ≪ELSEIF this
     .usesSymmKey()≫not is_asymmkey(ad);
       ≪ELSE≫not is_aprivatekey(ad);
         ≪ENDIF–≫
≪ENDIF–≫
(: generated theorems :)
≪IF this.onlyTrivialUserClasses()≫
user–message–empty :
  is_user_message(msg) –>
  sub\integral(amessage(msg), adset) = \emptyset;
  used for : s,ls,generated;
≪ENDIF≫
≪IF this.usesASymmKey()–≫
   subint–publickey :
   ≪subi≫(ad, adset ++ apublickey(a_PublicKey))
   = ≪subi≫(ad, adset) ; used for : generated, s, ls ;
≪ENDIF≫
≪IF this.usesAsymmEncryption()–≫
   subint–encdataasymm :
   ≪subi≫(ad, adset ++ aencdataasymm(a_EncDataAsymm))
   = ≪subi≫(ad, adset); used for : generated, s, ls ;
≪ENDIF≫
≪IF this.usesEncryption()–≫
   subint–encdatasymm :
   ≪subi≫(ad, adset ++ aencdata(a_EncData))
   = ≪subi≫(ad, adset) ; used for : generated, s, ls ;
≪ENDIF≫
≪IF this.existsNonce() || this.usesSymmKey()–≫
   subint–nonce :
   ≪subi≫(ad, adset ++ anonce(a_Nonce))
   = ≪subi≫(ad, adset) ; used for : generated, s, ls ;
≪ENDIF≫
≪IF this.existsSecret() || this.usesSymmKey()–≫
   subint–secret :
   ≪subi≫(ad, adset ++ asecret(a_Secret))
   = ≪subi≫(ad, adset); used for : generated, s, ls ;
```

```
≪ENDIF≫
≪IF this.usesHashing()−≫
   subint−hasheddata :
   ≪subi≫(ad, adset ++ ahasheddata(a_HashedData))
   = ≪subi≫(ad, adset); used for : generated, s, ls ;
≪ENDIF≫
≪IF this.usesSigning()−≫
   subint−signeddata :
   ≪subi≫(ad, adset ++ asigneddata(a_SignedData))
   = ≪subi≫(ad, adset); used for : generated, s, ls ;
≪ENDIF≫
≪IF this.usesAsymmEncryption()−≫
   subint−no−keys−encdataasymm :
   no−keys(adset) −>
   ≪subi≫(aencdataasymm(a_EncDataAsymm), adset)
   = \emptyset ;
   used for : generated, s, ls ;
≪ENDIF≫
≪IF this.usesEncryption()−≫
   subint−no−keys−encdatasymm :
   no−keys(adset) −>
   ≪subi≫(aencdata(a_EncData), adset) = \emptyset ;
   used for : generated, s, ls ;
≪ENDIF≫
≪IF this.usesSymmKey() || this.usesASymmKey()−≫
≪IF this.usesAsymmEncryption()−≫
   no−keys−encdataasymm :
   no−keys(adset ++ aencdataasymm(a_EncDataAsymm))
   <−> no−keys(adset); used for : generated, s, ls ;
≪ENDIF≫
≪IF this.usesEncryption()−≫
   no−keys−encdatasymm :
   no−keys(adset ++ aencdata(a_EncData))
   <−> no−keys(adset); used for : generated, s, ls ;
≪ENDIF≫
≪IF this.usesASymmKey()−≫
   no−keys−publickey :
   no−keys(adset ++ apublickey(a_PublicKey))
   <−> no−keys(adset); used for : generated, s, ls ;
≪ENDIF≫
≪IF this.existsNonce() || this.usesSymmKey()−≫
   no−keys−nonce :
   no−keys(adset ++ anonce(a_Nonce))
   <−> no−keys(adset); used for : generated, s, ls ;
≪ENDIF≫
≪IF this.existsSecret() || this.usesSymmKey()−≫
   no−keys−secret :
   no−keys(adset ++ asecret(a_Secret))
   <−> no−keys(adset); used for : generated, s, ls ;
≪ENDIF≫
≪IF this.usesSigning()−≫
   no−keys−signeddata :
   no−keys(adset ++ asigneddata(a_SignedData))
   <−> no−keys(adset) ; used for : generated, s, ls ;
```

```
«ENDIF»
«IF this.usesHashing()-»
   no-keys-hasheddata :
   no-keys(adset ++ ahasheddata(a_HashedData))
   <-> no-keys(adset) ; used for : generated, s, ls ;
«ENDIF»
«ENDIF»
end enrich
«ENDLET»
«ENDFILE»
«ENDDEFINE»

«DEFINE generateAnalyze FOR Model-»
«FILE "Analyze/specification"-»
«LET "max"+convertSymbol("integral") AS maxi»
«LET convertSymbol("integral") AS integ»
enrich submessages with
functions
 «integ» . : attackerdataset
   «convertSymbol("implies")» attackerdataset;
predicates
 «maxi» . : attackerdataset;
axioms
 add-max: max
   «convertSymbol("integral")» adset
     «convertSymbol("implies")»
       «convertSymbol
         ("integral"
           )» adset = adset; used for: s, ls;
 add-rec: ad «convertSymbol("in")» adset
   «convertSymbol("implies")»
     «convertSymbol("integral")» adset =
       «convertSymbol("integral")» (adset
         «convertSymbol("union")» sub
           «convertSymbol("integral")»
             (ad, adset));
 maxknown: max
   «convertSymbol("integral")» adset
     «convertSymbol("equivalent")» (
       «convertSymbol("all")» ad. ad
         «convertSymbol("in")» adset
           «convertSymbol("implies")» (sub
             «convertSymbol("integral")»
               (ad, adset)
                 «convertSymbol
                   ("subseteq")» adset));
(: generated theorems :)
 add-subset : adset
   «convertSymbol("subseteq")»
     «convertSymbol
       ("integral"
         )» adset; used for : s, ls, generated;
 integ-is-max : «maxi»
   «integ» adset; used for : s, ls, generated;
```

```
≪IF this.existsSecret() || this
  .usesSymmKey()−≫max−known−secret−01 :
    ≪maxi≫ adset −>
    ≪maxi≫ (adset ++ asecret(a_Secret)) ;
    used for : generated, s, ls ;

    max−known−secret :
    ≪maxi≫ adset
    −> ≪integ≫ (adset ++ asecret(a_Secret))
       = adset ++ asecret(a_Secret) ;
    used for : generated, s, ls ;
≪ENDIF≫
≪IF this.existsNonce() || this.usesSymmKey()−≫
   max−known−nonce :
   ≪maxi≫ adset
   −> ≪integ≫ (adset ++ anonce(a_Nonce))
      = adset ++ anonce(a_Nonce) ;
   used for : generated, s, ls ;

   max−known−nonce−01 :
   ≪maxi≫ adset
   −> ≪maxi≫ (adset ++ anonce(a_Nonce));
   used for : generated, s, ls ;
≪ENDIF≫
≪IF this.usesSigning()−≫
   max−known−signeddata :
   ≪maxi≫ adset
   −> ≪integ≫ (adset ++ asigneddata(a_SignedData))
      = adset ++ asigneddata(a_SignedData) ;
   used for : generated, s, ls ;

   max−known−signeddata−01 :
   ≪maxi≫ adset
   −> ≪maxi≫ (adset ++ asigneddata(a_SignedData));
   used for : generated, s, ls ;
≪ENDIF≫
≪IF this.usesHashing()−≫
   max−known−hasheddata :
   ≪maxi≫ adset
   −> ≪integ≫ (adset ++ ahasheddata(a_HashedData))
      = adset ++ ahasheddata(a_HashedData) ;
   used for : generated, s, ls ;

   max−known−hasheddata−01 :
   ≪maxi≫ adset
   −> ≪maxi≫ (adset ++ ahasheddata(a_HashedData)) ;
   used for : generated, s, ls ;
≪ENDIF≫
≪IF this.usesSymmKey() || this.usesASymmKey()−≫
≪IF this.usesAsymmEncryption()−≫
   max−known−encdataasymm :
   no−keys(adset) and ≪maxi≫ adset
   −> ≪integ≫ (adset ++ aencdataasymm(a_EncDataAsymm))
      = adset ++ aencdataasymm(a_EncDataAsymm) ;
```

```
   used for : generated, s, ls ;

   max-known-encdataasymm-01 :
   no-keys(adset) and «maxi» adset
   -> «maxi» (adset ++ aencdataasymm(a_EncDataAsymm)) ;
   used for : generated, s, ls ;
«ENDIF»
«IF this.usesEncryption()-»
   max-known-encdatasymm :
   no-keys(adset) and «maxi» adset
   -> «integ» (adset ++ aencdata(a_EncData))
      = adset ++ aencdata(a_EncData) ;
   used for : generated, s, ls ;

   max-known-encdatasymm-01 :
   no-keys(adset) and «maxi» adset
   -> «maxi» (adset ++ aencdata(a_EncData)) ;
   used for : generated, s, ls ;
«ENDIF»
«ENDIF»
end enrich
«ENDLET»«ENDLET»
«ENDFILE-»
«ENDDEFINE»

«DEFINE generateAddAttackerdata FOR Model-»
«FILE "AddAttackerdata/specification"-»
enrich Analyze,messagelist with
functions
 .
   «convertSymbol("integral")»
     + . : attackerdataset
       «convertSymbol("times")» message
         «convertSymbol
           ("implies")» attackerdataset;
 .
   «convertSymbol("integral")»
     + . : attackerdataset
       «convertSymbol
         ("times")» messagelist
           «convertSymbol
             ("implies")» attackerdataset;
axioms
 «IF this.usesEncryption() || this
   .usesAsymmEncryption()-»
 add-one: adset
   «convertSymbol("integral")»+ msg =
     «convertSymbol("integral")» (adset
       «convertSymbol("union")» sub
         «convertSymbol("integral")»(amessage
           (msg),adset)); used for: s,ls;
 «ELSE-»
 add-one: adset
   «convertSymbol("integral")»
```

```
    + msg = adset
      ≪convertSymbol("union")≫ sub
        ≪convertSymbol("integral")≫(amessage
          (msg), adset); used for: s,ls;
≪ENDIF–≫
add–empty: adset
  ≪convertSymbol("integral")≫+ [] =
    ≪convertSymbol
      ("integral")≫ adset; used for: ls;
adds–rec: adset
  ≪convertSymbol("integral")≫+ (msg '
    + msgs) = (adset
      ≪convertSymbol("union")≫ sub
        ≪convertSymbol("integral")≫(amessage
          (msg),adset))
            ≪convertSymbol("integral")≫
              + msgs; used for : ls;
end enrich
≪ENDFILE–≫
≪ENDDEFINE≫

≪DEFINE generateGenerate FOR Model≫
≪FILE "Generate/specification"≫
enrich AddAttackerdata with
predicates
 .
  ≪convertSymbol("grgr")≫
    . : attackerdataset
      ≪convertSymbol("times")≫ attackerdata;
axioms
≪IF this.existsNonce() || this.usesSymmKey()–≫
nonce : adset
  ≪convertSymbol("grgr")≫ anonce(
    ≪varname("Nonce")≫)
      ≪convertSymbol("equivalent")≫ anonce
        (≪varname("Nonce")≫)
          ≪convertSymbol
            ("in")≫ adset; used for: s,ls;
≪ENDIF–≫
≪IF this.existsSecret() || this.usesSymmKey()–≫
secret : adset
  ≪convertSymbol("grgr")≫ asecret(
    ≪varname("Secret")≫)
      ≪convertSymbol("equivalent")≫ asecret
        (≪varname("Secret")≫)
          ≪convertSymbol
            ("in")≫ adset; used for: s,ls;
≪ENDIF–≫
≪IF this.usesSymmKey()–≫
symmkey : adset
  ≪convertSymbol("grgr")≫ asymmkey(
    ≪varname("SymmKey")≫)
      ≪convertSymbol("equivalent")≫ asymmkey
        (≪varname("SymmKey")≫)
```

```
          ≪convertSymbol
            ("in")≫ adset; used for: s,ls;
≪ENDIF–≫
≪IF this.usesASymmKey()–≫
publickey : adset
  ≪convertSymbol("grgr")≫ apublickey(
    ≪varname("PublicKey")≫); used for: s,ls;
privatekey : adset
  ≪convertSymbol("grgr")≫ aprivatekey(
    ≪varname("PrivateKey")≫)
      ≪convertSymbol
        ("equivalent")≫ aprivatekey(
          ≪varname("PrivateKey")≫)
            ≪convertSymbol
              ("in")≫ adset; used for: s,ls;
≪ENDIF–≫
≪IF this.usesHashing()–≫
hasheddata : adset
≪convertSymbol("grgr")≫ ahasheddata(
≪varname("HashedData")≫)
≪convertSymbol("equivalent")≫
ahasheddata(≪varname("HashedData")≫)
≪convertSymbol("in")≫ adset
≪convertSymbol("or")≫ adset
≪convertSymbol("grgr")≫
ahashdata(≪varname("HashedData")≫.hash);
used for: s,ls;
≪ENDIF–≫
≪IF this.usesEncryption()–≫
encdata : adset
≪convertSymbol("grgr")≫
aencdata(mkEncData(≪varname("SymmKey")≫,
                   ≪varname("PlainData")≫))
≪convertSymbol("equivalent")≫
aencdata(mkEncData(≪varname("SymmKey")≫,
                     ≪varname("PlainData")≫))
≪convertSymbol("in")≫
adset ≪convertSymbol("or")≫
(adset≪convertSymbol("grgr")≫
asymmkey(≪varname("SymmKey")≫)
≪convertSymbol("and")≫ adset
≪convertSymbol("grgr")≫
aplaindata(≪varname("PlainData")≫));
used for : s,ls;
≪ENDIF–≫
≪IF this.usesAsymmEncryption()–≫
encdataasymm : adset
≪convertSymbol("grgr")≫
aencdataasymm(mkEncDataAsymm(≪varname("PublicKey")≫,
                             ≪varname("PlainData")≫))
≪convertSymbol("equivalent")≫
aencdataasymm(mkEncDataAsymm(≪varname("PublicKey")≫,
                             ≪varname("PlainData")≫))
≪convertSymbol("in")≫
```

```
adset «convertSymbol("or")»
(adset «convertSymbol("grgr")»
apublickey( «varname("PublicKey")»)
«convertSymbol("and")»
adset «convertSymbol("grgr")»
aplaindata(«varname("PlainData")»));
used for : s,ls;
«ENDIF-»
«IF this.usesSigning()-»
signeddata : adset
«convertSymbol("grgr")»
asigneddata(mkSignedData(«varname("PrivateKey")»,
                         «varname("SignData")»))
«convertSymbol("equivalent")»
asigneddata(mkSignedData(«varname("PrivateKey")»,
                         «varname("SignData")»))
«convertSymbol("in")»
adset «convertSymbol("or")»
( adset «convertSymbol("grgr")»
aprivatekey(«varname("PrivateKey")»)
«convertSymbol("and")»
adset «convertSymbol("grgr")»
asigndata(«varname("SignData")»));
used for : s,ls;
«ENDIF-»
(: messages :)
«FOREACH this
  .getAllMessageAndUserMessageClasses() AS msg-»
«calculateGenerateMessage(msg)-»
«ENDFOREACH-»
(: normal, hash, plain, sign classes :)
«FOREACH this.getNormalClasses() AS c»
«calculateGenerateData(c)»
«ENDFOREACH»
(: generated axioms :)
«IF this.onlyTrivialUserClasses()»
user-messages-known :
is_user_message(msg)
-> adset \grgr amessage(msg);
used for : s,ls,generated;
«ENDIF»
generate-max-in :
ad \in adset and max
«convertSymbol("integral")» adset
-> adset \grgr ad ;
used for : generated, s, ls ;

generate-max-in :
ad \in adset and max
«convertSymbol("integral")» adset
-> adset0 «convertSymbol("union")» adset \grgr ad ;
used for : generated, s, ls ;

generate-add-message :
```

```
not adset ++ ad \grgr amessage(msg)
-> not adset \grgr amessage(msg);
used for : s,ls,generated;

generate-add :
not adset ++ ad \grgr ad0
-> not adset \grgr ad0;
used for : s,ls,generated;

generate-subset :
adset \grgr ad and adset \subseteq adset0
-> adset0 \grgr ad;
used for : generated;

generate-message :
sub≪convertSymbol("integral")≫(amessage(msg), adset)
≪convertSymbol("subseteq")≫ \emptyset ++ ad
-> adset ++ ad \grgr amessage(msg);
used for : s,ls,generated;

generate-message :
sub≪convertSymbol("integral")≫(amessage(msg), adset)
≪convertSymbol("subseteq")≫ \emptyset ++ ad ++ ad0
-> adset ++ ad ++ ad0 \grgr amessage(msg);
used for : s,ls,generated;

generate-more :
not adset ++ ad ++ ad0 \grgr ad1 -> not adset \grgr ad1;
used for : generated, s, ls ;

generate-more :
not adset ++ ad \grgr ad1
-> not adset \grgr ad1 ;
used for : generated, s, ls ;

generate-message-subset :
sub≪convertSymbol("integral")≫(amessage(msg), adset1)
≪convertSymbol("subseteq")≫
adset and adset1
≪convertSymbol("union")≫ adset
≪convertSymbol("subseteq")≫ adset0
-> adset0 \grgr amessage(msg);
used for : generated;
≪IF this.usesAsymmEncryption()-≫
generate-encdataasymm :
aencdataasymm(a_EncDataAsymm) \in adset
-> adset \grgr aencdataasymm(a_EncDataAsymm);
used for : generated, s, ls ;

generate-encdataasymm :
aencdataasymm(a_EncDataAsymm) \in adset
-> adset0≪convertSymbol("union")≫
adset \grgr aencdataasymm(a_EncDataAsymm) ;
used for : generated, s, ls ;
```

```
≪ENDIF≫
≪IF this.usesEncryption()−≫
generate−encdata : aencdata(a_EncData) \in adset
−> adset \grgr aencdata(a_EncData);
used for : generated, s, ls;

generate−encdata :
aencdata(a_EncData) \in adset
−> adset0≪convertSymbol("union")≫
adset \grgr aencdata(a_EncData) ;
used for : generated, s, ls ;
≪ENDIF≫
≪IF this.usesSigning()−≫
generate−signeddata : asigneddata(a_SignedData) \in adset
−> adset \grgr asigneddata(a_SignedData);
used for : generated, s, ls ;

generate−signeddata :
asigneddata(a_SignedData) \in adset
−> adset0 ≪convertSymbol("union")≫
adset \grgr asigneddata(a_SignedData);
used for : generated, s, ls ;
≪ENDIF≫
≪IF this.usesHashing()−≫
generate−hasheddata :
ahasheddata(a_HashedData) \in adset
−> adset \grgr ahasheddata(a_HashedData);
used for : generated, s, ls;

generate−hasheddata :
ahasheddata(a_HashedData) \in adset
−> adset0≪convertSymbol("union")≫
adset \grgr ahasheddata(a_HashedData);
used for : generated, s, ls ;
≪ENDIF≫
≪IF this.existsNonce() || this.usesSymmKey()−≫
generate−nonce :
anonce(a_Nonce) \in adset
−> adset \grgr anonce(a_Nonce);
used for : generated, s, ls ;

generate−nonce :
anonce(a_Nonce) \in adset
−> adset0 ≪convertSymbol("union")≫
adset \grgr anonce(a_Nonce);
used for : generated, s, ls ;
≪ENDIF≫
≪IF this.existsSecret() || this.usesSymmKey()−≫
generate−secret :
asecret(a_Secret) \in adset
−> adset \grgr asecret(a_Secret);
used for : generated, s, ls ;

generate−secret :
```

```
asecret(a_Secret) \in adset
-> adset0 ≪convertSymbol("union")≫
adset \grgr asecret(a_Secret);
used for : generated, s, ls ;
≪ENDIF≫
end enrich
≪ENDFILE≫
≪ENDDEFINE≫
```

Chapter 7

Transformations for security operations

```
«IMPORT uml»
«EXTENSION templates::getter»

«DEFINE Root FOR uml::Model»
«EXPAND generateFromClassDiagram FOR this»
«ENDDEFINE»

«DEFINE generateFromClassDiagram FOR Model»
«EXPAND generateSecurityOperations»
«IF this.existsSecret() || this
  .existsSymmKey()»
    «EXPAND generateSecret»«ENDIF»
«IF this.existsNonce() || this
  .usesSymmKey()»
    «EXPAND generateNonce»«ENDIF»
«IF this.usesSymmKey()»
  «EXPAND generateGenerateKey FOR this»
    «ENDIF»
«IF this.usesSymmKey() || this.usesASymmKey()»
    «EXPAND generateKeys»
    «ENDIF»
«ENDDEFINE»

«DEFINE generateSecret FOR Model»
«FILE "Secret/specification"-»
data specification
using string-data
Secret = mkSecret( . .secret : string);
variables «varname("Secret")»,
  «varname("Secret")»0,
    «varname("Secret")»1 : Secret;
end data specification
«ENDFILE»
«ENDDEFINE»

«DEFINE generateNonce FOR Model»
```

```
≪FILE "Nonce/specification"–≫
data specification
using string-data
Nonce = mkNonce( . .nonce : string);
variables ≪varname("Nonce")≫,
  ≪varname("Nonce")≫0,
    ≪varname("Nonce")≫1 : Nonce;
end data specification
≪ENDFILE≫
≪FILE "generateNonce/specification"–≫
enrich Nonce, int-pair with
predicates
. \in . : Nonce , (nat -> Nonce);
comment : The given nonce appears somewhere in the stream;
   dups : (nat -> Nonce);
comment: The stream contains duplicates;
in-rest : Nonce , (nat -> Nonce), nat;
comment : The given nonce appears somewhere in the stream
          above the given lower limit;
old : Nonce , (nat -> Nonce) , nat;
comment: The given nonce is old, i.e. does not appear
         above the given lower limit;

procedures
generateNonce#(nat ≪
               convertSymbol("implies")≫ Nonce) :
               nat ≪convertSymbol("times")≫ Nonce nonfunctional;

variables
   all-nonces, all-nonces0 : nat
            ≪convertSymbol("implies")≫ Nonce;
   next-nonce : nat;
axioms
nonce-in-rest-def :
in-rest(a_Nonce, all-nonces, next-nonce) <-> ex n
  . (n \ge next-nonce and all-nonces(n) = a_Nonce);
nonce-old-def : old
  (a_Nonce, all-nonces, next-nonce) <-> not ex n
    . (n \ge next-nonce and all-nonces(n) = a_Nonce);
nonce-in-def :
a_Nonce \in all-nonces <-> ex n
  . (a_Nonce = all-nonces(n));
nonce-dups-def :
dups(all-nonces) <-> ex n,n0
  . (n \unequal n0 and all-nonces(n) = all-nonces(n0));
old-next-nonce :
old(a_Nonce, all-nonces, next-nonce) ->
old(a_Nonce, all-nonces, next-nonce
  + 1); used for : generated, s, ls;
old-next-nonce-nodups :
not dups(all-nonces) -> old(all-nonces
  (next-nonce), all-nonces, next-nonce
    + 1); used for : generated, s, ls;
old-not-this-nonce : not old(all-nonces
```

```
  (next-nonce), all-nonces, next-nonce
    ); used for : generated, s, ls;
declaration
generateNonce:
generateNonce#(all-nonces; var next-nonce,
  ≪varname("Nonce")≫)
begin
  ≪varname("Nonce")≫ := all-nonces
    (next-nonce) ; next-nonce := next-nonce + 1
end;
end enrich
≪ENDFILE≫
≪ENDDEFINE≫

≪DEFINE generateGenerateKey FOR Model≫
≪FILE "generateKey/specification"-≫
enrich generateNonce, Key with
functions
   nonce2key : Nonce -> SymmKey;
procedures
   generateKey#
     (nat -> Nonce) : nat, SymmKey nonfunctional;
variables
   all-nonces : nat -> Nonce;
   next-nonce : nat;
axioms
nonce2key-def : nonce2key(mkNonce(str)) = mkSymmKey
  (str);
declaration
generateKey#:
generateKey#(all-nonces; var next-nonce,
  ≪varname("SymmKey")≫)
begin
  ≪varname("SymmKey")≫ := nonce2key(all-nonces
    (next-nonce)) ; next-nonce := next-nonce + 1
end;
end enrich
≪ENDFILE≫
≪ENDDEFINE≫

≪DEFINE generateKeys FOR Model≫
≪FILE "Key/specification"-≫
data specification
using string-data
≪IF this.usesSymmKey()-≫
SymmKey = mkSymmKey( . .key : string) with isSymmKey;
≪ENDIF-≫
≪IF this.usesASymmKey()-≫
PrivateKey = mkPrivateKey( .
  .key : string) with isPrivateKey;
PublicKey =  mkPublicKey ( .
  .key : string) with isPublicKey;
≪ENDIF-≫
variables
```

```
≪IF this.usesSymmKey()–≫
≪varname("SymmKey")≫,
  ≪varname("SymmKey")≫0,
    ≪varname("SymmKey")≫1 : SymmKey;
≪ENDIF–≫
≪IF this.usesASymmKey()–≫
≪varname("PrivateKey")≫,
  ≪varname("PrivateKey")≫0,
    ≪varname("PrivateKey")≫1 : PrivateKey;
≪varname("PublicKey")≫,
  ≪varname("PublicKey")≫0,
    ≪varname("PublicKey")≫1 : PublicKey;
≪ENDIF–≫
end data specification
≪ENDFILE≫
≪ENDDEFINE≫

≪DEFINE generateSecurityOperations FOR Model≫
≪FILE "SecurityOperations/specification"–≫
enrich data with
≪LET this.usesHashing() AS hasHashed–≫
≪LET this.usesEncryption() AS hasEncrypted–≫
≪LET this.usesAsymmEncryption()
  AS hasEncryptedAsymm–≫
≪LET this.usesSigning() AS hasSigned–≫
≪LET this.getCertificateClasses() AS certs–≫
≪LET this.usesSymmKey() AS hasSymmKey–≫
≪LET hasHashed || hasEncrypted || hasEncryptedAsymm || hasSigned ||
  (! certs.isEmpty) || hasSymmKey AS usesCrypto–≫
≪IF usesCrypto–≫
functions
≪ENDIF–≫
≪IF hasEncrypted–≫
 encrypt : SymmKey
   ≪convertSymbol("times")≫ PlainData
     ≪convertSymbol("implies")≫ EncData;
 decrypt : SymmKey
   ≪convertSymbol("times")≫ EncData
     ≪convertSymbol("implies")≫ PlainData;
≪ENDIF–≫
≪IF hasEncryptedAsymm–≫
 encrypt : PublicKey
   ≪convertSymbol("times")≫ PlainData
     ≪convertSymbol("implies")≫ EncDataAsymm;
 decrypt : PrivateKey
   ≪convertSymbol("times")≫ EncDataAsymm
     ≪convertSymbol("implies")≫ PlainData;
≪ENDIF–≫
≪IF hasSigned–≫
    sign : PrivateKey , SignData
      ≪convertSymbol("implies")≫ SignedData;
≪ENDIF–≫
≪IF hasHashed–≫
 hash : HashData
```

```
   ≪convertSymbol("implies")≫ HashedData;
≪ENDIF-≫
≪FOREACH certs AS cert-≫
    generateCertificate : PrivateKey
      ≪convertSymbol("times")≫
        ≪cert.getCertificateDataClass()
          .name≫
            ≪convertSymbol("implies")≫
              ≪cert.name≫;
≪ENDFOREACH-≫
≪IF hasSymmKey-≫
 generateKeyFromSecret : Secret -> SymmKey;
≪ENDIF-≫
≪IF hasHashed || hasEncrypted || hasEncryptedAsymm || hasSigned ||
  (! certs.isEmpty)-≫
predicates
≪ENDIF-≫
≪IF hasEncrypted-≫
 can_decrypt : SymmKey
   ≪convertSymbol("times")≫ EncData;
≪ENDIF-≫
≪IF hasEncryptedAsymm-≫
 can_decrypt : PrivateKey
   ≪convertSymbol("times")≫ EncDataAsymm;
≪ENDIF-≫
≪IF hasSigned-≫
    verify : PublicKey, SignedData, SignData;
≪ENDIF-≫
≪IF hasEncryptedAsymm || hasSigned-≫
 is_keypair : PublicKey
   ≪convertSymbol("times")≫ PrivateKey;
≪ENDIF-≫
≪IF hasHashed-≫
    verifyhash : HashedData, HashData;
≪ENDIF-≫
≪FOREACH certs AS cert-≫
    verifyCertificate : PublicKey
      ≪convertSymbol("times")≫
        ≪cert.name≫;
≪ENDFOREACH-≫
≪IF usesCrypto-≫
axioms
≪ENDIF-≫
≪IF hasEncrypted-≫
encrypt-def : encrypt(
  ≪varname("SymmKey")≫,
    ≪varname("PlainData")≫) = mkEncData(
      ≪varname("SymmKey")≫,
        ≪varname
          ("PlainData")≫); used for : s,ls;
decrypt-def : can_decrypt(
  ≪varname("SymmKey")≫, mkEncData(
    ≪varname("SymmKey")≫0,
      ≪varname("PlainData")≫))
```

```
        ≪convertSymbol("implies")≫ decrypt
          (
            ≪varname("SymmKey")≫, mkEncData
              (≪varname("SymmKey")≫0,
                ≪varname
                  ("PlainData")≫)) =
                    ≪varname
                      ("PlainData"
                        )≫; used for : s,ls;
can_decrypt-SymmKey-def : can_decrypt(
  ≪varname("SymmKey")≫, mkEncData(
    ≪varname("SymmKey")≫0,
      ≪varname("PlainData")≫))
        ≪convertSymbol("equivalent")≫
          ≪varname("SymmKey")≫ =
            ≪varname
              ("SymmKey")≫0; used for : s,ls;
≪ENDIF≫
≪IF hasEncryptedAsymm-≫
encryptAsymm-def : encrypt(
  ≪varname("PublicKey")≫,
    ≪varname("PlainData")≫) = mkEncDataAsymm
      (≪varname("PublicKey")≫,
        ≪varname
          ("PlainData")≫); used for : s,ls;
decryptAsymm-def : can_decrypt(
  ≪varname("PrivateKey")≫, mkEncDataAsymm(
    ≪varname("PublicKey")≫0,
      ≪varname("PlainData")≫))
        ≪convertSymbol("implies")≫ decrypt
          (
            ≪varname
              ("PrivateKey")≫, mkEncDataAsymm(
                ≪varname("PublicKey")≫0,
                  ≪varname
                    ("PlainData")≫)) =
                      ≪varname
                        ("PlainData"
                          )≫; used for : s,ls;
can_decrypt-PrivateKey : can_decrypt(
  ≪varname("PrivateKey")≫, mkEncDataAsymm(
    ≪varname("PublicKey")≫0,
      ≪varname("PlainData")≫)) <-> is_keypair
        (≪varname("PublicKey")≫0,
          ≪varname
            ("PrivateKey")≫); used for : s,ls;
≪ENDIF≫
≪IF hasSigned-≫
sign-def : sign(≪varname("PrivateKey")≫,
  ≪varname("SignData")≫) = mkSignedData(
    ≪varname("PrivateKey")≫,
      ≪varname("SignData")≫); used for : s,ls;
verify-def : verify(
  ≪varname("PublicKey")≫, mkSignedData(
```

```
    «varname("PrivateKey")»0,
      «varname("SignData")»0),
        «varname("SignData")»)
          «convertSymbol
            ("equivalent")» is_keypair(
              «varname("PublicKey")»,
                «varname("PrivateKey")»0)
                  «convertSymbol("and")»
                    «varname
                      ("SignData")» =
                        «varname
                          ("SignData"
                            )»0; used for : s,ls;
«ENDIF-»
«IF hasEncryptedAsymm || hasSigned-»
is_keypair-def  : is_keypair(
  «varname("PublicKey")»,
    «varname("PrivateKey")»0) <->
      «varname("PublicKey")».key =
        «varname("PrivateKey")»0.key;
is_keypair-same : is_keypair(mkPublicKey
  (str), mkPrivateKey(str)); used for : generated,s,ls;
«ENDIF-»
«IF hasHashed-»
hash-def : hash(
  «varname("HashData")») = mkHashedData(
    «varname("HashData")»); used for : s,ls;
verifyhash-def : verifyhash(
  «varname("HashedData")»,
    «varname("HashData")»)
      «convertSymbol("equivalent")» (
        «varname("HashedData")» = mkHashedData
          (
            «varname
              ("HashData")»)); used for : s,ls;
«ENDIF-»
«FOREACH certs AS cert-»
generateCertificate-«cert.name» :
generateCertificate(a_PrivateKey,
  «varname(cert.getCertificateDataClass()
    .name)») =
mk«cert.name»(
  «varname(cert.getCertificateDataClass()
    .name)»,
sign(a_PrivateKey,
  «cert.getCertificateDataClass()
    .WRAP2("SignData")»(
      «varname(cert.getCertificateDataClass()
        .name)»)));
verifyCertificate-«cert.name» :
verifyCertificate(a_PublicKey,
  «varname(cert.name)») <->
verify(a_PublicKey, «varname(cert.name)».
  «cert.getCertificateSignatureAttribute()
```

```
    .name≫,
≪cert.getCertificateDataClass()
  .WRAP2("SignData")≫(
    ≪varname(cert.name)≫.
      ≪cert.getCertificateDataAttribute()
        .name≫));
≪ENDFOREACH-≫
≪IF hasSymmKey≫
generateKeyFromSecret :
generateKeyFromSecret(a_Secret) = mkSymmKey(a_Secret
  .secret);
≪ENDIF≫
≪ENDLET≫≪ENDLET≫≪ENDLET≫
  ≪ENDLET≫≪ENDLET≫≪ENDLET≫
    ≪ENDLET≫
end enrich
≪ENDFILE≫
≪ENDDEFINE≫
```

Chapter 8

Transformations for the communication infrastructure

```
≪IMPORT uml≫
≪EXTENSION templates::deployment≫
≪EXTENSION templates::getter≫
≪EXTENSION templates::parse≫
≪EXTENSION templates::attacker≫

≪DEFINE Root FOR uml::Model≫
≪IF this.getDeploymentDiagram() != null≫
  ≪EXPAND generateFromDeploymentDiagram≫
    ≪ENDIF≫
≪ENDDEFINE≫

≪DEFINE generateFromDeploymentDiagram
  FOR Model≫
≪EXPAND generateEndpoint FOR this≫
≪EXPAND generateEndpointPreds FOR this≫
≪EXPAND generateExagent FOR this≫
≪EXPAND generateAgentConsts FOR this≫
≪EXPAND generateAllInstances FOR this≫
≪EXPAND generateConnection FOR this≫
≪EXPAND generateConnections FOR this≫
≪EXPAND generateConnectionPreds FOR this≫
≪EXPAND generatePorts FOR this≫
≪ENDDEFINE≫

≪DEFINE generateEndpoint FOR Model≫
≪FILE "endpoint/specification"-≫
data specification using
   agentPreds, ports
endpoint = .
  ≪convertSymbol("odot")≫ . (.
    .agent : agent; . .port : ports);
variables
endpoint, endp, endp0, endp1, endp2, endp3 : endpoint;
end data specification
≪ENDFILE≫
≪ENDDEFINE≫
```

```
≪DEFINE generateEndpointPreds FOR Model≫
≪FILE "endpointPreds/specification"−≫
enrich connections with
functions
 other−endpoint : endpoint
   ≪convertSymbol("times")≫ connection
     ≪convertSymbol("implies")≫ endpoint;
predicates
        endpoint−ok    : endpoint;
        is−endpoint    : endpoint
          ≪convertSymbol("times")≫ connection;
        is−endpoint    : agent
          ≪convertSymbol("times")≫ connection;
        is−valid−port : agent
          ≪convertSymbol("times")≫ ports;
axioms
is−valid−port−def :
is−valid−port(ag, port) <−> endpoint−ok
  (ag \odot port); used for : s, ls;
other−endpoint−left : other−endpoint
  (endp1, mk−connection
    (endp1,endp2)) = endp2; used for : s, ls;
other−endpoint−right : other−endpoint
  (endp2, mk−connection
    (endp1,endp2)) = endp1; used for : s, ls;
is−endpoint−agent−def :
is−endpoint(ag, conn)
  ≪convertSymbol("equivalent")≫ conn.endpoint1
    .agent = ag
      ≪convertSymbol("or")≫ conn.endpoint2
        .agent = ag;
is−endpoint−def :
is−endpoint(endp,conn)
  ≪convertSymbol("equivalent")≫ (endp = conn
    .endpoint1
      ≪convertSymbol("or")≫ endp = conn
        .endpoint2);
endpoint−ok−def :
endpoint−ok(endp)
  ≪convertSymbol("equivalent")≫
 (
≪LET this.getAllNodes() AS nodes−≫
≪FOREACH nodes
  AS node SEPARATOR convertSymbol("or")−≫
≪LET node.portNames() AS ports−≫
≪FOREACH ports
  AS end SEPARATOR convertSymbol("or")−≫
    ≪IF node.isTerminal()−≫
    (is_≪node.name≫(endp.agent)
      ≪convertSymbol("and")≫ (endp.port =
        ≪end≫))
    ≪ENDIF−≫
    ≪IF node.isSmartCard()−≫
    (is_≪node.name≫(endp.agent)
```

```
      «convertSymbol("and")» (endp.port =
        «end»))
    «ENDIF-»
    «IF node.isService()-»
    (is_«node.name»(endp.agent)
      «convertSymbol("and")» (endp.port =
        «end»))
    «ENDIF-»
 «IF node.isUser()-»
    (is_user(endp.agent)
      «convertSymbol("and")» (endp.port =
        «end»))
    «ENDIF-»
 «IF node.isSmartCard() && node
   .isFakeCardlet()-»
or (is_fakecardlet(endp.agent)
  «convertSymbol("and")» (endp.port =
    «end»))
 «ENDIF-»
«ENDFOREACH-»
«ENDLET-»
«ENDFOREACH-»
«ENDLET-»
 );
end enrich
«ENDFILE»
«ENDDEFINE»

«DEFINE generateExagent FOR Model»
«LET this.getAllNodes() AS nodes-»
«FILE "exagent/specification"-»
enrich agentconsts with
predicates
«convertSymbol("exists")»agent : agent;
«IF nodes.exists(e|e.isTerminal())-»
is_terminal : agent;
«ENDIF-»
«IF nodes.exists(e|e.isSmartCard())-»
is_smartcard : agent;
«ENDIF-»
«IF nodes.exists(e|e.isService())-»
is_service : agent;
«ENDIF-»
axioms
«FOREACH nodes.select(e|e.isTerminal()) AS t-»
ex-«t.name»:
  «convertSymbol("exists")»agent(
    «t.name»(n))
      «convertSymbol
        ("equivalent")» n < NUMOF
          «t.name.toUpperCase()»S;
ex-«t.name»-simp: NUMOF
  «t.name.toUpperCase()»S
    «convertSymbol("le")» n
```

```
      «convertSymbol("implies")»
        «convertSymbol("not")»
          «convertSymbol("exists")»agent(
            «t
              .name»
                (n)); used for : s,ls,generated;
ex-«t.name»-zero:
  «convertSymbol("exists")»agent(
    «t.name»(0)); used for : s,ls,generated;
«ENDFOREACH»
«FOREACH nodes.select(e|e.isSmartCard())
  AS sc»
ex-«sc.name»:
  «convertSymbol("exists")»agent(
    «sc.name»(n))
      «convertSymbol
        ("equivalent")» n < NUMOF
          «sc.name.toUpperCase()»S;
ex-«sc.name»-simp: NUMOF
  «sc.name.toUpperCase()»S
    «convertSymbol("le")» n
      «convertSymbol("implies")»
        «convertSymbol("not")»
          «convertSymbol("exists")»agent(
            «sc
              .name»
                (n)); used for : s,ls,generated;
ex-«sc.name»-zero:
  «convertSymbol("exists")»agent(
    «sc.name»(0)); used for : s,ls,generated;
«ENDFOREACH»
«FOREACH nodes.select(e|e.isService()) AS t»
ex-«t.name»:
  «convertSymbol("exists")»agent(
    «t.name»(n))
      «convertSymbol
        ("equivalent")» n < NUMOF
          «t.name.toUpperCase()»S;
ex-«t.name»-simp: NUMOF
  «t.name.toUpperCase()»S
    «convertSymbol("le")» n
      «convertSymbol("implies")»
        «convertSymbol("not")»
          «convertSymbol("exists")»agent(
            «t
              .name»
                (n)); used for : s,ls,generated;
ex-«t.name»-zero:
  «convertSymbol("exists")»agent(
    «t.name»(0)); used for : s,ls,generated;
«ENDFOREACH»
«IF nodes.exists(e|e.isUser())»
ex-user: exagent(user(n))
  «convertSymbol("equivalent")» n < NUMOFUSERS;
```

```
ex-user-zero:
  «convertSymbol("exists")»agent(user
    (0)); used for : s,ls,generated;
«ENDIF-»
ex-attacker:
  «convertSymbol("exists")»agent
    (attacker); used for : s,ls;
«IF nodes.exists(e|e.isTerminal())-»
is-terminal: is_terminal(ag)
  «convertSymbol("equivalent")-»
 «FOREACH this.getModel().getAllTerminals()
   AS terminal SEPARATOR convertSymbol("or")-»
 is_«terminal.name»(ag)«ENDFOREACH-»;
«ENDIF-»
«IF nodes.exists(e|e.isSmartCard())-»
is-smartcard: is_smartcard(ag)
  «convertSymbol("equivalent")-»
 «FOREACH this.getModel().getAllSmartcards()
   AS sc SEPARATOR convertSymbol("or")-»
 is_«sc.name»(ag)«ENDFOREACH-»;
«ENDIF-»
«IF nodes.exists(e|e.isService())-»
is-service: is_service(ag)
  «convertSymbol("equivalent")-»
 «FOREACH this.getModel().getAllServices()
   AS service SEPARATOR convertSymbol("or")-»
 is_«service.name»(ag)«ENDFOREACH-»;
«ENDIF-»
end enrich
«ENDFILE»
«ENDLET»
«ENDDEFINE»

«DEFINE generateAgentConsts FOR Model»
«LET this.getAllNodes() AS nodes-»
«FILE "agentconsts/specification"-»
enrich endpointPreds with
constants
«FOREACH nodes.select(e|e.isTerminal()) AS t-»
NUMOF«t.name.toUpperCase()»S : nat;
«ENDFOREACH-»
«FOREACH nodes.select(e|e.isSmartCard())
  AS sc-»
NUMOF«sc.name.toUpperCase()»S : nat;
«ENDFOREACH-»
«FOREACH nodes.select(e|e.isService()) AS s-»
NUMOF«s.name.toUpperCase()»S : nat;
«ENDFOREACH-»
«IF nodes.exists(e|e.isUser())-»
NUMOFUSERS : nat;
«ENDIF-»
axioms
«FOREACH nodes.select(e|e.isTerminal()) AS t-»
NUMOF
```

```
  ≪t.name.toUpperCase()≫S-not-zero : NUMOF
    ≪t.name
      .toUpperCase()≫S \neq 0; used for : s,ls;
≪ENDFOREACH-≫
≪FOREACH nodes.select(e|e.isSmartCard())
  AS sc-≫
NUMOF
  ≪sc.name
    .toUpperCase()≫S-not-zero : NUMOF
      ≪sc.name
        .toUpperCase()≫S \neq 0; used for : s,ls;
≪ENDFOREACH-≫
≪FOREACH nodes.select(e|e.isService()) AS s-≫
NUMOF
  ≪s.name.toUpperCase()≫S-not-zero : NUMOF
    ≪s.name
      .toUpperCase()≫S \neq 0; used for : s,ls;
≪ENDFOREACH-≫
≪IF nodes.exists(e|e.isUser())-≫
NUMOFUSERS-not-zero : NUMOFUSERS \neq 0; used for : s
  ,ls;
≪ENDIF-≫
end enrich
≪ENDFILE≫
≪ENDLET≫
≪ENDDEFINE≫

≪DEFINE generateAllInstances FOR Model≫
≪LET this.getAllNodes().select(n|n
  .isTerminal() || n.isSmartCard() || n.isService())
    AS nodes-≫
≪FILE "allInstances/specification"-≫
enrich agents, natlist with
constants
≪FOREACH nodes AS n-≫
      ≪n.name≫AllInstances : agents;
≪ENDFOREACH-≫
functions
      count-m-to-n : nat, nat -> natlist;
≪FOREACH nodes AS n-≫
      mk
        ≪n
          .name≫AllInstances : natlist -> agents;
≪ENDFOREACH-≫
predicates
≪FOREACH nodes AS n-≫
      ex≪n.name≫Agents : agents;
≪ENDFOREACH-≫
axioms
count-m-to-n-done : n
  ≪convertSymbol("le")≫ m -> count-m-to-n
    (m, n) = []; used for : s, ls;
count-m-to-n-rec : m < n -> count-m-to-n(m, n) = m
  + count-m-to-n(m + 1, n); used for : s, ls;
```

```
≪FOREACH nodes AS n–≫
mk≪n.name≫AllInstances–empty : mk
  ≪n
    .name≫AllInstances([]) = []; used for : s, ls;
mk≪n.name≫AllInstances–rec : mk
  ≪n.name≫AllInstances(m ' + nats) =
    ≪n.name≫(m) + mk
      ≪n
        .name≫AllInstances(nats); used for : s, ls;
≪n.name≫AllInstances–def :
  ≪n.name≫AllInstances = mk
    ≪n
      .name≫AllInstances(count–m–to–n(0, NUMOF
        ≪n.name.toUpperCase()≫S));
ex≪n.name≫Agents–def : ex
  ≪n.name≫Agents(agents) <–> all ag. ag
    ≪convertSymbol("in")≫ agents –> exagent
      (ag) and is_≪n.name≫(ag);
≪ENDFOREACH–≫
(: generated theorems :)
count–m–to–n–dups : not dups(count–m–to–n
  (m, n)); used for : s, ls, generated;
count–m–to–n–in : m \in count–m–to–n
  (n, n0) <–> not not
    (n \le m and m < n0); used for : s, ls, generated;
≪FOREACH nodes AS n–≫
≪n.name≫AllInstances–dups : not dups(
  ≪n
    .name≫AllInstances
      ); used for : s, ls, generated;
≪n.name≫AllInstances–in : ag \in
  ≪n
    .name≫AllInstances <–> not not (exagent
      (ag) and is_
        ≪n
          .name≫(ag)); used for : s,ls, generated;
ex≪n.name≫Agents–
  ≪n.name≫AllInstances : ex
    ≪n.name≫Agents(
      ≪n
        .name≫AllInstances
          ); used for : s,ls, generated;
ex
  ≪n
    .name≫Agents–not : not exagent
      (ag) and ag \in agents –> not ex
        ≪n
          .name≫Agents
            (agents); used for : s,ls, generated;
ex≪n.name≫Agents–not : not is_
  ≪n.name≫(ag) and ag \in agents –> not ex
    ≪n
      .name≫Agents
        (agents); used for : s,ls, generated;
```

```
ex≪n.name≫Agents-rec : ex
  ≪n.name≫Agents(ag '
    + agents) <-> not not (exagent(ag) and is_
      ≪n.name≫(ag) and ex
        ≪n
          .name≫Agents
            (agents)); used for : s,ls, generated;
mk≪n.name≫AllInstances-dups : dups(mk
  ≪n
    .name≫AllInstances(nats)) <-> dups
      (nats); used for : s,ls, generated;
mk≪n.name≫AllInstances-in : ag \in mk
  ≪n
    .name≫AllInstances(nats) <-> not not (is_
      ≪n.name≫(ag) and ag
        .name \in nats); used for : s,ls, generated;
≪ENDFOREACH≫
end enrich
≪ENDFILE≫
≪ENDLET≫
≪ENDDEFINE≫

≪DEFINE generateConnection FOR Model≫
≪FILE "connection/specification"-≫
data specification
using endpoint
connection = mk-connection(. .endpoint1 : endpoint; .
  .endpoint2 : endpoint);
variables
 conn, conn0, conn1, conn2, conn3 : connection;
end data specification
≪ENDFILE≫
≪ENDDEFINE≫

≪DEFINE generateConnections FOR Model≫
≪FILE "connections/specification"-≫
actualize set-union with connection
by morphism
elem
  ≪convertSymbol
    ("implies")≫ connection; set
      ≪convertSymbol("implies")≫ connections;
a
  ≪convertSymbol
    ("implies")≫ connection; b
      ≪convertSymbol
        ("implies")≫ connection0; c
          ≪convertSymbol
            ("implies")≫ connection1;
s
  ≪convertSymbol
    ("implies")≫ connectionset; s0
      ≪convertSymbol
        ("implies")≫ connectionset0; s1
```

```
          ≪convertSymbol
            ("implies")≫ connectionset1; s2
              ≪convertSymbol
                ("implies")≫ connectionset2;
end actualize
≪ENDFILE≫
≪ENDDEFINE≫

≪DEFINE generateConnectionPreds FOR Model≫
≪FILE "connectionPreds/specification"-≫
enrich exagent with
predicates
conn-ok : connection;
valid-conn : connection;
valid-conns : connections;
connected : endpoint
  ≪convertSymbol("times")≫ connections;
connected : agent
  ≪convertSymbol("times")≫ connections;
connect-possible : connection
  ≪convertSymbol("times")≫ connections;
variables
conns, connections : connections;
axioms
conn-ok : conn-ok(conn)
  ≪convertSymbol("equivalent")≫
endpoint-ok(conn.endpoint1)
  ≪convertSymbol("and")≫ endpoint-ok(conn
    .endpoint2) ≪convertSymbol("and")≫ (
≪FOREACH this.getDeploymentDiagram()
  .packagedElement.typeSelect(CommunicationPath)
    AS a SEPARATOR convertSymbol("or")-≫
≪a.generateThreatString()≫
≪ENDFOREACH-≫
);
valid-conn : valid-conn(conn)
             ≪convertSymbol
               ("equivalent")≫ conn-ok(conn)
             ≪convertSymbol("and")≫
               ≪convertSymbol("exists")≫agent
                 (conn.endpoint1.agent)
             ≪convertSymbol("and")≫
               ≪convertSymbol("exists")≫agent
                 (conn.endpoint2.agent);
valid-conns : valid-conns(connections) <->
  ≪convertSymbol("all")≫ conn. conn
    ≪convertSymbol
      ("in")≫ connections -> valid-conn(conn);
connected-endpoint-def : connected
  (endp1,connections)
    ≪convertSymbol("equivalent")≫
 (
   ≪convertSymbol("exists")≫ endp2
     . mk-connection(endp1, endp2)
```

```
        ≪convertSymbol("in")≫ connections
              ≪convertSymbol
                ("or")≫ mk-connection
                  (endp2, endp1)
                    ≪convertSymbol
                      ("in")≫ connections);
connected-agent-def : connected(ag, connections)
  ≪convertSymbol("equivalent")≫
                      ≪convertSymbol
                        ("exists")≫ port
                          . connected(ag
                            ≪convertSymbol
                              ("odot"
                                )≫ port
                                  , connections);
connect-possible :
              connect-possible(conn,connections)
          ≪convertSymbol("equivalent")≫
            (     valid-conn(conn)
                      ≪convertSymbol
                        ("and")≫
                          ≪convertSymbol
                            ("not")≫ connected(conn
                              .endpoint1,connections)
                      ≪convertSymbol
                        ("and")≫
                          ≪convertSymbol
                            ("not")≫ connected(conn
                              .endpoint2,connections)
                      ≪convertSymbol("and")≫
                        (is_statefulService(conn
                          .endpoint2.agent)
                            ≪convertSymbol
                              ("implies")≫
                       ≪convertSymbol
                         ("not")≫ connected(conn
                           .endpoint2
                             .agent, connections) )
                     );
(: generated axioms :)
valid-conns-add : valid-conns
  (connections) and connect-possible
    (conn, connections) -> valid-conns(connections +
      + conn); used for : generated, s, ls;
valid-conns-delete : valid-conns
  (connections) -> valid-conns
    (connections -- conn); used for : generated, s, ls;
end enrich
≪ENDFILE≫
≪ENDDEFINE≫

≪DEFINE generatePortsWithNatSort FOR Model≫
≪FILE "ports/specification"-≫
enrich connection with
```

```
constants
«LET this.getAllPortNames() AS ports»
«FOREACH ports AS end-»
«end» : nat;
«ENDFOREACH-»
axioms
«FOREACH ports AS end-»
«end» :
  «convertSymbol("follows")»
    «end» = «ports.indexOf(end) + 1»;
«ENDFOREACH-»
«ENDLET»
end enrich
«ENDFILE»
«ENDDEFINE»

«DEFINE generatePorts FOR Model»
«FILE "ports/specification"-»
data specification
«LET this.getAllPortNames() AS ports»
ports =
  «FOREACH ports AS end SEPARATOR " | "-»
    «end»
      «ENDFOREACH-»;
«ENDLET»
variables port, port0, port1 : ports;
end data specification
«ENDFILE»
«ENDDEFINE»
```

Chapter 9

Transformations for miscellaneous parts

```
«IMPORT uml»
«EXTENSION templates::getter»
«EXTENSION templates::parse»
«EXTENSION templates::deployment»

«DEFINE Root FOR uml::Model»
«EXPAND generateConnectDisconnect FOR this»
«EXPAND generateSend FOR this»
«EXPAND generateAsmstep FOR this»
«EXPAND generateInputs FOR this»
«EXPAND generateUser FOR this»
«EXPAND generateAttacker FOR this»
«EXPAND generateLists FOR this»
«IF this.existsFakeCardlet()-»
 «EXPAND generateFakeCardletStep FOR this»
 «EXPAND generateFakeCardletSimple FOR this»
«ENDIF»
«ENDDEFINE»

«DEFINE generateAsmstep FOR uml::Model»
«FILE "asmstep/specification"»
data specification
asm-step =
      «FOREACH this.getAllSmartcards()
        AS scclass-»
      «scclass.name»-agent-step |
      «ENDFOREACH-»
      «FOREACH this.getAllTerminals()
        AS terminalclass-»
      «terminalclass.name»-agent-step |
      «ENDFOREACH-»
      «FOREACH this.getAllServices()
        AS serviceclass-»
      «serviceclass.name»-agent-step |
      «ENDFOREACH-»
      «IF this.existsFakeCardlet()-»
```

```
      forgeable-cardlet-agent-step |
      «ENDIF-»
      connect | disconnect | user-agent-step | attacker-agent-step;

attacker-step
  = attacker-send | attacker-suppress | attacker-disconnect;

user-step = read | send;

variables
asm-step, asm-step0 : asm-step;
attacker-step, attacker-step0 : attacker-step;
user-step, user-step0 : user-step;
end data specification
«ENDFILE»
«ENDDEFINE»

«DEFINE generateFakeCardletStep FOR uml::Model»
«FILE "fakecardletstep/specification"»
data specification
fake-cardlet-step = generate-response | call-method;

variables
fake-cardlet-step, fake-cardlet-step0 : fake-cardlet-step;
end data specification
«ENDFILE»
«ENDDEFINE»

«DEFINE generateMessagelistSublist
  FOR uml::Model»
«FILE "messagelist-sublist/specification"»
enrich messagelist with
predicates
 . «convertSymbol("uparrow")» . : messagelist
        «convertSymbol("times")» messagelist;

axioms
 empty-sublist :
   «convertSymbol("follows")» []
     «convertSymbol
       ("uparrow")» msgs; used for : s, ls;
 sublist-bad : «convertSymbol("follows")»
   «convertSymbol("not")» msg ' + msgs
     «convertSymbol
       ("uparrow")» []; used for : s, ls;
 sublist-rec-same-head :
   «convertSymbol("follows")» ((msg '
     + msgs
       «convertSymbol("uparrow")» msg '
         + msgs0)
           «convertSymbol("equivalent")»
             (msgs
               «convertSymbol
                 ("uparrow")» msgs0)
```

```
                    ); used for : s, ls;
 sublist-rec-different-head :
   ≪convertSymbol("follows")≫ (msg
     ≪convertSymbol("unequal")≫ msg0)
       ≪convertSymbol("implies")≫ (msg '
         + msgs
           ≪convertSymbol("uparrow")≫ msg0 '
             + msgs0
               ≪convertSymbol("equivalent")≫
                 (msg ' + msgs
                   ≪convertSymbol
                     ("uparrow")≫ msgs0)
                       ); used for : s,ls;
    sublist-member:
      msg \in msgs and msgs \uparrow msgs0 -> msg \in msgs0;
      used for : s,ls,generated;

    sublist-rec-different-head : |-
      (msgs0 \neq [] and msg \neq msgs0
        .first) -> (msg '
          + msgs \uparrow msgs0 <-> ((msg '
            + msgs) \uparrow msgs0
              .rest)); used for: generated;
 sublist-rec-different-head2 : |-
   (msgs \neq [] and msg0 \neq msgs
     .first) -> (msgs \uparrow msg0 '
       + msgs0 <->
         (msgs \uparrow msgs0)); used for: generated;
 sublist-empty: msgs \uparrow [] <-> msgs
   = []; used for: s,ls,generated;
end enrich
≪ENDFILE≫
≪ENDDEFINE≫

≪DEFINE generateConnectDisconnect
  FOR uml::Model≫
≪FILE "ConnectDisconnect/specification"≫
enrich Attacker with
predicates
    disconnect-possible : connection, connections ,
      (agent -> ports -> messagelist);
       connect-possible : connection, connections ,
         (agent -> ports -> messagelist);
procedures
      CONNECT : connections
        ≪convertSymbol("times")≫ (agent
          ≪convertSymbol
            ("implies")≫ ports
              ≪convertSymbol
                ("implies")≫ messagelist
                  ) nonfunctional indeterministic;
   DISCONNECT : connections
     ≪convertSymbol("times")≫ (agent
       ≪convertSymbol("implies")≫ ports
```

```
        ≪convertSymbol
          ("implies")≫ messagelist
            ) nonfunctional indeterministic;
variables
   input  : ports
     ≪convertSymbol("implies")≫ messagelist;
      inputs : agent
        ≪convertSymbol("implies")≫ ports
          ≪convertSymbol
            ("implies")≫ messagelist;
axioms
connect-possible-def : connect-possible
  (conn, connections, inputs)
                    <-> connect-possible
                      (conn, connections) and
                         inputs(conn .endpoint1
                           .agent)(conn .endpoint1
                             .port) = [] and
                         inputs(conn .endpoint2
                           .agent)(conn .endpoint2
                             .port) = []
                         ;
disconnect-possible-def : disconnect-possible
  (conn, connections, inputs)
                    <-> conn \in connections and
                         inputs(conn .endpoint1
                           .agent)(conn .endpoint1
                             .port) = [] and
                         inputs(conn .endpoint2
                           .agent)(conn .endpoint2
                             .port) = []
                         ;
valid-conns-add : valid-conns
  (connections) and connect-possible
    (conn, connections, inputs) -> valid-conns
      (connections +
       + conn); used for : generated, s, ls;
declaration
   CONNECT
   : CONNECT (; var connections, inputs)
       {
          choose conn
               with connect-possible
                 (conn, connections, inputs)
               in {
                     connections := connections +
                       + conn ;
                     (:
                     if ( is_statefulService(conn
                       .endpoint2 .agent) ) then
                      { reset(conn .endpoint2 .agent) }
                     :)
                  } ifnone skip
       };
```

```
   DISCONNECT
   : DISCONNECT (var  connections, inputs)
       {
          choose  conn
                with disconnect-possible
                  (conn, connections, inputs)
                in {
                      connections :
                        = connections -- conn ;
                      inputs(conn .endpoint1
                        .agent)(conn .endpoint1
                          .port) := [] ;
                      inputs(conn .endpoint2
                        .agent)(conn .endpoint2
                          .port) := [] ;
                   } ifnone skip
       };
end enrich
≪ENDFILE≫
≪ENDDEFINE≫

≪DEFINE generateSend FOR uml::Model≫
≪FILE "SEND/specification"≫
enrich ConnectDisconnect with
procedures
      SEND  message ≪convertSymbol("times")≫ ports
            ≪convertSymbol("times")≫ agent
            ≪convertSymbol("times")≫ connections :
            attackerdataset ≪convertSymbol("times")≫
            (agent≪convertSymbol("implies")≫ ports
            ≪convertSymbol("implies")≫ messagelist)
            nonfunctional indeterministic;

variables
      inputs : agent
               ≪convertSymbol("implies")≫ ports
               ≪convertSymbol("implies")≫ messagelist;

      input : ports
              ≪convertSymbol("implies")≫ messagelist;
      attacker-known : attackerdataset;
      outport, rem-port : ports;
      outmsg : message;
      rem-agent : agent;

declaration
   SEND
   : SEND
     (outmsg, outport, ag
       , connections; var attacker-known, inputs)
       {
        choose  conn
               with is-endpoint(ag
                 ≪convertSymbol
```

```
                    ("odot")≫ outport,conn)
                      ≪convertSymbol
                        ("and")≫ conn
                          ≪convertSymbol
                            ("in")≫ connections
                in let rem-agent = other-endpoint
                  (ag
                    ≪convertSymbol
                      ("odot")≫ outport, conn)
                                  .agent,
                     rem-port = other-endpoint
                       (ag
                         ≪convertSymbol
                           ("odot"
                             )≫ outport, conn)
                               .port
                   in {
                       if attacker-can-read(conn)
                       then attacker-known :
                         = attacker-known
                           ≪convertSymbol
                             ("integral")≫
                               + outmsg ;
                       if
                         ≪convertSymbol
                           ("not")≫ is_attacker
                             (rem-agent)
                       then inputs := add
                         (rem-agent, rem-port, outmsg
                           , inputs);
                     } ifnone skip
      };
end enrich
≪ENDFILE≫
≪ENDDEFINE≫

≪DEFINE generateInputs FOR uml::Model≫
≪FILE "inputs/specification"≫
enrich  messagelist, connectionPreds with
functions
   add : agent, ports, message,
     (agent -> ports -> messagelist) ->
       (agent -> ports -> messagelist
         ); comment
           : add message to the end of the message list
             .;
   rem : agent, ports,
     (agent -> ports -> messagelist) ->
       (agent -> ports -> messagelist
         ); comment
           : remove first message from message list
             (unspecified if empty).;
predicates
   input-available : agent
```

```
    ≪convertSymbol("times")≫ (agent
      ≪convertSymbol("implies")≫ ports
        ≪convertSymbol
          ("implies")≫ messagelist);
  msgininputs : message
    ≪convertSymbol("times")≫ (agent
      ≪convertSymbol("implies")≫ ports
        ≪convertSymbol
          ("implies")≫ messagelist);
variables
    inputs, inputs0, inputs1, inputs2 : agent
      ≪convertSymbol("implies")≫ ports
        ≪convertSymbol
          ("implies")≫ messagelist;
    portmsgs, portmsgs0, portmsgs1
      , portmsgs2 : ports
        ≪convertSymbol
          ("implies")≫ messagelist;
axioms
add-def : add(ag, port, msg, inputs) = inputs(ag ;
  (inputs(ag)(port ; inputs(ag)(port) + msg)));
rem-def : rem(ag, port, inputs) = inputs(ag ; (inputs
  (ag)(port ; inputs(ag)(port).rest)));
inputavailable : input-available(ag,inputs)
  ≪convertSymbol("equivalent")≫
    ≪convertSymbol("exists")≫ port
      . inputs(ag)(port)
        ≪convertSymbol("neq")≫ [];
msgininputs : msgininputs(msg,inputs)
  ≪convertSymbol("equivalent")≫
    ≪convertSymbol("exists")≫ ag, port
      . msg
        ≪convertSymbol("in")≫ inputs(ag)(port);
inputs-identity : inputs(ag ; inputs
  (ag)) = inputs; used for : generated, s,ls;
inputs-identity : portmsgs(port ; portmsgs
  (port)) = portmsgs; used for : generated, s,ls;
inputs-identity : portmsgs(port) = msgs    -> portmsgs
  (port ; msgs) = portmsgs; used for : generated;
inputs-identity : inputs(ag) = portmsgs -> inputs
  (ag ; portmsgs) = inputs; used for : generated;
inputs-identity : not ag = ag0 -> inputs(ag ; portmsgs
  (port ; msgs))(ag0 ; inputs(ag0))
                                = inputs(ag ; portmsgs
                                  (port ; msgs)
                                    ) ; used for
                                      : generated, s
                                        , ls ;
inputs-override : inputs(ag ; portmsgs)
  (ag ; portmsgs0) = inputs
    (ag ; portmsgs0); used for : generated, s,ls;
inputs-override : portmsgs(port ; msgs)
  (port ; msgs0) = portmsgs
    (port ; msgs0); used for : generated, s,ls;
```

```
inputs-switch : not ag = ag0 -> inputs(ag ; portmsgs)
  (ag0 ; portmsgs0) = inputs(ag0 ; portmsgs0)
    (ag ; portmsgs); used for : generated;
inputs-switch : not port = port0 -> portmsgs
  (port ; msgs)(port0 ; msgs0) = portmsgs
    (port0 ; msgs0)(port ; msgs); used for : generated;
msgininputs-add : msgininputs(msg, add
  (ag, port, msg, inputs)
    ); used for : generated, s, ls;
msgininputs-notin : not msgininputs
  (msg, inputs) -> not msg \in inputs(ag)
    (port); used for : generated, s, ls;
msgininputs-notin : not msgininputs
  (msg, inputs) and msg \in msgs0 -> not inputs(ag)
    (port) = msg0 '
      + msgs0; used for : generated, s, ls;
end enrich
≪ENDFILE≫
≪ENDDEFINE≫

≪DEFINE generateUser FOR uml::Model≫
≪FILE "USER/specification"≫
enrich SEND with
 procedures
      USER-READ agent
        ≪convertSymbol
          ("times")≫ connections : (agent
            ≪convertSymbol
              ("implies")≫ ports
                ≪convertSymbol
                  ("implies")≫ messagelist
                    ) nonfunctional indeterministic;
      USER-SEND  agent
        ≪convertSymbol("times")≫ connections : attackerdataset
        ≪convertSymbol("times")≫ (agent
        ≪convertSymbol("implies")≫ ports
        ≪convertSymbol("implies")≫ messagelist)
        nonfunctional indeterministic;
      USER  agent
        ≪convertSymbol("times")≫ connections : attackerdataset
        ≪convertSymbol("times")≫(agent
        ≪convertSymbol("implies")≫ ports
        ≪convertSymbol("implies")≫ messagelist)
        nonfunctional indeterministic;

variables
   connections : connections;
declaration
   USER-READ : USER-READ(ag, connections; inputs)
       {
          choose  port
          with connected(ag
            ≪convertSymbol
              ("odot")≫ port, connections)
```

```
            in {
                  inputs(ag)(port) := []
               } ifnone skip
         };
   USER-SEND : USER-SEND
      (ag, connections; var attacker-known, inputs)
        {
           choose  port, msg
           with connected(ag
             ≪convertSymbol
               ("odot")≫ port, connections
                 ) and is_user_message(msg)
              in SEND
                (msg, port, ag
                  , connections; attacker-known, inputs)
           ifnone skip
         };
   USER : USER
      (ag, connections; attacker-known, inputs)
        {
             let user-step = [?]
                in if user-step = send
                   then USER-SEND
                     (ag, connections; attacker-known
                       , inputs)
                   else if user-step = read
                        then USER-READ
                          (ag, connections; inputs)
         };
end enrich
≪ENDFILE≫
≪ENDDEFINE≫

≪DEFINE generateAttacker FOR uml::Model≫
≪FILE "Attacker/specification"≫
enrich Generate, AttackerAbilities, inputs, asmstep with
procedures
  ATTACKER-SEND  connections
     ≪convertSymbol("times")≫ attackerdataset :
     (agent≪convertSymbol("implies")≫ ports
     ≪convertSymbol("implies")≫ messagelist)
     nonfunctional indeterministic;

  ATTACKER-SUPPRESS  connections :
     (agent≪convertSymbol("implies")≫ ports
     ≪convertSymbol("implies")≫ messagelist)
     nonfunctional indeterministic;

  ATTACKER-DISCONNECT : connections,
    (agent -> ports -> messagelist)
    nonfunctional indeterministic;

  ATTACKER  connections
    ≪convertSymbol("times")≫ attackerdataset :
```

```
    (agent ≪convertSymbol("implies")≫ ports
    ≪convertSymbol("implies")≫ messagelist)
    nonfunctional indeterministic;

variables
      inputs : agent
        ≪convertSymbol("implies")≫ ports
          ≪convertSymbol
            ("implies")≫ messagelist;
      input : ports
        ≪convertSymbol("implies")≫ messagelist;
      attacker-known : attackerdataset;
declaration
   ATTACKER-SEND
   : ATTACKER-SEND
     (connections, attacker-known; var inputs)
       {
          choose  msg, endp
          with attacker-known \grgr amessage(msg)
           and attacker-can-send(endp, connections)
             in inputs(endp .agent)(endp.port) :=
                inputs(endp.agent)(endp.port) + msg '
             ifnone skip
       };
   ATTACKER-SUPPRESS
   : ATTACKER-SUPPRESS (connections; var inputs)
       {  choose endp
          with attacker-can-suppress(endp, connections)
           and not inputs(endp .agent)(endp .port) = []
          in inputs := rem(endp .agent, endp
            .port, inputs)
          ifnone skip
       };
       ATTACKER-DISCONNECT
   : ATTACKER-DISCONNECT (var connections, inputs)
       { choose  conn
                with conn
                  ≪convertSymbol
                    ("in"
                      )≫ connections and attacker-can-suppress
                        (conn, connections)
                in {
                      connections :
                        = connections -- conn ;
                      inputs(conn .endpoint1
                        .agent)(conn .endpoint1
                          .port) := [] ;
                      inputs(conn .endpoint2
                        .agent)(conn .endpoint2
                          .port) := [] ;
                   } ifnone skip
        };
   ATTACKER
   : ATTACKER (connections, attacker-known; var inputs)
```

```
        {
           let attacker-step = [?] in
                if attacker-step = attacker-send
                   then ATTACKER-SEND
                     (connections, attacker-known
                                    ; inputs
                                    )
                   else if attacker-step
                     = attacker-suppress then ATTACKER-SUPPRESS
                        (connections; inputs)
                   else ATTACKER-DISCONNECT
                     (; connections, inputs);
        };
end enrich
≪ENDFILE≫
≪ENDDEFINE≫

≪DEFINE generateFakeCardletSimple
  FOR uml::Model≫
≪FILE "FakeCardletSimple/specification"≫
enrich inputs, Generate, fakecardletstep with
  procedures
      FAKE-CARDLET-SEND-RESPONSE agent
        ≪convertSymbol("times")≫
        connections ≪convertSymbol("times")≫ message :
        (agent ≪convertSymbol("implies")≫ ports
        ≪convertSymbol("implies")≫ messagelist)
        nonfunctional indeterministic;
      SIMPLE-FAKE-CARDLET agent
        ≪convertSymbol
          ("times")≫ connections : (agent
            ≪convertSymbol
              ("implies")≫ attackerdataset)
                ≪convertSymbol("times")≫
              (agent ≪convertSymbol("implies")≫ ports
              ≪convertSymbol("implies")≫ messagelist)
              nonfunctional indeterministic;
   variables
      rem-agent : agent;
      rem-port, local-port, inport : ports;
      connections : connections;
      inputs : agent
        ≪convertSymbol("implies")≫ ports
          ≪convertSymbol
            ("implies")≫ messagelist;
      fake-cardlet-known : agent
        ≪convertSymbol
          ("implies")≫ attackerdataset;
declaration
   FAKE-CARDLET-SEND-RESPONSE
   : FAKE-CARDLET-SEND-RESPONSE
     (ag, connections, msg; var inputs)
       { choose conn, local-port
                with conn
```

```
                ≪convertSymbol
                  ("in")≫ connections
                    ≪convertSymbol
                      ("and")≫ is-endpoint
                        (ag
                          ≪convertSymbol
                            ("odot"
                              )≫ local-port
                                , conn)
            in {
                  inputs(other-endpoint(ag
                    ≪convertSymbol
                      ("odot"
                        )≫ local-port,conn)
                          .agent)(other-endpoint
                            (ag
                              ≪convertSymbol
                                ("odot"
                                  )≫ local-port
                                    ,conn).port)
              :=    inputs(other-endpoint(ag
                ≪convertSymbol
                  ("odot")≫ local-port,conn)
                    .agent)(other-endpoint(ag
                      ≪convertSymbol
                        ("odot"
                          )≫ local-port,conn)
                            .port)+ msg '
            } ifnone skip
    };
SIMPLE-FAKE-CARDLET
: SIMPLE-FAKE-CARDLET (ag, connections
                       ; var fake-cardlet-known
                         , inputs
                       )
    {
       if input-available(ag, inputs)
       then choose  inport
            with inputs(ag)(inport)
              ≪convertSymbol
                ("notequals")≫ []
            in {
                  fake-cardlet-known
                    (ag) :=     fake-cardlet-known(ag)
                    ≪convertSymbol("integral")≫
                    + inputs(ag)(inport).first ;
                  inputs(ag)(inport) := inputs(ag)
                    (inport) .rest ;
                  choose  msg
                  with (fake-cardlet-known(ag)
                    ≪convertSymbol
                      ("grgr")≫ amessage(msg))
                  in FAKE-CARDLET-SEND-RESPONSE
                    (ag, connections, msg; inputs)
```

```
                    }
        };
end enrich
≪ENDFILE≫
≪ENDDEFINE≫

≪DEFINE generateLists FOR Model≫
≪LET this.getListAssociations() AS atts1≫
≪LET getListPropertyofOCLConstraints()
  AS atts2≫
≪LET atts1.addAll(atts2) AS atts≫
≪FOREACH atts AS a≫
≪logging("Ausgabe der ListAssoziationen: "
  + a)≫
≪LET a.type.name AS ty≫
≪LET "listof"+ty AS lity≫
≪LET "a_listof"+ty AS livar≫
≪FILE lity+"/specification"≫
actualize list-perm with message by morphism
elem ≪convertSymbol("implies")≫
  ≪a.type.name≫; list
    ≪convertSymbol("implies")≫ listof
      ≪a.type.name≫;
a ≪convertSymbol("implies")≫
  ≪varname(ty)≫; a0
    ≪convertSymbol("implies")≫
      ≪varname(ty)≫0;
b ≪convertSymbol("implies")≫
  ≪varname(ty)≫1; c
    ≪convertSymbol("implies")≫
      ≪varname(ty)≫2;
x ≪convertSymbol("implies")≫
  ≪livar≫; x0
    ≪convertSymbol("implies")≫
      ≪livar≫0;
x1 ≪convertSymbol("implies")≫
  ≪livar≫1; x2
    ≪convertSymbol("implies")≫
      ≪livar≫2;
y ≪convertSymbol("implies")≫
  ≪livar≫3; y0
    ≪convertSymbol("implies")≫
      ≪livar≫4;
y1 ≪convertSymbol("implies")≫
  ≪livar≫5; y2
    ≪convertSymbol("implies")≫
      ≪livar≫6;
z ≪convertSymbol("implies")≫
  ≪livar≫7; z0
    ≪convertSymbol("implies")≫
      ≪livar≫8;
z1 ≪convertSymbol("implies")≫
  ≪livar≫9; z2
    ≪convertSymbol("implies")≫
```

```
      ≪livar≫10;
end actualize
≪ENDFILE≫
≪FILE lity+"funs/specification"≫
enrich ≪lity≫ with
functions
       add : ≪lity≫, ≪ty≫ ->
         ≪lity≫;
       size : ≪lity≫ -> int;
         at : ≪lity≫, int ->
           ≪ty≫; comment: zero based
             , unspecified if too large or negative.;
    remove : ≪lity≫, int ->
      ≪lity≫; comment
        : remove element at given index (zero based)
          . Does nothing if index is too large or negative
            .;
    remove : ≪lity≫, ≪ty≫ ->
      ≪lity≫; comment
        : removes only one occurrence, the first
          , of the given element.;
≪IF a.type.hasMapKey()-≫
       get : ≪lity≫,
         ≪a.type.getMapKey().type
           .translateType()≫ -> ≪ty≫;
       set : ≪lity≫, ≪ty≫ ->
         ≪lity≫;
≪ENDIF-≫
predicates
        hasFree : ≪lity≫;
        contains : ≪lity≫, ≪ty≫;
≪IF a.type.hasMapKey()-≫
       containsKey : ≪lity≫,
         ≪a.type.getMapKey().type
           .translateType()≫;
≪ENDIF-≫
axioms
hasFree-def : hasFree(≪livar≫)
  ≪IF a.upper > 1≫ <-> #
    ≪livar≫ < ≪a.upper≫
      ≪ENDIF≫; used for : s,ls;
size-def : size(≪livar≫) = n\impliesi(#
  ≪livar≫); used for : s,ls;
contains-def : contains(≪livar≫,
  ≪varname(ty)≫) <->
    ≪varname(ty)≫
      ≪convertSymbol("in")≫
        ≪livar≫; used for : s,ls;
at-index : at(≪livar≫, i) =
  ≪livar≫[i\impliesn
    (i
      )]; used for : s,ls; comment: zero based
        , unspecified if too large.;
add-def : add(≪livar≫,
```

```
  ≪varname(ty)≫) = ≪livar≫ +
    ≪varname(ty)≫; used for : s,ls;
remove-index-ok : 0
  ≪convertSymbol("le")≫ i -> remove(
    ≪livar≫, i) =
      ≪livar≫ -1l i\impliesn
        (i); used for : s,ls;
remove-index-negative : i < 0 -> remove(
  ≪livar≫, i) =
    ≪livar≫; used for : s,ls;
remove-element : remove(≪livar≫,
  ≪varname(ty)≫) = ≪livar≫ -1l
    ≪varname
      (ty
        )≫; used for : s
          ,ls; comment: removes only one occurrence
            , the first.;
≪IF a.type.hasMapKey()-≫
≪LET a.type.getMapKey() AS k≫
containsKey-empty : not containsKey([],
  ≪k.type.varname4type()≫); used for : s,ls;
containsKey-found :
  ≪k.type.varname4type()≫ =
    ≪varname(ty)≫.
      ≪k.name≫ -> containsKey(
        ≪varname(ty)≫ ' +
          ≪livar≫,
            ≪k.type
              .varname4type()≫); used for : s,ls;
containsKey-rec    : not
  ≪k.type.varname4type()≫ =
    ≪varname(ty)≫.
      ≪k.name≫ -> (containsKey(
        ≪varname(ty)≫ ' +
          ≪livar≫,
            ≪k.type.varname4type()≫)
                          <-> containsKey(
                            ≪livar≫,
                              ≪k.type
                                .varname4type
                                  ()≫)
                                    ); used for : s
                                      ,ls;
get-found : ≪k.type.varname4type()≫ =
  ≪varname(ty)≫.≪k.name≫ -> get(
    ≪varname(ty)≫ ' + ≪livar≫,
      ≪k.type.varname4type()≫) =
        ≪varname(ty)≫; used for : s,ls;
get-rec : not ≪k.type.varname4type()≫ =
  ≪varname(ty)≫.≪k.name≫ -> get(
    ≪varname(ty)≫ ' + ≪livar≫,
      ≪k.type.varname4type()≫)
                      = get(≪livar≫,
                        ≪k.type
```

```
                                .varname4type
                                  ()≫); used for : s,ls;
set-empty : set([], ≪varname(ty)≫) =
  ≪varname(ty)≫ '; used for : s,ls;
set-found : ≪varname(ty)≫.
  ≪k.name≫ = ≪varname(ty)≫0.
    ≪k.name≫ -> set(
      ≪varname(ty)≫0 ' + ≪livar≫,
        ≪varname(ty)≫)
          = ≪varname(ty)≫ ' +
            ≪livar≫; used for : s,ls;
set-rec   : not ≪varname(ty)≫.
  ≪k.name≫ = ≪varname(ty)≫0.
    ≪k.name≫ -> set(
      ≪varname(ty)≫0 ' + ≪livar≫,
        ≪varname(ty)≫)
                      =
                        ≪varname(ty)≫0 '
                          + set(≪livar≫,
                            ≪varname
                              (ty)≫
                                ); used for : s,ls;
≪ENDLET≫
≪ENDIF-≫
end enrich
≪ENDFILE≫
≪ENDLET≫≪ENDLET≫≪ENDLET≫
≪ENDFOREACH≫
≪ENDLET≫
≪ENDLET≫
≪ENDLET≫
≪ENDDEFINE≫
```

Chapter 10

Transformations for update operations

```
≪IMPORT uml≫
≪EXTENSION templates::getter≫
≪EXTENSION templates::parse≫

≪DEFINE Root FOR uml::Model≫
 ≪EXPAND generateFromClassDiagram≫
≪ENDDEFINE≫

≪DEFINE generateFromClassDiagram FOR Model≫
 ≪EXPAND generateAccessFuns FOR this≫
 ≪EXPAND generateMessageAccessFuns FOR this≫
≪ENDDEFINE≫

≪DEFINE generateAccessFuns FOR Model≫
≪LET this.getNormalClasses() AS classes≫
≪FILE "accessfuns/specification"≫
enrich messageaccessfuns with
≪LET classes.ownedAttribute AS attg≫
≪IF ! attg.isEmpty–≫
functions
≪FOREACH attg AS att–≫
. .≪att.name≫:= . :
  ≪att.class.name≫
    ≪convertSymbol("times")–≫
 ≪IF att.isCryptoAtt()≫
   ≪att.translateCryptoType()≫≪ELSE–≫
 ≪att.type.translateType()≫
   ≪ENDIF≫
     ≪convertSymbol("implies")≫
       ≪att.class.name≫;
≪ENDFOREACH≫
axioms
≪FOREACH attg AS att–≫
(≪att.class.varname()≫.
  ≪att.name≫:=
    ≪att.translateToVariable()≫) = mk
```

```
      ≪att.class.name≫(
        ≪att.class
          .calculateArgumentsAccessFuns(att)≫);
≪ENDFOREACH≫
≪ENDIF≫≪ENDLET≫
end enrich
≪ENDFILE≫
≪ENDLET≫
≪ENDDEFINE≫

≪DEFINE generateMessageAccessFuns FOR Model≫
≪LET this.getAllMessageAndUserMessageClasses()
  AS classes≫
≪FILE "messageaccessfuns/specification"-≫
enrich Attacker with
functions
≪LET classes.ownedAttribute
  .removeNameAndTypeDuplicates() AS attg-≫
≪FOREACH attg AS att-≫
. .≪att.name≫:= . : message
  ≪convertSymbol("times")≫
    ≪IF att.isCryptoAtt()≫
      ≪att.translateCryptoType()≫
        ≪ELSE≫
          ≪att.type.translateType()≫
            ≪ENDIF≫
              ≪convertSymbol
                ("implies")≫ message;
≪ENDFOREACH-≫
≪ENDLET-≫
axioms
≪LET classes.ownedAttribute AS attg≫
≪FOREACH attg AS att-≫
( is≪att.class.name≫(msg)
  ≪convertSymbol("implies")≫ (msg.
    ≪att.name≫:=
      ≪att.translateToVariable()≫ = mk
        ≪att.class.name≫(
          ≪att.class
            .calculateArgumentsAccessFuns(att)≫)));
≪ENDFOREACH-≫
≪ENDLET-≫
end enrich
≪ENDFILE≫
≪ENDLET≫
≪ENDDEFINE≫
```

Chapter 11

Transformations for Manual Methods

```
«IMPORT uml»
«EXTENSION templates::getter»
«EXTENSION templates::parse»

«DEFINE Root FOR uml::Model»
«EXPAND generateManualFuns»
«ENDDEFINE»

«DEFINE generateManualFuns FOR Model»
«FILE "ManualFuns/specification"-»
«LET this.getAllManualOperations() AS manops»
«LET manops.select(o|o.ownedParameter.select(e|e
  .direction.toString()=="in" || e.direction
    .toString()=="inout").size == 0) AS constants»
«LET manops.select(o|o.ownedParameter.select(e|e
  .direction.toString()=="in" || e.direction
    .toString()=="inout").size != 0) AS predfuns»
«LET predfuns.select(e|e.getReturnResult().type
  .translateType()!="bool") AS funs»
«LET predfuns.select(e|e.getReturnResult().type
  .translateType()=="bool") AS preds»
enrich message
 with
«IF constants.size != 0»
constants
«FOREACH constants AS op-»
  «op.name» :
    «op.getReturnResult().type
      .translateType()»;«ENDFOREACH»
«ENDIF-»
«IF funs.size != 0»
functions
«FOREACH funs AS op-»
«LET op.getReturnResult().type.translateType()
  AS returnType-»
 «op.name» :
```

```
    «FOREACH op.ownedParameter.select(e|e.direction
      .toString()=="in" || e.direction
        .toString()=="inout")
          AS inoutParam SEPARATOR ","-»
            «inoutParam.type
              .translateType()-»
                «ENDFOREACH-» ->
                  «returnType»;
«ENDLET-»
«ENDFOREACH-»
«ENDIF»
«IF preds.size != 0»
predicates
«FOREACH preds AS op-»
 «op.name» :
    «FOREACH op.ownedParameter.select(e|e.direction
      .toString()=="in" || e.direction
        .toString()=="inout")
          AS inoutParam SEPARATOR ","-»
            «inoutParam.type
              .translateType()-»
                «ENDFOREACH-»;
«ENDFOREACH-»
«ENDIF»
end enrich
«ENDLET»«ENDLET»«ENDLET»
  «ENDLET»«ENDLET»
«ENDFILE»
«ENDDEFINE»
```

Chapter 12

Auxiliary Xtend operations for the deployment diagram

```
import uml;
extension templates::getter;

List[CommunicationPath] removeTypeDuplicatesCommPath
  (List[CommunicationPath] l) :
 ( (!l.isEmpty)
   ? (    (l.withoutFirst().contains(l.first()))
     ? (removeTypeDuplicatesCommPath(l.withoutFirst
       ())) : (removeTypeDuplicatesCommPath
         (l.withoutFirst()).add(l.first()))   ) :
           ({}) );
cached Boolean existsFakeCardlet(Model mo) :
 (mo.getAllNodes().exists(e|e.hasStereotype
   ("SecureMDD::forgeable")));

List[Node] getAllNodes(Model m) :
 m.getDeploymentDiagram().packagedElement.typeSelect
   (Node);

List[String] toInportStrings(List[Property] p) :
 (p.isEmpty)
   ?({}):({"inport = " + p.first().name}.addAll
     (p.withoutFirst().toInportStrings()));

String toInportString(List[String] l) :
 (l.isEmpty)
   ?(""):(l.first() + ((l.withoutFirst().isEmpty)
     ?(""):(" " + convertSymbol
       ("or") + " " + l.withoutFirst().toInportString
         ())));

List[String] getTerminalPorts
  (uml::Model m, Class terminalclass) :
 let terminalnodes = m.getDeploymentDiagram
   ().packagedElement.typeSelect(Node).select
     (e|e.hasStereotype
```

```
        ("SecureMDD::Terminal") && e.name
          == terminalclass.name) :
 terminalnodes.getPorts().toInportStrings();

List[String] getCardletPorts
  (uml::Model m, Class scclass) :
 let cardletnodes = m.getDeploymentDiagram
   ().packagedElement.typeSelect(Node).select
     (e|e.hasStereotype
       ("SecureMDD::Smartcard") && e.name
         == scclass.name) :
 cardletnodes.getPorts().toInportStrings();

List[String] getServicePorts
  (uml::Model m, Class serviceclass) :
 let servicenodes = m.getDeploymentDiagram
   ().packagedElement.typeSelect(Node).select
     (e|e.hasStereotype
       ("SecureMDD::Service") && e.name
         == serviceclass.name) :
 servicenodes.getPorts().toInportStrings();
List[Property] getPorts(List[Node] nodes) :
  nodes.getCommunicationPaths
    ().removeTypeDuplicatesCommPath
      ().memberEnd.flatten();

Property getPort(Node n, CommunicationPath cp) :
 (n.getPorts().select(e|e.association == cp).first());
List[Property] getPorts(Node n) :
 n.getCommunicationPaths
   ().removeTypeDuplicatesCommPath().memberEnd.select
     (e|e.type.name!=n.name);

String portNameWithClass(Property p) :
 ( p.opposite.type.name + "2" + p.type
   .name + "DefaultPort" );
private String portNameWithNodeName(Property p) :
 let s1 = p.getOtherEnd().type.name :
 let s2 = p.type.name :
 ( s1 + "2" + s2 + "DefaultPort");
String portName(Property p) : (! ( p.name
  == null || p.name
    == "" ) ? p.name : p.portNameWithNodeName() );

List[String] portNames(Node n) :
 n.getPorts().portName().removeDuplicates();

List[String] getAllPortNames(Model m) :
 m.getAllNodes().getPorts().portName
   ().removeDuplicates();

String getAnyPort
  (String component1, String component2, Model m) :
 m.getDeploymentDiagram().getNode
```

```
    (component1).getAssociations().typeSelect
      (CommunicationPath).removeTypeDuplicatesCommPath
        ().memberEnd.select(e|e.class.name
          == component2).opposite.first().name;

String getAnyPortSend
  (String component1, String component2, Model m) :
 m.getDeploymentDiagram().getNode
    (component1).getAssociations().typeSelect
      (CommunicationPath).removeTypeDuplicatesCommPath
        ().memberEnd.select(e|e.class.name
          == component2).first().name;

Node getNode(Package depl, String nodename) :
 (depl.packagedElement.typeSelect(Node).select
    (e|e.name == nodename));
```

Chapter 13

Auxiliary Xtend operations

```
import uml;
String convertSymbol
  (String symbolname) : JAVA swt.KIVSymbolConverter
    .convertSymbol(java.lang.String);
List[Property] getListPropertyofOCLConstraints
  () : JAVA swt.Parser.getListPropertyofOCLConstraints
    ();
String toLower(String s) : JAVA swt.Util.getLower
  (java.lang.String);
String toFirstLower
  (String s) : JAVA swt.Util.getFirstLower
    (java.lang.String);
Void logging(String s) : JAVA swt.Util.logging
  (java.lang.String);
Void debug(String s) : JAVA swt.Util.debug
  (java.lang.String);

String ppList(List[String] strs, String between) :
 (strs.size == 0 ? "" :
 (strs.size == 1 ? strs.first() :
 strs.first() + between + ppList(strs.withoutFirst
   (), between)
 ));

String getSecond(List[String] ls) :
 ls.withoutFirst().first();

List[Class] getAllMessageClasses(Model mo) :
 mo.getClassDiagramClasses().select(c|
   (! c.isAbstract) && c.isMessage());

List[Class] getAllMessageAndUserMessageClasses
  (Model mo) :
 mo.getClassDiagramClasses().select(c|
   (! c.isAbstract) && (c.isMessage()
     || c.isUserMessage()));
List[Class] getAllUserMessageClasses(Model mo) :
 mo.getClassDiagramClasses().select(e|e.isUserMessage
   () && !e.isAbstract);
```

```
Boolean onlyTrivialUserClasses(Model mo) :
 (let cs = mo.getAllUserMessageClasses() :
  cs.forAll(c | c.ownedAttribute.forAll
    (a | a.type.name == "Number"
         || a.type.name == "String"
         || a.type.name == "Boolean")));

Package getClassDiagram(Model m) :
 m.packagedElement.typeSelect(Package).select
   (e|e.hasStereotype
     ("SecureMDD::ClassDiagram")).first();
cached Package getSecurityDatatypes(Model m) :
 m.packagedElement.typeSelect(Package).select
   (e|e.hasStereotype("SecureMDD::SecurityDatatypes"));
cached List[Class] getClassDiagramClasses(Model m) :
 m.getClassDiagram().ownedElement.typeSelect(Class);
cached List[Enumeration] getClassDiagramEnumerations
  (Model m) :
 m.getClassDiagram().ownedElement.typeSelect
   (Enumeration);
cached List[Class] getSecurityDatatypesClasses
  (Model m) :
 m.getSecurityDatatypes().ownedElement.typeSelect
   (Class);
Class getClass
  (Model m, String n) : m.getClassDiagramClasses
    ().select(c|c.name==n).first();
cached Package getDeploymentDiagram(Model m) :
 m.packagedElement.typeSelect(Package).select
   (e|e.hasStereotype
     ("SecureMDD::DeploymentDiagram")).first();
cached Property getStateAttribute(Class this) :
 this.ownedAttribute.selectFirst(a|a.isStateAttribute
   ());

List[Operation] instanceOperations(Class this) :
 this.ownedOperation;
Boolean hasConstructor(Class this) :
 (let ops = this.instanceOperations() :
  ops.exists(op | op.name == this.name));

Operation getConstructor(Class this) :
 (let ops = this.instanceOperations() :
  ops.selectFirst(op | op.name == this.name)
  );
cached List[Property] getAttributesInCorrectOrder
  (Class this) :
 ( this.hasConstructor()
   ? this.getAttributesFromConstructor
     () : this.getGeneral().getAllAttributes().addAll
       (this.instanceAttributes()) );

List[Property] instanceAttributesFromAssociation
  (Class this) :
 this.ownedAttribute.select(e|e.association != null);
```

```
List[Property] instanceAttributesWithoutAssociation
  (Class this) :
 this.ownedAttribute.select(e|e.association == null);

List[Property] instanceAttributes(Class this) :
 this.instanceAttributesWithoutAssociation().addAll
   (this.instanceAttributesFromAssociation());

List[Property] getAttributesFromConstructor
  (Class this) :
 (let op = this.getConstructor() :
 (let params = op.ownedParameter :
  params.collect(p| this.getAttributeWithName(p.name))
 ));

Property getAttributeWithName(Class this, String n) :
 this.ownedAttribute.selectFirst(a|a.name == n);

List[String] removeDuplicates(List[String] l) :
 (l.isEmpty)
   ?(l):((l.withoutFirst().contains(l.first())
     ?(l.withoutFirst().removeDuplicates()):({l.first
       ()}.addAll(l.withoutFirst().removeDuplicates
         ())))));

List[Property] removeNameAndTypeDuplicates
  (List[Property] l):
 ( (!l.isEmpty)
   ? ((l.withoutFirst(). typeAndNameContains(l.first
     ()))
               ? (removeNameAndTypeDuplicates
                 (l.withoutFirst())) :
                 (removeNameAndTypeDuplicates
                   (l.withoutFirst()).add(l.first
                     ()))  )
               : ({}) );

List[Property] removeTypeDuplicates(List[Property] l) :
 ( (!l.isEmpty)
   ? ((l.withoutFirst(). typeContains(l.first()))
     ? (removeTypeDuplicates(l.withoutFirst())) :
       (removeTypeDuplicates(l.withoutFirst()).add
         (l.first()))  ) : ({}) );

Boolean typeAndNameContains
  (List[Property] l, Property p) :
  l.isEmpty ? false : ((l.first().name
    == p.name) && (l.first().translateCryptoType()
      == p.translateCryptoType()) ? true :
  l.withoutFirst().typeAndNameContains(p));

Boolean typeContains(List[Property] l, Property p) :
 JAVA swt.Util.typeContains
   (java.util.List, org.eclipse.uml2.uml.Property);
```

```
List[Property] calcListActualizations(List[Class] l) :
 ( let atts = (List[Property])(l.collect
   (c | c.ownedAttribute).flatten()) :
   atts.select(a| !a.isPrimitiveType
     () && !a.isPrimitiveTypeList() && a.isList
       ()).removeTypeDuplicates());

String getMK(Class c) :
 let attc = c.getAttributesInCorrectOrder() :
 ("mk" + c.name+((attc.isEmpty)
   ?(""):("("+ attc.generateArgumentString() +")"))) ;

String generateArgumentString(List[Property] lp) :
 let newnams = lp.getUniqueVariables4Properties() :
 newnams.ppList(", ");

List[String] getUniqueVariables4Properties
  (List[Property] lp) :
 let nams = lp.translateToVariable() : nams.makeUnique
   ();
List[String] makeUnique
  (List[String] todo) : todo.makeUniqueRec({});

List[String] makeUniqueRec
  (List[String] todo, List[String] done) :
 (todo.isEmpty ? done :
 (let s = todo.first() :
 (let news = (done.contains(s)
   ? s.newName(0, done) : s) :
 (todo.withoutFirst().makeUniqueRec(
   (List[String])done.add(news))
 ))));

String newName(String s, Integer c, List[String] li) :
  (li.contains(s + c) ? s.newName(c + 1, li) : s + c);

List[String] reverse(List[String] li) :
 li.isEmpty
   ? {} : li.withoutFirst().reverse().add(li.first());

String calculateArgumentsAccessFuns
  (Class c, Property att) :
 let allatts = c.getAttributesInCorrectOrder() :
 ( (allatts.isEmpty)
   ?(""): (allatts.calcArgStringList(c,att)) );

String calcArgStringList
  (List[Property] atts, Class c, Property att) :
 ((atts.isEmpty)
   ?(""):(atts.first().calcArgString(c,att) + (
     (atts.withoutFirst().isEmpty)
       ?(""):(", " + atts.withoutFirst
         ().calcArgStringList(c,att)))));
```

```
String calcArgString
  (Property atts, Class c, Property att) :
 (atts.name
   == att.name)
     ?(att.translateToVariable()):(calcClassInstance
       (c) + "." + atts.name);

String calcClassInstance(Class c) :
 ((c.isMessage()
   || c.isUserMessage()) ? "msg" : c.varname());

Boolean isCryptoAtt(Property p) :
 (p.type.name == "Nonce" ||
  p.type.name == "Secret" ||
  p.type.name == "Key" ||
  p.type.name == "SymmKey" ||
  p.type.name == "PrivateKey" ||
  p.type.name == "PublicKey" ||
  p.type.name == "SignedData" ||
   p.type.name == "HashedData" ||
   p.type.name == "MACData" ||
   p.type.name == "EncData" ||
   p.type.name == "EncDataSymm" ||
   p.type.name == "EncDataAsymm" ||
  p.hasStereotype("SecureMDD::hashed") ||
  p.hasStereotype("SecureMDD::signed") ||
  p.hasStereotype("SecureMDD::encrypted") ||
  p.hasStereotype("SecureMDD::encryptedAsymm")
 );
String WRAP(Class c) : "wrap"+c.name;
String WRAP2
  (Class c, String s) : "wrap"+c.name + "2" + s;
String WRAP2
  (Type ty, String s) : "wrap"+ty.name + "2" + s;
String WRAP2
  (String str, String s) : "wrap"+str + "2" + s;

String getArguments(Class c) :
 let attg = c.getAttributesInCorrectOrder() :
 (attg.isEmpty
   ? "" : "(" + attg.getArgumentString() + ")");

String getArgumentString(Property p) :
 ". ." + p.name + " : " +  (p.hasHashedStereotype()
   ? "HashedData" :
     (p.hasEncryptedStereotype() ? "EncData" :
     (p.hasEncryptedAsymmStereotype() ? "EncDataAsymm" :
     (p.hasSignedStereotype() ? "SignedData" :
      p.translateType()))));

String getArgumentString(List[Property] ps) : ppList
  (ps.collect(p|p.getArgumentString()), ";");
List[Property] getListAssociations(Model m) :
 let classes = m.getClassDiagramClasses() :
  classes.getListAssociations4();
```

```
List[Property] getListAssociations4(List[Class] cs) :
 (cs.collect(e|e.getListAssociations3()).flatten());

List[Property] getListAssociations2(List[Class] cs) :
 (cs.size == 0 ? { } :
  (let c = cs.first() : cs.withoutFirst
    ().getListAssociations2().addAll
      (c.getListAssociations3()) ));
List[Property] getListAssociations3(Class c) :
 (let atts = c.ownedAttribute : atts.select
   (a | a.upper != 1));
cached List[Operation] getAllManualOperations
  (Model mo) :
 mo.getClassDiagramClasses().ownedOperation.select
   (e|e.hasManualStereotype());
cached List[Activity] getAllActivities(Model m) :
 m.ownedElement.typeSelect(Package)
 .select(e|e.hasStereotype
   ("SecureMDD::ActivityDiagram"))
 .ownedElement.typeSelect(Activity);
cached List[Enumeration] getAllEnumerations(Model m) :
 m.getClassDiagram().packagedElement.typeSelect
   (Enumeration);
cached List[Class] getAllTerminals(Model m) :
 (m.getClassDiagramClasses().select(e|e.isTerminal
   () && !e.isAbstract));
cached List[Class] getAllSmartcards(Model m) :
 (m.getClassDiagramClasses().select(e|e.isSmartCard
   () && !e.isAbstract));
cached List[Class] getAllServices(Model m) :
 (m.getClassDiagramClasses().select(e|e.isService
   () && !e.isAbstract));
cached List[Class] getAllUsers(Model m) :
 (m.getClassDiagramClasses().select(e|e.isUser
   () && !e.isAbstract));
cached List[Class] getAllConstants(Model m) :
 m.getClassDiagramClasses().select(e|e.isConstants());
cached List[Class] getNormalClasses(Model m) :
 m.getClassDiagramClasses().select(c|c.isNormalClass
   ());
cached List[Class] getUsermessages(Model mo) :
 mo.getClassDiagramClasses().select
   (e|e.hasUsermessageStereotype());
cached List[Class] getPlainClasses(Model mo) :
 mo.getClassDiagramClasses().select
   (e|e.hasPlainDataStereotype());
cached List[Class] getHashClasses(Model mo) :
 mo.getClassDiagramClasses().select
   (e|e.hasHashDataStereotype());
cached List[Class] getSignClasses(Model mo) :
 mo.getClassDiagramClasses().select
   (e|e.hasSignDataStereotype());
List[Class] getGeneral(Classifier this) : this.general;
```

```
Boolean isNormalClass(Class c) :
 (! c.isSmartCard()) &&
 (! c.isTerminal()) &&
 (! c.isService()) &&
 (! c.isUser()) &&
 (! c.isMessage()) &&
 (! c.isUserMessage()) &&
 (! c.isConstants()) &&
 (! c.hasManualStereotype()) &&
 (! c.isEnumeration());
Boolean isEnumeration(Class c) : c.metaType
  == Enumeration;

Boolean isConstants(Class this) :
 this.hasStereotype("SecureMDD::Constant");

Boolean isSmartCard(Class this) :
 this.hasStereotype("SecureMDD::Smartcard") ||
 this.general.typeSelect(Class).exists
   (c|c.hasStereotype("SecureMDD::Smartcard"));

Boolean isTerminal(Class this) :
 this.hasStereotype("SecureMDD::Terminal") ||
 this.general.typeSelect(Class).exists
   (c|c.hasStereotype("SecureMDD::Terminal"));

Boolean isService(Class this) :
 this.hasStereotype("SecureMDD::Service") ||
 this.general.typeSelect(Class).exists
   (c|c.hasStereotype("SecureMDD::Service"));
Boolean isMessage(Class this) :
 this.hasMessageStereotype() ||
 this.general.typeSelect(Class).exists
   (c|c.hasMessageStereotype());

Boolean isUserMessage(Class this) :
 this.hasUsermessageStereotype() ||
 this.general.typeSelect(Class).exists
   (c|c.hasUsermessageStereotype());

Boolean isPlainData(Class this) :
 this.hasStereotype("SecureMDD::PlainData");

Boolean isFakeCardlet(Class this) :
 this.hasStereotype("SecureMDD::forgeable");

Boolean isUser(Class this) :
 this.hasStereotype("SecureMDD::User");

Boolean isSignData(Class this) :
 this.hasStereotype("SecureMDD::SignData");
Boolean isList(Property p) : p.upper > 1;

Boolean isStateAttribute(Property a) :
 a.hasStereotype("SecureMDD::status");
```

```
Boolean isHashed(Property this) :
 this.hasStereotype("SecureMDD::hashed");

Boolean isEncrypted(Property this) :
 this.hasStereotype("SecureMDD::encrypted");
Boolean isEncryptedAsymm (Property this) :
 this.hasStereotype("SecureMDD::encryptedAsymm");

Boolean isSigned(Property this) :
 this.hasStereotype("SecureMDD::signed");

Boolean isSecurityDatatype(Class this) :
 this.package.hasStereotype
   ("SecureMDD::SecurityDatatypes");

Boolean isSecurityDatatype(Type this) :
 (this.metaType
   == Class && ((Class) this).isSecurityDatatype());

Boolean isStateful(Class server):
 let stereotype=server.getAppliedStereotype
   ("SecureMDD::Service"):
 stereotype
   ==null
     ? server.isGeneralStateful() :
       (let b= server.getValue(stereotype,"stateful"):
 b==true ? true : server.isGeneralStateful());

Boolean isGeneralStateful(Class server):
 let gStereotype=server.getGeneralServer
   ().getAppliedStereotype("SecureMDD::Service"):
 gStereotype
   ==null
     ? false : (let b= server.getGeneralServer
       ().getValue(gStereotype,"stateful"):
 b==true ? true : false);

Class getGeneralServer(Classifier this) :
 this.getGeneral().select(e|e.hasStereotype
   ("SecureMDD::Service")).first() ;

Boolean constantsExist(Model m) :
 m.getClassDiagramClasses().exists(e|e.hasStereotype
   ("SecureMDD::Constant"));

Boolean statusExists(Model m) :
 let clas = m.getAllSmartcards().addAll
   (m.getAllTerminals()).addAll(m.getAllServices()) :
 clas.ownedAttribute.exists(e|e.hasStereotype
   ("SecureMDD::status"));

List[Property] getCertificateAttributes(Class c) :
 (let certData = (Class)(c.ownedAttribute.first
   ().type) :
  certData.getAttributesInCorrectOrder()
 );
```

```
Property getCertificateSignatureAttribute(Class c) :
 (c.ownedAttribute.first().hasSignedStereotype()
   ? c.ownedAttribute.first
     () : c.ownedAttribute.getSecond());

Property getCertificateDataAttribute(Class c) :
 (c.ownedAttribute.first().hasSignedStereotype()
   ? c.ownedAttribute.getSecond
     () : c.ownedAttribute.first());
Class getCertificateDataClass(Class c) : (Class)
  (c.ownedAttribute.first().type);

String toString(uml::String this) :
 this;

String translateTypeToSpecName(String s) :
 (s == "Number" ? "int-pair" :
 (s == "String" ? "string-data" :
 (s == "Boolean" ? "bool" :
 ((s == "SymmKey" || s == "PublicKey" || s
   == "PrivateKey" || s == "EncDataSymm" || s
     == "EncDataAsymm" || s == "SignedData" || s
       == "HashedData" )? "SecurityOperations" :
 (s == "Secret" ? "Secret" :
 (s == "Nonce" ? "Nonce" :
 "data"))))));

String translateCryptoType(Property p) :
 ((p.hasStereotype("SecureMDD::hashed"))
   ?("HashedData"):
 ((p.hasStereotype("SecureMDD::signed"))
   ?("SignedData"):
 ((p.hasStereotype("SecureMDD::encrypted"))
   ?("EncData"):
 ((p.hasStereotype("SecureMDD::encryptedAsymm"))
   ?("EncDataAsymm"):
 ((p.type.name == "EncDataSymm")? "EnData" :
 ((p.type.name == "EncDataAsymm")? "EncDataAsymm" :
 translateType(p)))))));

String translateType(String s) :
 (s == "Number" ? "int" :
 (s == "String" ? "string" :
 (s == "Boolean" ? "bool" :
 s)));

String translateType(Type this) :
 let n =  this.name : n.translateType();

String translateType(Property this) :
 let n = this.type.name :
 let ty1 = translateType(n) :
 let rest = (this.isList() ? "listof" : "") :
 rest + ty1
```

```
 ;
Boolean isPrimitiveType
  (Property this) : this.type.isPrimitiveType();
Boolean isPrimitiveType
  (Type this) : this.name.isPrimitiveType();

Boolean isPrimitiveType(String ty) :
    ty == "Number" || ty == "String" || ty
      == "Boolean";

Boolean isPrimitiveTypeList(Property p) :
 (let t = p.translateType() : (t == "intlist" || t
   == "stringlist" || t == "boollist") );
String varname(NamedElement this) : varname(this.name);
String varname(String this) : "a_" + this;
String varname4type(String s) :
 (s == "int") ? "i" :
 ((s == "nat")? "n" :
 ((s == "string")? "str" :
 ((s == "bool")? "boolvar": varname(s) )));
String varname4type(Type ty) : varname4type
  (ty.translateType());

String translateToVariable(Property p) :
 (p.translateType() == "int")?("i"):
 ((p.translateType() == "nat")?("n"):
 ((p.translateType() == "string")?("str"):
 ((p.translateType() == "bool")?("boolvar"):
 ((p.hasHashedStereotype())?(varname("HashedData")):
 ((p.hasEncryptedStereotype())?(varname("EncData")):
 ((p.hasEncryptedAsymmStereotype())
   ?(varname("EncDataAsymm")):
 ((p.hasSignedStereotype())
   ?(varname("SignedData")):(varname(p.translateType
     ()))))))))));

Boolean hasStaticAttributes(Class this) :
 this.isConstants();

Boolean hasPlainDataStereotype(Element e) :
 e.hasStereotype("SecureMDD::PlainData");

Boolean hasHashDataStereotype(Element e) :
 e.hasStereotype("SecureMDD::HashData");

Boolean hasUsermessageStereotype(Element e) :
 e.hasStereotype("SecureMDD::Usermessage");

Boolean hasMessageStereotype(Element e) :
 e.hasStereotype("SecureMDD::Message");

Boolean hasSignDataStereotype(Element e) :
 e.hasStereotype("SecureMDD::SignData");
```

```
Boolean hasHashedStereotype(Element e) :
 e.hasStereotype("SecureMDD::hashed");

Boolean hasSignedStereotype(Element e) :
 e.hasStereotype("SecureMDD::signed");

Boolean hasEncryptedStereotype(Element e) :
 e.hasStereotype("SecureMDD::encrypted");

Boolean hasEncryptedAsymmStereotype(Element e) :
 e.hasStereotype("SecureMDD::encryptedAsymm");

Boolean hasManualStereotype(Element e) :
 e.hasStereotype("SecureMDD::Manual");

Boolean hasCertificateStereotype(Class c) :
 c.hasStereotype("SecureMDD::Certificate");

Boolean existsStereotype
  (List[Class] classes, String stereotypename) :
 (classes.exists(c|c.getAppliedStereotype
   (stereotypename) != null));

Boolean hasStereotype(Element this,String s) :
  this.getAppliedStereotype(s)!=null;
cached Boolean existsNonce(Model mo) :
 (mo.getClassDiagramClasses().ownedAttribute.exists
   (e|e.type != null && e.type.name == "Nonce"));
cached Boolean existsSecret(Model mo) :
 (mo.getClassDiagramClasses().ownedAttribute.exists
   (e|e.type != null && e.type.name == "Secret"));
cached Boolean existsSymmKey(Model mo) :
 (mo.getClassDiagramClasses().ownedAttribute.exists
   (e|e.type != null && e.type.name == "SymmKey" ));
cached Boolean existsASymmKey(Model mo) :
 (mo.getClassDiagramClasses().ownedAttribute.exists
   (e| e.type != null && (e.type.name
     == "PublicKey" || e.type.name == "PrivateKey")));
cached Boolean usesEncryption(Model mo) :
 (mo.getClassDiagramClasses().ownedAttribute.exists
   (e| e.hasEncryptedStereotype()));
cached Boolean usesAsymmEncryption(Model mo) :
 (mo.getClassDiagramClasses().ownedAttribute.exists
   (e| e.hasEncryptedAsymmStereotype()));
cached Boolean usesHashing(Model mo) :
 (mo.getClassDiagramClasses().ownedAttribute.exists
   (e| e.hasHashedStereotype()));
cached Boolean usesSigning(Model mo) :
 (mo.getClassDiagramClasses().ownedAttribute.exists
   (e| e.hasSignedStereotype()));
cached Boolean usesSymmKey
  (Model mo) : mo.usesEncryption()
    || mo.existsSymmKey();
cached Boolean usesASymmKey
```

```
  (Model mo) : mo.usesAsymmEncryption()
    || mo.existsASymmKey();
cached Boolean usesCertificates(Model mo) :
 mo.getClassDiagramClasses().exists
   (c | c.hasCertificateStereotype());

List[Class] getCertificateClasses(Model mo) :
 mo.getClassDiagramClasses().select
   (c | c.hasCertificateStereotype());
Boolean hasMapKey(Type ty) :
 ty.metaType == Class && ((Class)ty).hasMapKey();

Boolean hasMapKey(Class c) :
 c.ownedAttribute.exists(a|a.hasStereotype
   ("SecureMDD::key"));
Property getMapKey(Type ty) : ((Class)ty).getMapKey();
Property getMapKey(Class c) : c.ownedAttribute.select
  (a|a.hasStereotype("SecureMDD::key")).first();
```

Chapter 14

Auxiliary Xtend operations for parsing

```
import uml;

Void setContext(Model m) :
 JAVA swt.Parser.setContext
    (org.eclipse.uml2.uml.Model);

Void setCurrentClass(String c) :
 JAVA swt.Parser.setCurrentClass(java.lang.String);

String getASMAUXSpec() :
 JAVA swt.Parser.getASMAUXSpec();

String getASMVariablesDef() :
 JAVA swt.Parser.getASMVariablesDef();

String activities2ASM() :
 JAVA swt.Parser.activities2ASM();

String getSendReceiveClass(String s) :
 JAVA swt.Parser.getSendReceiveClass(java.lang.String);

String getPortString(String SendString) :
 JAVA swt.Parser.getPortOfSendAction(java.lang.String);

Boolean stringLess(String a, String b) :
 JAVA swt.Parser.stringLess
    (java.lang.String, java.lang.String);

String MELReceiveAction2ASM(String c) :
 JAVA swt.Parser.MELReceiveAction2ASM
    (java.lang.String);

String MELSendSignalAction2ASM
   (String sendstring, String port) :
 JAVA swt.Parser.MELSendSignalAction2ASM
    (java.lang.String, java.lang.String);
```

```
String MELAction2ASM(String s) :
 JAVA swt.Parser.MELAction2ASM(java.lang.String);

String MELGuard2ASM(String s) :
 JAVA swt.Parser.MELGuard2ASM(java.lang.String);

Void logging(String s) :
 JAVA swt.Parser.logging(java.lang.String);
Void  parseOCL(Class s) :
 JAVA swt.OCLParser.parse(org.eclipse.uml2.uml.Class);
Void  parse_eigen(Class s) :
 JAVA swt.OCLParser.parse_eigen
   (org.eclipse.uml2.uml.Class);

String getOCLConstraints(String str) :
 JAVA swt.Parser.getOCLConstraints(java.lang.String);
Void  evaluateOCL(Class s) :
 JAVA swt.OCLParser.evaluate
   (org.eclipse.uml2.uml.Class);

Void setOCLContext(Model m):
 JAVA swt.OCLParser.setContext
   (org.eclipse.uml2.uml.Model);
```

Bibliography

[1] M. Balser, W. Reif, G. Schellhorn, K. Stenzel, and A. Thums. Formal system development with KIV. In *Fundamental Approaches to Software Engineering*. Springer LNCS 1783, 2000.

[2] E. Börger and R. F. Stärk. *Abstract State Machines—A Method for High-Level System Design and Analysis*. Springer-Verlag, 2003.

[3] *Java Card 2.2.2 Application Programming Interfaces*, 2006. URL:http://www.oracle.com/technetwork/java/javacard/specs-138637.html.

[4] K. Katkalov, N. Moebius, K. Stenzel, M. Borek, and W. Reif. Model-Driven Testing of Security Protocols with SecureMDD. In *Fifth IFIP International Conference on New Technologies, Mobility and Security (NTMS 2012)*. IEEE XPlore, 2012.

[5] N. Moebius, K. Stenzel, H. Grandy, and W. Reif. SecureMDD: A Model-Driven Development Method for Secure Smart Card Applications. In *Workshop on Secure Software Engineering, SecSE, at ARES 2009*. IEEE Press, 2009.

[6] N. Moebius, K. Stenzel, and W. Reif. Formal Verification of Application-Specific Security Properties in a Model-Driven Approach. In *Proc. of International Symposium on Engineering Secure Software and Systems 2010*. Springer LNCS 5965, 2010.

[7] K. Stenzel, N. Moebius, and W. Reif. Formal verification of QVT transformations for code generation. In *Model Driven Engineering Languages and Systems, 14th International Conference, MODELS 2011*. Springer LNCS 6981, 2011.