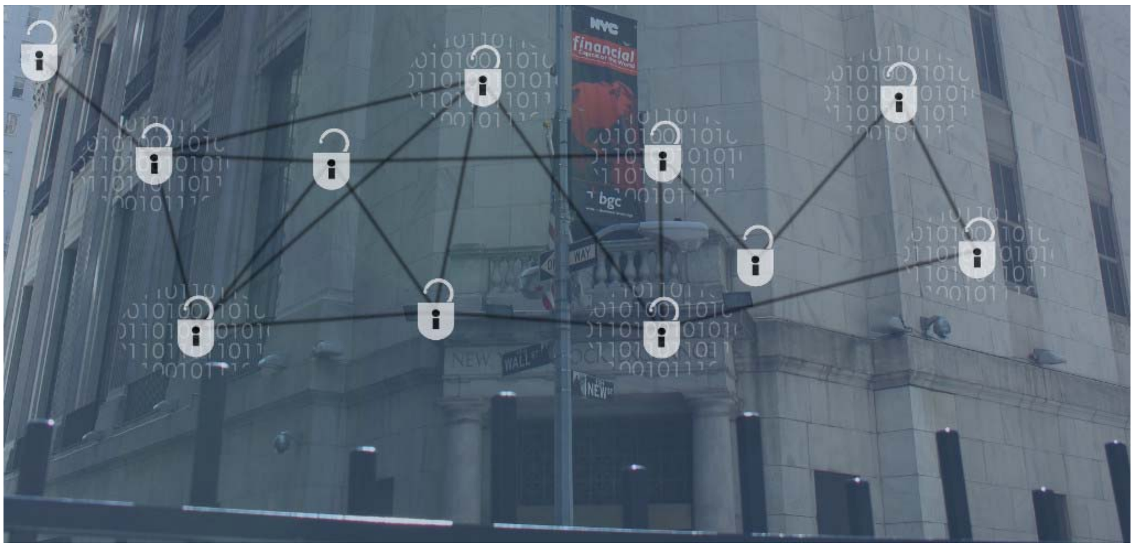
SYSTEMS OF SYSTEMS WITH SECURITY

Electrical and Computer Engineering
Technical Report ECE-TR-10



AARHUS UNIVERSITY
DEPARTMENT OF ENGINEERING

DATA SHEET

Title: Systems of Systems with Security
Subtitle: Electrical and Computer Engineering
Series title and no.: Technical report ECE-TR-10

Author: Rasmus Winther Lauritsen
Department of Engineering – Electrical and Computer Engineering,
Aarhus University

Internet version: The report is available in electronic format (pdf) at the Department of Engineering website http://www.eng.au.dk.

Publisher: Aarhus University©
URL: http://www.eng.au.dk

Year of publication: 2013 Pages: 34
Editing completed: March 2013

Abstract: In this report we present two case studies with Systems of Systems modelling. One model illustrates how Cryptographic parameter consistency can be checked using VDMPP for a System of Systems uses encryption to enforce Digital Right Management. The other model shows how a new formalism (CML) tailored specifically to Systems of Systems can express Multi-Party Computation protocol. The idea of using Canetti simulation proofs from Multi-Party computation as a model for refinement of models in CML is presented. Our goal is modest. We do not aim at proving security through refinement but to assists modellers/developers in maintaining security properties during refinement of a concept to designs.

Keywords: COMPASS, modelling, Vienna Development Method, COMPASS Modelling Language, digital rights management, multi party computation, systems of systems, systems

Supervisors: Peter Gorm Larsen, Ivan Damgaard
Financial support: Supported by the 7th framework programme project COMPASS grant 287829

Please cite as: Rasmus Winther Lauritsen, 2013. Systems of Systems with Security. Department of Engineering, Aarhus University. Denmark. 34 pp. - Technical report ECE-TR-10

Cover image: Rasmus Winther Lauritsen

ISSN: 2245-2087

SYSTEMS OF SYSTEMS WITH SECURITY

Rasmus Winther Lauritsen

Aarhus University, Department of Engineering

Abstract

In this report we present two case studies with Systems of Systems modelling. One model illustrates how Cryptographic parameter consistency can be checked using VDMPP for a System of Systems uses encryption to enforce Digital Right Management. The other model shows how a new formalism (CML) tailored specifically to Systems of Systems can express Multi-Party Computation protocol. The idea of using Canetti simulation proofs from Multi-Party computation as a model for refinement of models in CML is presented. Our goal is modest. We do not aim at proving security through refinement but to assists modellers/developers in maintaining security properties during refinement of a concept to designs.

Table of Contents

List of Figures

1 Introduction

With the presence of the Internet we have strong indications that successful systems will increasingly rely on interaction with a number of existing systems. Hence the systems engineering challenges are moving towards those of System of Systems (SoS) [Mai98, NLF$^+$13]. For Systems Engineering there exists well-founded Model Based Systems Engineering (MBSE) technologies for assisting the engineering process. These enable formal verification of desirable properties with a system through a model describing the system at an appropriate abstraction level. An interesting question is to which extent tools can support such MBSE also for SoS , which is the question that this work will evolve around. More precisely, the overall goal for this PhD is the following:

> *This project aims at developing models, tools and processes for the development of SoSs. In particular we will explore whether and to which extent security properties can be captured and analysed in this SoS setting.*

This report is mainly based on the work from [LL12] and on some ongoing work yet to be published. Additionally, as part of my PhD project I have also worked extensively on developing a type checker[1] for the CML formalism. This is however not covered in detail as it is out of the scope of this report. The report is structured as follows. The rest of this chapter gives an overview of the overall project which this work is part of. Chapter 2 introduces the different concepts that are needed. Then, in chapters 3 and 4 initial work on modelling security for two different kinds of systems is presented. Chapter 3 covers work modelling systems with DRM components, and chapter 4 is about modelling a secure protocol for multi-party computation. Finally, in Chapter 5 concrete suggestions for future directions are made.

SoS with Security

This work is a part of the European project *Comprehensive Modelling for Advanced System of Systems* (COMPASS[2]) under the Seventh Framework Programme. COMPASS's mission is to create tools to support developers working with SoS. The vision is to support an MBSE process that achieves a wide range of goals (for details of COMPASS see [COM11]) including but not limited to:

I Understanding a conceptual system through a diagrammatic view for easy comprehension by none experts (e.g. customers and users).

II Tool support for generating a *formal* model consistent with the diagrammatic model. (Possibly expert guided).

[1]CML type checker source – http://tinyurl.com/b4sydb3

[2] http://www.compass-research.eu/

III Validation of System design at an early stage of the engineering process. E.g. early proof of concept though model simulations of Use Cases and scenarios.

IV Validation of global functional as well as non-functional (e.g. security) properties through theorem proving, model checking and test automation.

V Tool support for step-wise refinement through the design phase and evolution of the SoS.

To give a feeling of how the above goals are going to be achieved an overview of COMPASS tasks and partners is provided here: The COMPASS tool suite [CML+12] consists at its core of a new modelling language, CML[3]. In Aarhus we are working on making an (Open Source) Eclipse based platform for CML with an extensible front-end. Other tools and technologies will interact through CML: The COMPASS partner Atego[4] provides their tool Artisan Studio[5] for modelling SysML[6] (see [Sys10, Bal07]). An extension for their commercial tool enables Artisan Studio to export SysML models to CML. At University of York work on integrating the theorem prover Isabell/HOL into the project is in progress. UFPE[7] is responsible for creating a model checker for CML . At the University of Bremen a group working with Test Automation that is being integrated into the COMPASS platform. Bremen has a commercial tool for test case generation using constraint solvers (SMT[8]) to generate concrete test cases. An extension to the test case generator is in progress such that test cases can be generated from a CML model. A small website has been setup to provide resource referred to in this report: `http://www.cs.au.dk/~rwl/progrep`.

[3]The COMPASS Modelling Language.

[4]`http://www.atego.com/`

[5]`http://www.atego.com/products/artisan-studio/`

[6]`http://www.omgsysml.org/`

[7]Universidade Federal de Parnambuco – `http://www.ufpe.br/ufpenova/`

[8]Satisfiability Modulo Theories – `http://en.wikipedia.org/wiki/Satisfiability_Modulo_Theories`

2 Background

2.1 Systems Engineering

Systems Engineering is the general term covering the processes and activities carried out to create or improve a system [HFK+10]. A branch of Systems Engineering is Model Based Software Engineering (MBSE) which includes a modelling step to describe the shape and form for a system or part thereof [Est08]. Modelling can be formal in the sense of a mathematical technique for describing software specification, validation, development, and verification. Modelling can also be informal through rigours diagrams and stories describing a system. This work considers a process using Systems Modelling Language (SysML) [Sys10, Bal07] for informal modelling. To put the tools and goal into the broader perspective consider how formal and informal modelling can be utilised in different phases of a Systems Engineering process here described by the Vee-model: The System Engineering process follows the line into the V going through the different

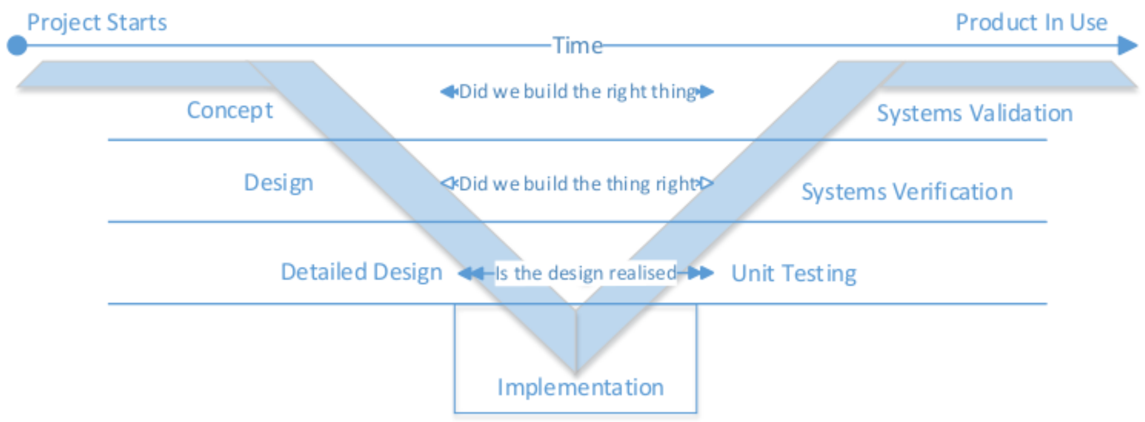


Figure 2.1: Simplified Vee-model.

phases: concept, design, detailed design and so on. In the concept and early design phases informal modelling like SysML is suited for defining system boundaries[1]. Then, formal techniques can be involved as the design crystallises into logic, moving towards an implementable design. This project seeks to contribute to this kind of engineering processes with theory backed by tools to assist in validation of security properties. One benefit from such work is that inconsistencies in security (between the specification and detailed design) can be caught before entering the implementation phase in a SoS setting.

[1]By drawing boxes and figures illustrating the environment of the conceptual idea identifying which elements are going to be part of the system .

2.2 System of Systems

SoS Engineering is an emerging System Engineering discipline even though SoS goes back a long time. SoS was mentioned for the first time in [Bou56]. A characterisation of a SoS by [Ber64] is where SoS is described in terms of the cities on earth. A city can be characterised by the same models as systems can. As such cities are subject to the general systems theory presented by [Bou56]. Consequently, Berry describes "cites as systems within a system of cities" in which different cities are constituents in a larger system. Work on SoS has appeared in the literature but only gained significance up through the 90s with [Mai96] being one of the most cited taxonomies for SoS.

The description of SoS adopted in this work is based on the eight characteristics identified in [NLF$^+$13]. These characteristics do not try to be a conclusive definition for SoS. One purpose is to provide a measure for whether or not SoS modelling techniques are useful for a given Systems Engineering task. For more background and a careful history on SoS see [NLF$^+$13].

Characteristic	**Description**
Anonymity	Each constituent system is free and self-governing, also known as managerial independence [Mai96].
Emergence of Behaviour	The combined system has increased capabilities arising from synergistic collaboration between the constituent systems.
Independence	Each Constituent system is self sufficient and serves a purpose in its own right, e.g. an existing system handling information like a database with business logic on top.
Distribution	The combined systems do not share mass or energy. They are physically detached and cooperate through communication, e.g. two systems communicating over the Internet.
Evolution	The system evolves continually, but slowly, e.g. constituents may acquire new functionality or lose functionality throughout the life cycle of the SoS.
Dynamicity of Behaviour	Constituents systems may come and leave as the SoS continues to function (albeit with reduced capabilities if constituents disappear).
Interoperability	The ability of the SoS to incorporate a wide range of heterogeneous constituents into a collaborative collection.
Interdependence	Constituents rely on each other to achieve the common tasks of the SoS.

2.3 UTP and CSP

Hoare's Unified Theories of Programming (UTP[2]) captures denotational, operational and algebraic semantics in a combined framework for formally specifying designs and implementations of programs [WC04]. UTP is founded on first order predicate calculus and consists of three main

[2]http://www.unifyingtheories.org

components: An alphabet, a signature and healthiness conditions enabling transformations from designs to programs. Hence, for a language like CML that combines two programming paradigms, UTP is a framework in which the meaning of both and the combination can be described . Communicating Sequential Processes (CSP[3]) is a process algebra for modelling concurrency of processes. CSP is an integrated part of Hoare's UTP [RHB97, Hoa85] as one paradigm of programming specialised in concurrency and as such it was created in terms of UTP semantics.

2.4 Vienna Development Method

The Vienna Development Method (VDM) is a formal method technique for specifying, modelling, and evaluating software systems. VDM is one of the oldest formal methods, with its origin dating back to the Vienna Definition Language (VDL) in the early 1970s [FLV08]. Through the addition and combination of multiple techniques, the approach was defined and named as the Vienna Development Method in 1973 [Jon99]. Since then VDM has been applied to a range of industrial projects [FLS08].

The basis of VDM is the ISO standardised VDM-SL notation [PL92, LH$^+$96] which is a language for modelling functional specifications of sequential systems [FL09]. In order to meet new technology and the latest industrial challenges VDM has developed over time by introducing several new language dialects with extended functionality. VDM++ is an object-oriented extension of VDM in which the models consists of collections of classes [FLM$^+$05].
The relevant VDM dialects to this project are:

VDM-SL an ISO standardised sequential language for defining software functionality;

VDM++ which includes the fundamental functionality of VDM-SL but extends it with concurrency and object oriented design.

2.5 The Overture Platform

The Overture project[4] is aimed at the development of an open-source platform for constructing, executing, and analysing VDM models. The project integrates a wide set of tools that include an editor, syntax and type-checker, interpreter, debugger and proof obligation generator.
Figure 2.2 gives an overview of the current state of the Overture Architecture.
Overture consists of two main parts:

- an IDE based on the extensible Eclipse framework, and
- VDMJ [Bat09] an open-source command-line Java tool that contains a parser, syntax- and type-checker, an interpreter, a debugger and a proof obligation generator for all of the VDM dialects.

For further information on Overture and VDMJ please refer to [Bat09, Nie10, CMNL12].

[3]The book on CSP is freely available here http://www.usingcsp.com/cspbook.pdf
[4]Overture project web portal: http://www.overturetool.org

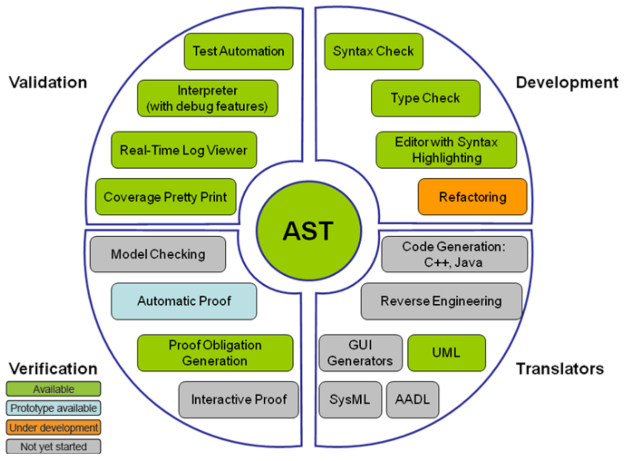


Figure 2.2: Overture Architecture Overview

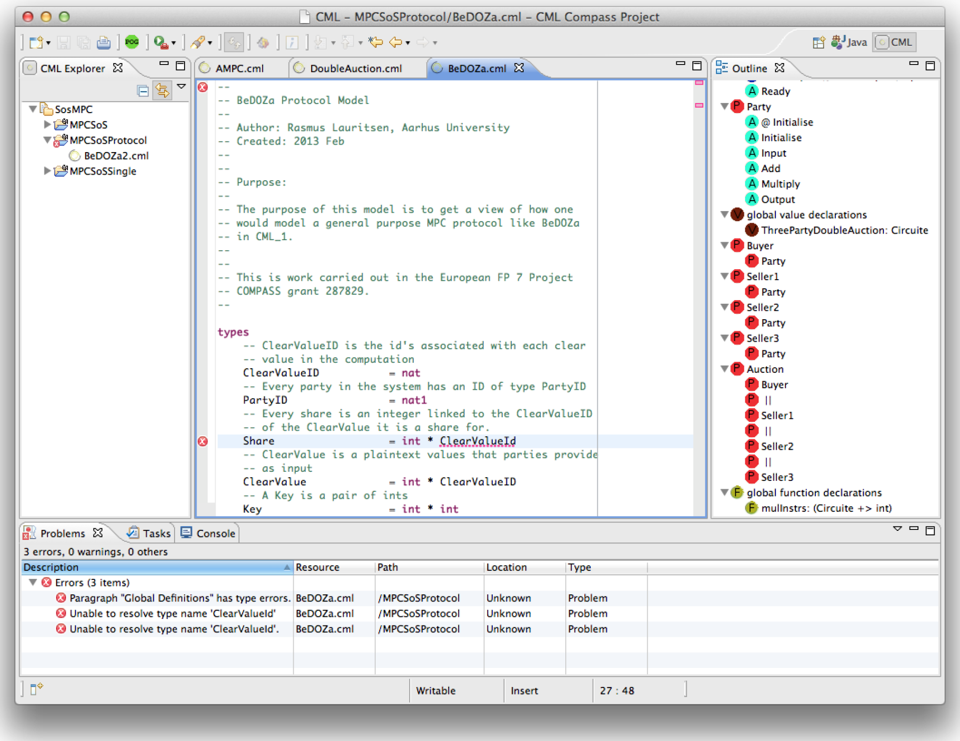
2.6 From Overture and VDM to COMPASS and CML

CML combines VDM++ and CSP , bringing Jones's three valued logic found in VDM++ and the reactive behaviours of CSP together in one language. At the University of York people are working on expressing VDM++ in UTP. Having both the UTP semantics for CSP and VDM++ forms a formal semantics for CML. The CML language is under continuous development throughout the project. The syntax and semantics so far are defined in [Col13] and [WBB+12] respectively.

The effort of making tool support for the CML formalism is spread on several PhD projects. Our ongoing work is an extension of the Overture platform. That is, tool components like the type checker for CML are extending the equivalent counterparts from Overture. This is made possible by parsing CML source into an extended version of the AST in Figure Figure 2.2 enabling extensive reuse from Overture for analysis including proof obligation generation and interpretation/animation. The CML compiler front-end parser and type checker are well on the way. As part of this work I have contributed with the development of the CML type checker. Figure 2.3 illustrates the type checker in action: A type error is found in the definition of the "Share" type. The error is caused by "ClearValueID" being misspelled with a small d. This project has been working on unpublished work describing the typing rules as implemented in the tool. The tool is an open source project located at `http://sourceforge.net/projects/compassresearch/`.

2.7 Multi-Party Computation

In the seemingly nonrelated world of cryptography lies the notion of Multi-Party Computation (MPC). Briefly explained, MPC enables a set of parties $P_1, ..., P_n$ to compute a function $f(x_1, ..., x_n) = y$. Special to MPC is that each party holds the secret input x_i and the function is computed such that first of all the output is correct and secondly the output is the only new information that the parties learn. Despite the different backgrounds MPC and SoS can have a very common structure. In particular in the presence of privacy requirements; consider a typical SoS problem where a number of organisations $O_1, ..., O_n$ want to cooperate in order to achieve some common task. If for some O_i the common task requires sensitive information to the organisation an MPC solution may be the answer.



```
--
-- BeDOZa Protocol Model
--
-- Author: Rasmus Lauritsen, Aarhus University
-- Created: 2013 Feb
--
--
-- Purpose:
--
-- The purpose of this model is to get a view of how one
-- would model a general purpose MPC protocol like BeDOZa
-- in CML_1.
--
--
-- This is work carried out in the European FP 7 Project
-- COMPASS grant 287829.
--

types
    -- ClearValueID is the id's associated with each clear
    -- value in the computation
    ClearValueID            = nat
    -- Every party in the system has an ID of type PartyID
    PartyID                 = nat1
    -- Every share is an integer linked to the ClearValueID
    -- of the ClearValue it is a share for.
    Share                   = int * ClearValueId
    -- ClearValue is a plaintext values that parties provide
    -- as input
    ClearValue              = int * ClearValueID
    -- A Key is a pair of ints
    Key                     = int * int
```

3 errors, 0 warnings, 0 others

Description	Resource	Path	Location	Type
Errors (3 items)				
Paragraph "Global Definitions" has type errors.	BeDOZa.cml	/MPCSoSProtocol	Unknown	Problem
Unable to resolve type name 'ClearValueId'	BeDOZa.cml	/MPCSoSProtocol	Unknown	Problem
Unable to resolve type name 'ClearValueId'.	BeDOZa.cml	/MPCSoSProtocol	Unknown	Problem

Figure 2.3: Screen shot of the Eclipse IDE running with the COMPASS compiler front-end

The area of Multi-Party Computation (MPC) starts in 1982 with Andrew Yao's initial paper [Yao82]. Yao puts forward the Millionaires Problem: Two millionaires wish to know who is richer; however, they do not want to find out any additional information about the others wealth. How can they carry out such a conversation? Yao presents solutions to this problem and generalises it into the general problem of MPC briefly introduced above. Another interesting problem is that of a double auction of some commodity. In one flavor of a double auction a set of prices is publicly known. Each buyer submits a bid in the form of a quantity he wishes to buy at every price. Similarly each seller submits the quantity she wishes to sell at each price. The result of the auction is computed as the price at which all buyers and all sellers are willing to buy and sell an equal amount the commodity (or as close as possible to that given their bids). This price is known as the market clearing price. The agreement is such that sellers get to sell the quantity they bid at the clearing prices. Likewise, buyers get to buy the quantity they bid at the clearing price. The MPC challenge for the double auction is for the sellers and buyers to find the market clearing price without revealing their bids to anyone.

In this section the concepts and technologies used in this project has been presented. Initially Systems Engineering was presented and this work (and COMPASS) was placed in that context. Zooming in, this section briefly introduced UTP, CSP and VDM, that were the theories upon which CML is founded. Additionally, a short account of the tools being used and developed was included. Finally, a brief introduction to MPC with highlights of its similarity with SoS was given.

3 Modelling Digital Rights Management

The work presented in this chapter is based on [LL12]. In [LL12] exploration with B&O was carried out to find which elements should be in a reasonable model for a system required to handle Digital Rights Management (DRM). DRM is a common term for a variety of methods to limit or control the usage of digital premium content[1] and the distribution mechanisms involved.

The work builds on previous work describing DRM in [WK04] and [PCC$^+$04]. Section 3.1 below puts this work in context of the COMPASS project. The body of work is described in sections 3.2 and 3.3, presenting the modelling carried out.

3.1 Setting the Research Context

As part of the COMPASS project, B&O is looking for ways of tackling some of the challenges in a quickly moving Consumer Electronics market. In [LL12] it is argued that B&O's situation is a collaborative SoS. One interesting aspect of their SoS with respect to security is how DRM is handled. Two very important properties for B&O is compliance with current DRM standards and a smooth experience for the user. As an example consider the challenge of handing an update for DRM requirements such that B&O equipments remain able to play future media. It is the idea that having a rigorous model for DRM will assist in ensuring continuous operation of B&O products while also supporting the task of documenting DRM compliance. To achieve the task above, this work tries to come up with a generic model for DRM technology.

Motivated by B&O's involvement in the COMPASS project, work on modelling DRM has been carried out. The main goal is to create a general reusable core model for DRM upon which trade-off studies of different DRM solutions can be made. For most users the understanding of DRM is related to restrictions on the content. For example these restrictions could be on how many playbacks are allowed, who can play it and where. Other common restrictions are copy-protection and similar redistribution limiting mechanisms [Soh07]. Strict requirements are also imposed on the entire chain of DRM entities, from copyright holder to consumer. The DRM system specifies who authorises release of content, how it should be authorised, how to store key-material for accessing the content etc. [PCC$^+$04, WK04]. Hence for the kinds of Home Entertainment Systems that B&O manufactures, the engineering task of coping with DRM carries many of the characteristics for SoS. The goal of tackling the challenge of maintaining DRM in a product line for Home Entertainment Systems suggests an initial core model for capturing central DRM concepts. Then, for proof of concept a model leveraging the core model for modelling cryptographic parameters is presented.

[1]Premium content and services are those that the user explicitly pays for.

3.2 The Core Model for DRM

This section presents work carried out for creating a generic VDM-model representing the view on DRM systems laid out in [WK04]. The VDM model is generic in the sense that it should comprise the elements necessary (albeit they may need to be specialised) to model any concrete DRM system as described by [WK04]. The following presents each of the model elements included in this generic VDM model for describing a given DRM system. Then an example of how to utilize the generic model elements to model a concrete (but simplified) DRM setup is presented. This latter model enables the Overture Tool to check for cryptographic parameter consistency though simulation of the model.

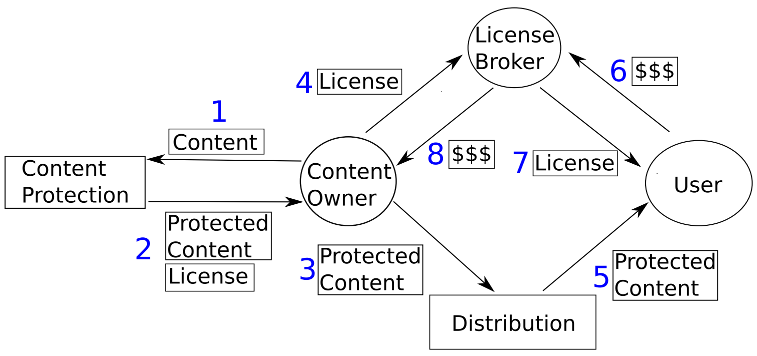


Figure 3.1: DRM Model from Ku et al.

In Ku et al. [WK04] DRM is viewed as shown in Figure 3.1. The interplay between these elements is briefly summarised here:

1 The content owner inputs some new content to the content protection mechanism with a description[2] of how it may be used.

2 The content protection mechanism of the DRM System responds with a protected version of the content and a license.

3 The content owner distributes the protected content through the distribution mechanism.

4 The content owner sends the licenses to the license broker which sells them to viewers.

5 The end user acquires material from the distribution mechanism and examines its meta-data to figure out which license to acquire in order to play the content.

6 If the viewer does not already have a license for the acquired content, one can be bought from the license broker.

7 When a viewer buys a license from a license broker two things happen, a license is transferred to the viewer and money is transferred to the license broker.

8 All or some of the money transferred to a license broker is subsequently transferred to the appropriate content owner.

[2]Such description are the rights granted on the content by the resulting license and are typically expressed in a Rights Expression Language (REL). In this model we have a quite simple version of REL which may be subject to future work.

Ku et al. focus on six important areas that a DRM model typically covers. In the following focus is on one of them, namely the concept in DRM of "Secure Containers" which covers means of restricting access to the content, for instance by encrypting it. The other areas are briefly treated as well in [LL12].

The DRM Model

The approach for creating this model follows techniques in Larsen et al. [LFW09]. Thus, in addition to the DRM entity classes a *World* class for entry points and an *Environment* class modeling interaction with entities outside the core-DRM system are added. For a comprehensive treatment of VDM++ see Fitzgerald et al. [FLM$^+$05]. Figure 3.1 on the previous page describes the principal elements in the Ku et al. view on DRM. From this figure the following VDM++ classes immediately spring out: *ContentOwner*, *ContentProtection*, *Content*, *ProtectedContent*, *License*, *LicenseBroker*, *Viewer* which is depicted as user on the figure, *Distribution*, and *Money*. To determine which properties and operations to include for each of the entities above, the following sections derive their requirements based on their operations described in Section 2.2 of Ku et al. [WK04].

The **content owner** initially has some "raw content". The DRM system might require this raw content to be formatted. Optionally, the content owner may add a watermark to the content. As part of entering their content to the DRM system a set of rights must be specified for the content. The DRM system will produce a set of licenses and a protected version of their content. These will be disseminated to relevant license brokers and a distribution channel respectively. Upon a purchase of a license between a license broker and a viewer, the content owner may receive payment. These operations give rise to the following operations and instance variables on our VDM *ContentOwner* class.

The **content protection** component in the DRM system encrypts and packages content for distribution. This component also creates licenses for accessing protected content. The fact that viewers need to access the protected content creates a link between the *ContentProtection* and *Viewer* elements. This link is not made visible in Ku et al.. As a first attempt to capture this link the model implements the idea of a Protection Domain in the VDM++ class *ProtectionDomain*. *Viewer*'s and *ContentProtection* classes are both instantiated with a *ProtectionDomain* instance. Enforcement of access limitations are implemented in the *Viewer* class that makes sure only to access un-protected content or *ProtectedContent* instances created by a *ContentProtection* instance having the same domain as it self.

The **content** represents raw content before protection. Content has two properties namely its Format and Watermark. Naturally, our VDM++ class *Content* carries these two properties. Otherwise ontent is abstract, only carrying an additional id for comparison.

The **protected content** is content which has entered the DRM system and has been protected by some content protection entity. Protected content is modeled using aggregation rather than inheritance here to reflect the fact that the content is "inside" the protected content. Our *ProtectedContent* classes therefore carries two instances variables, one for its content and one for the protection domain that protects this content.

The **License** grants access to perform a set of actions with a piece of content on a Viewer in some Protection Domain. Therefore, the *License* class naturally carries a pointer to an instance of the *ProtectedContent* class and an instance of *RELInstance*. The license is basically an immutable data carrier and its values could be public. However to, facilitate open behaviour for extension models it is a design choice to have accessors methods instead.

The **License Broker** receives *Licenses* from *ContentOwner*s and set these available for sale. This enables the *LicenseBroker* to serve a *Viewer* with a *License* for a given piece of *ProtectedCon-*

tent. Transferring a license to a *Viewer* given an instance of *Money* and an identifier for content is the operation *SellLicense*. SellLicense may fail if no license can be found and in such cases it returns the value <*None*>. One interesting detail unclear in the Ku et al. article is how and when money is transferred from the *LicenseBroker* to *ContentOwner*'s. In this model it is captured with the method *GetPayment* to be called by a *ContentOwner* in case money is relevant.

A **Viewer** is capable of playing back content protected by a content protection instance in a Protection Domain the Viewer can access. Hence, *Viewer*'s carry a list of Protection Domains from which they can access *Content*. In order to actually access *Content* (or try to), a *Viewer* has a Play operation which takes a piece of content as argument and returns a Boolean stipulating whether access was successful or not. The Viewer also carries a set of licenses that have been acquired through invocation to its *BuyContent* operation. To support playback of *Content* a viewer has a *PlayContent* operation.

Distribution allows *ContentOwners* to publish *Content* and allows *Viewers* to browse all published content. Hence, the Distribution has a *PublishContent* operation and a *BrowseContent* operation.

The complete model can be downloaded from `http://www.cs.au.dk/~rwl/progrep/kumodel.zip` and executed using the Overture tool suite (see Section 2.5).

3.3 Modelling Security

In this section the core model above is tested on a simplified DRM system where symmetric encryption is utilised. The key distribution is assumed to be handled by actors outside the model. The goal for this section is to describe the model created that allows a system designer to model different kinds of encryption for DRM protection while still accessing content in the Viewer. The kind of encryption used is stipulated by parameters such as: Cipher Algorithm, Mode of Operation, Padding and the Key used (inspired by [Dam11]) see Listing 3.1.

The aim is to extend the generic VDM++ model only when needed, using the inheritance feature of the language. Otherwise the generic components can be used. This is illustrated in Listing 3.2. Instances of each kind of element needed to DRM like Ku et al. are created. For distribution, content, content owner and license broker the generic elements could be used without alteration. With the generic model describing how components are interacting it became clear from creating the example, that the protected content, the content protection, and the viewer needed to be changed. The *ProtectedContent* needed to be extended with parameters telling which encryption parameters with which it was protected. The *ContentProtection* needed to know which kind of parameters it is using and finally the *Viewer* needed to be setup with the set of encryption-mechanisms it can handle. These considerations gave rise to three new elements specialising existing ones: *CryptoContentProtection*, *CryptoProtectedContent* and *CryptoViewer* enhanced with the information just discussed.

```
class CryptoParameters
types

  public EncAlg = <AES> | <TDES> | <DES> | <...>
  public ModOpr = <CBC> | <ECB> | <RAW>;
  public BlkSiz = <bs8> | <bs16> ;
  public PadAlg = <ZeroPad> | <Pkcs7> | <...>

  public SymmetricMechanism ::
   cipher : EncAlg
   mode   : ModOpr
   pad    : PadAlg;

  public ProtectionDomainType = <Top> | ProtectionDomain;

  public ProtectionDomain ::
    parent : ProtectionDomainType
    cryptoCapability: CryptoCapability
    domain: seq of char;

functions
  public checkAccess: Viewer * ProtectionDomain *
                            SymmetricMechanism * Content -> bool
  checkAccess( viewer, key, mech, content ) == <...>
```

Listing 3.1: Modelling of cryptographic-parameters.

In Listing 3.1 the function checkAccess encapsulates the logic of computing whether or not access for a given viewer can be granted. Below is a listing for the scenario which also illustrates how a DRM system is set up, here with the Crypto extensions elements created for this example.

```
class CryptoExample
values
   public www : Distribution = new Distribution();
   public aes_cbc_zeropad_content_protection :
       ContentProtection =
      new CryptoContentProtection(mk_SymmetricMechanism(<AES>,<
          CBC>,<bs16>,<Pkcs7>));
   public phil_collins_against_all_odds: Content =
     new Content(mk_token("Against all odds"));
   public virgin : ContentOwner =
      new ContentOwner( mk_token("virgin music inc."),
                        {phil_collins_against_all_odds});
   public virgin_enabled_viewer =
      new CryptoViewer(mk_SymmetricMechanism(<AES>,<CBC>,<bs16
          >,<Pkcs7>));
   public amazon_shop : LicenseBroker = new LicenseBroker();
```

Listing 3.2: A Correct cryptographic parameters setup.

In Listing 3.2 is a scenario describing a situation where the cryptographic parameters are correct. The scenario in Listing 3.4 will successfully simulate play back of a piece of music.

```
class CryptoExample
values
 public www : Distribution = new Distribution();
 public aes_cbc_zeropad_content_protection : ContentProtection =
  new CryptoContentProtection(mk_SymmetricMechanism(<AES>,<CBC>,<bs16>,<Pkcs7>));
 public phil_collins_against_all_odds: Content =
  new Content(mk_token("Against all odds"));
 public virgin : ContentOwner =
  new ContentOwner( mk_token("virgin music inc."),
   {phil_collins_against_all_odds});
 public virgin_enabled_viewer =
   new CryptoViewer(mk_SymmetricMechanism(<DES>,<CBC>,<bs8>,<Pkcs7>));
 public amazon_shop : LicenseBroker = new LicenseBroker();
```

Listing 3.3: An incorrect cryptographic parameters setup.

In Listing 3.3 the cryptographic parameters are incorrect because the *Viewer* is using `<DES>` encryption and hence this setup makes the scenario fail to play back the content.

```
 public static PublishAndPlayScenario: () ==> ()
   PublishAndPlayScenario() ==
   (dcl rights: RELInstance := virgin.SpecifyRights({<Play>});
    dcl a : Content := virgin.FormatContent(
        phil_collins_against_all_odds);
    dcl b : Content := virgin.AddWatermark(
        phil_collins_against_all_odds);
    dcl c : ProtectedContent * License
        := virgin.EnterContent( phil_collins_against_all_odds,
                 rights,
                 des_cbc_zeropad_content_protection);
    virgin.DisseminateContent( www );
    virgin.SendLicenses({amazon_shop});
    virgin_enabled_viewer.BuyLicense ( amazon_shop,
                   phil_collins_against_all_odds.id); --Acquire
                        license
   if (virgin_enabled_viewer.PlayContent(
       phil_collins_against_all_odds.id,
                                     www ))
    -- Acquire content and play it
   then
       IO`println("Crypto is ok, we are able to play.")
   else
       IO`println("There is a mistake in the model, unable to
          play")
   );
end CryptoExample
```

Listing 3.4: Code listing for an example checking for consistence of cryptographic-parameters.

The significant aspect of this model is that it leverages the core model elements described in the previous section. The model above extends the core model components to capture new properties of DRM, namely cryptographic parameter consistency. Hence, the core model for DRM as described in Ku et al. seems like a reasonable starting point when modelling specific properties with a given DRM setup. In comparison with the model from Popescu et al. the Ku et al. model aims at being a complete overview for DRM , whereas the Popescu et al. model details the User situation. For the purpose of finding a common set of elements that could define a terminology for modelling DRM-like systems, elements from both models are needed; It turns out that for B&O as a manufacturer of consumer electronics the model Ku et al. model needs to be *refined* for the user's situation, shereas the Popescu et al. model captures the whole range of entities (e.g. content owners are not captured). An immediate continuation of this work would be to extend the model with the granularity at the user's site found in [PCC$^+$04].

4 Modelling Multi-Party Computation

This section presents the work in [Lau13] made in preparation for submission to the 11th International Conference on Software Engineering and Formal Methods (SEFM2013[1]). The systems engineering approach for MBSE using VDM++ in [FLM$^+$05] is applied to the first industrial scale application of MPC found in [BCD$^+$08].

The section starts out by motivating the choice of protocol modelled in this work. Then follows a short description of the protocol in Section 4.1. Section 4.2 describes the motivating industrial application. The body of work is described in sections 4.3, 4.4 and 4.5 presenting three models, gradually and informally refined towards a secure solution to the industrial application.

To base this work on state of the art and because of new developments in MPC since 2008 the BeDOZa protocol named after its creators in [BDOZ11] is modelled. A feature with BeDOZa is that its on-line phase is secure in the presence of so-called active adversaries. Security against active adversaries captures the case where the protocol tolerates that the corrupted parties deviate from the protocol. This scenario is believed to be more realistic than assuming all parties always follow the protocol to the letter.

The goal is to show feasibility of the SoS MBSE formalism CML to describe MPC. The approach here starts with the modelling approach for VDM++ , resulting in a comprehensible and easily verifiable single system model in Section 4.3. Inspired by the Universally Composable framework [Can00] the model for the full protocol is derived through a model for an idealised situation. The purpose with this is two-fold. First, it is easier to comprehend and verify correctness on the idealised model. Secondly, it ignites the idea of using modelling to assist proving protocols secure: Can analysis on an idealised model against a protocol model show results related to indistinguishability between the real and ideal scenarios in the UC-framework? Note, that the refinement steps resulting from this approach are relatively crude and do not entirely follow the "normal" formal methods sense of the word. However, this is intended as an attempt to use the principle idea of comparing an idealised scenario to an implementable one from the UC-framework in the world of modelling.

4.1 The BeDOZa Protocol

In MPC the description of what a protocol achieves is described by a so-called ideal functionality. This functionality is ideal in the sense that it takes inputs securely from the parties and always computes the function correctly. The goal in proving a protocol secure is to show that the real world protocol has the same characteristics as the ideal functionality. The BeDOZa protocol realises the *Arithmetic Multi-Party Computation* ($AMPC$) ideal functionality. $AMPC$ can evaluate arithmetic circuits described by the four operations: $Input$, Add, Mul and $Output$.

[1] http://antares.sip.ucm.es/sefm2013/

MPC typically relies on heavy cryptographic machinery with computationally expensive primitives. One trick to make MPC fast when it is needed is by using pre-processing to handle the more heavy computations. Pre-processing means that the protocol is divided in two phases: An off-line phase and an on-line phase. BeDOZa uses pre-processing where the goal is to precompute some values in the off-line phase that enables efficient comptations in the on-line phase. These values are independent of the function to be computed and the inputs to the function, and hence the pre-processing can be run as preparation anytime. Then, in the on-line phase an actual computation on sensitive information is going on, where the parties agree on the function (expressed by an arithmetic circuit in terms $Input$, Add, Mul and $Output$) and each party provides his secret input to the protocol computing the function. In this work only the on-line phase is modelled. The on-line phase is described in [BDOZ11] pages 11 – 15. For completeness it is summarised here.

Additive Secret-Sharing. The goal of the protocol is to compute a circuit with values in $\mathbb{Z}_p$. A key technique is to *additively secret-share* the values between the parties. Assume there are N parties $P_1, ..., P_N$, then a value $a \in \mathbb{Z}_p$ is additvely shared as

$$x = x_1 + \cdots + x_N,$$

such that each P_i holds a random x_i. Note that a single party has no idea about what x is since he only knows a single share and even if all but one of the players go together this does not reveal information about x. $[x]$ will in the following denote this setup.

Linear Compuations. Now linear computations on secret-shared values can be done simply by local computations on the shares. Hence, it makes sense to write $[x] + [y] = [x + y]$, where each party computes $x_i + y_i$, and similarly follows multiplication by a constant δ: $\delta \cdot [x] = [\delta \cdot x]$. A public constant δ can be added by letting a single party, say P_1 add that constant to his share, so $\delta + [x] = [\delta + x]$, where P_1 computes $\delta + x_1$ and other parties do nothing.

Input Sharing. To begin the computation each party P_i wants to provide his input x to the circuit. Here the pre-processing comes to play. Assume it provides a secret-shared random value $[a]$. Now this a is privately revealed to the party providing his input. This is done by letting every other party P_i send his share a_i to P_1 and then P_1 can compute $a = a_1 + \cdots + a_N$ (this is a private opening of a to P_1). Finally P_1 publishes $\delta = a\text{-}x$ as a public constant and parties compute $[x] = \delta + [a]$. Note that the random a masks x so that δ reveals no information on x.

Multiplication What is left is a way to compute $[x \cdot y]$, for secret-shared values $[x], [y]$. For this purpose the pre-processing provides 3 secret shared values $[a], [b], [c]$ where a, b are random values and $c = a \cdot b$. Similarly as in the case of input, two constants are computed and publicly opened (all parties broadcast their shares): compute and open $[x]$-$[a]$ call it ϵ, and compute and open $[y]$-$[b]$, call it δ. By the computations described above it can easily be seen that $[x * y] = [c] + \epsilon \cdot [b] + \delta \cdot [a] + \epsilon \cdot \delta$.

Message Authentication Codes. If it is assumed that the parties follow the protocol, the above would be enough. However, the protocol should be secure against corrupted parties that might want to deviate. Hence, some kind of authentication is needed. This is done by so-called information theoretic Message Authentication Codes (MACs). A MAC on a value x is defines as $m_K(x) := \alpha \cdot x + \beta$, for a key $K = (\alpha, \beta)$, where α, β are randomly chosen, and α is "global". This means, that to authenticate another value y the MAC will be $m_{K'}(y) = \alpha \cdot y + \beta'$, for key $K' = (\alpha, \beta')$. The setup will be such that one party holds a value, and a MAC, and then the key is held by another party. Authentication now follows since for a party to be able to cheat with a value, he essentially has to guess α. This he cannot do better than by just guessing at random since β hides α in m.

It is possible to compute with this kind of authentication. Addition of keys is defined as the addition of the second component: $K = (\alpha, \beta), K' = (\alpha, \beta'), K + K' := (\alpha, \beta + \beta')$. Now it is easy to verify that $m_K(x) + m_{K'}(y) = m_{K+K'}(x + y)$. For addition with a constant and multiplication something similar can be done. Then, the idea is to augment the $[\cdot]$-notation with these MACs. P_i has a MAC $m_{K^j_{a_i}}(a_i)$ on a_i towards every other party P_j. Also, for P_i to be able to check the MAC on a_j from P_j, P_i has a key $K^i_{a_j}$ for every P_j. In summary each share will have N-1 MACs and each party will hold N-1 keys to verify the MACs of the other parties' shares. Furthermore, all the keys a party P_i holds towards P_j will have the same first component in order to enable computations as described above.

The protocol now assumes a sufficient number of single random values $[a]$ (Single) and triples $[a], [b], [c]$ (Triple) given from the pre-processing where MACs and keys are contained in $[\cdot]$. Then computing is done as described above, computing with the MACs along with the shares. In the final step when the output is opened (all the shares of the result are revealed), each party will need to convince every other party about the correctness of his share.

4.2 Case Study: Double Auction

This section summarises the first industrial application of MPC as described in [BCD$^+$08]. In Denmark thousands of farmers produce sugar beets which are sold to one company, Danisco. There is a pricing scheme and individual contracts are negotiated between the farmers and the company. The farmers are allowed to trade these contracts with each other. A contract describes an amount of sugar beets the farmer is allowed and obligated to sell to Danisco at the price given by the publicly available pricing scheme. As EU were reducing subsidisation for sugar beets the government decided to regulate the market. The goal was to find a fair market price for a commodity given the existing supply and demand in the market. The best way to do so was found be a so called double auction.

For an overview on auctions see [Kle99] in particular Section 12 for double auctions. The double auction in [BCD$^+$08] is carried out is the following way:

- A fixed set of prices is made publicly available to everyone before the auction starts.
- Each sellers creates an ordered list of numbers with one number for each price. This number is the quantity of beets he is willing to sell at the given price.
- Each buyer creates an ordered list describing the quantity of beets he is willing to buy at the given price.
- With the list of numbers from sellers and buyers above it is possible to compute market supply (summing the sellers quantities) at each price and market demand (summing the buyers quantities) at each price.
- The price at which demand and supply is minimal is known as the market clearing price, and the result of the double auction.
- Each seller having a non-zero quantity at the *market clearing price* is allowed to sell their quantity at that price.

- Likewise each buyer having a non zero quantity at the market clearing price is allowed to buy that quantity at that price [2].

Mathematically put we wish to compute the following function:
Let $N \in \mathbb{N}$ be the number of prices, $P_1, ..., P_N$ be the possible prices, $S > 0$ be the number of Sellers and $B > 0$ be the number of buyers.

$$
\begin{aligned}
d_j &= \sum_{i=0}^{B} \mathbf{buyer}_i[j] \text{ Demand at price } j. \\
s_j &= \sum_{i=0}^{S} \mathbf{seller}_i[j] \text{ Supply at price } j. \qquad (4.1)\\
&\min_{j \in \{1,...,N\}} (\mid d_j\text{-}s_j \mid)
\end{aligned}
$$

Here $\mathbf{buyer}_i$ is the bid (e.g. wanted quantities at each price) given by the ith-buyer. Likewise $\mathbf{seller}_i$ is the bid (e.g. quantities supplied to the market at each price) given by the ith-seller.

According to [BCD$^+$08] an MPC solution was chosen because no such system existed beforehand and a "cheap" computerised solution seemed attractive. Additionally, MPC short-circuits discussions and concerns about which parts of the data are sensitive and what common security policy one should have for handling the auction data.

Eight characteristics for SoS are listed in [NLF$^+$13]. The double auction as in the setting of [BCD$^+$08] has interdependence, distribution and anonymity. However evolution, independence, dynamicity of behaviour and interoperability is not necessarily achieved in that the system is constructed for a very specific purpose. However, it is reasonable to argue that the system is likely to acquire some of the missing characteristics. Still, this scenario has the most interesting SoS characteristic for modelling security in SoS, namely anonymity. Recall anonymity entails that each constituent system is a self-governing institution possibly having competing goals with other constituents in the SoS. This tension is clearly present for a the double auction considered here. To see why e.g. independence might be likely in the sugar beet scenario consider the task of automating the process of inputting the bids. These data (for the bids) must be coming from computations in some kind of accounting/bookeping system. This accounting system has a self-sufficient purpose (accounting) and thus independence is achieved.

Recent developments in MPC provide general purpose protocols that are independent of the function being computed, where efficiency has improved since 2008. Hence, these protocols are more interesting to model in this work. The BeDOZa protocol computes arithmetic circuits and therefore does not have a primitive to compute the $min(x, y)$ operation. This primitive is necessary to compute the result of a double auction. Luckily it has been shown that min can be computed efficiently based only on black-box access to AMPC[Tof07]. This will be included in the full paper but in the following the $min(x, y)$ operation assumed to be present.

Next, three CML models are presented. The first one is a single system model describing the ideal situation where a single computer takes as input all the quantities and the lists of prices, and computes the market clearing price. Then, a refinement of that is presented where an ideal functionality, AMPC, is communicating with the parties in the auction. Finally, yet a refinement into the full protocol description is presented where the double auction is realised without any by-all-trusted components.

[2] In [Kle99] Buyers input a bid whereas sellers input an "ask"s. Bids and Asks concretised in [BCD$^+$08] in terms of Quantity per Price.

4.3 Single System Model

CML consists of VDM++ and CSP elements where CSP provides interactive behavioural elements whereas VDM++ provides the structural ones. Creating the single system of a double auction we consider the method tailored for VDM++ described in [FL98] pg. 20. The first step is to *determine the purpose of the mode*.

Model Purpose The *purpose* with this model is to describe a double auction at a level of abstraction that is easy to comprehend and manually verify that it is functionally correct. A reasonable set of requirements for security are made but treated outside the model for the single system model.

The second step is to *read the requirements*. From [BCD$^+$08] the central requirements for the double auction are elicited and the result is given in the list below:

R0 A double auction should consists of a set of participants.

R1 It should be possible to make a set of prices known to all participants in the auction.

R2 Each participant should be either a seller or a buyer.

R3 Buyers should be able to input a negative quantity for each price (e.g. the number of sugar beets that buyer is willing to buy off the market at that price).

R4 Sellers should be able to input a positive quantity for each price (e.g. the number of sugar beets that seller is willing supply to the market at that price).

R5 When all sellers and buyers have provided input it should be possible to compute the market clearing price according to the definition in Equation 4.2.

R6 No seller should learn another seller's bid (except from what follows from the result of the auction).

R7 No seller should learn another buyer's bid (except from what follows from the result of the auction).

R8 There is only one buyer, Danisco. The buyer should not learn any sellers bid (except from what follows from the result of the auction).

The third step is to *analyse functional behaviour for the requirements*. Requirement **R0** indicates that there should be some way of introducing a set of participants corresponding to real world entities (persons or organisations) to the system. With **R2** the set could be partitioned in sellers and buyers. **R1** hints at a functionality that facilitates a set of prices to be made publicly known to all participants. Requirement **R3** states that buyers should be offered the option of inputting a set of none-positive numbers. Implicitly it requires this list having the same number of elements and the cardinality of prices made publicly known by **R1**. Similarly, the sellers are offered the option of inputting a set of non-negative numbers by requirement **R4**. Requirement **R5** implies that when all participants have provided their input, the system computes the market clearing price. Notice that the *privacy* related requirements **R6** – **R8** are not addressed yet for reasons that will become clear shortly.

One straightforward way to represent the conceptual functionality in Figure 4.1 is by representing the box in the double auction as a CML-class and having the edges going in and out of the box represented by operation calls. Sellers and buyers are represented by their bids and only the accumulated quantity for sellers and buyers are needed. The result is the model in Listing 4.1.

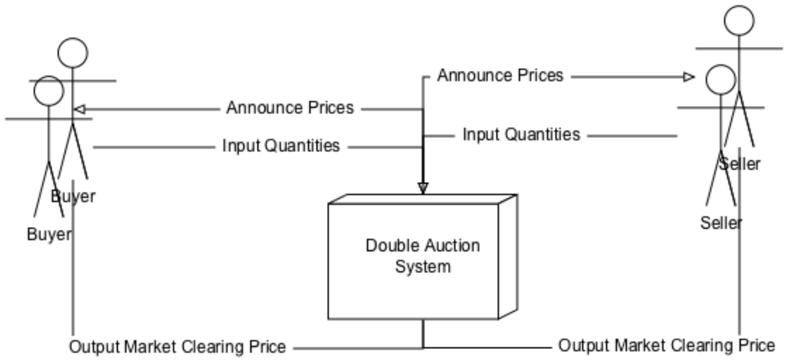


Figure 4.1: Double Auction Overview

The Model The model (Listing 4.1) uses *implicit operations* which are merely pre-conditions and post-conditions leaving out implementation details. Security is abstracted in an idealised way: It is assumed that the parties can only interact with (and cannot look inside) a system realising the described operations though the public operations and states. No party can access the private members of such a system. Also, input to the constructor is publicly known and agreed upon before instantiating the system. It is assumed that input given to public operations are handed to the system in a private and secure way. In effect, at the level of abstraction used here the *operation and state qualifiers* can be used to model the privacy concerns addressing requirements **R6** - **R8**.

```
class DoubleAuction =
begin
types
  public Quantity = int
  public Price    = rat inv p == p >= 0

state
   public numberOfBuyers : nat;
   public numberOfSellers: nat;
   -- seller bids contains the accumulated bids for sellers
   private sellerBids    : seq of Quantity
   -- buyer bids contains the accumulated bids for buyers
   private buyerBids     : seq of Quantity
   public  prices        : seq of Price
   inv
      len(sellerBids) = len(buyerBids) and
      len(sellerBids) = len(prices)

operations
-- Constructor initialises sellerBids, buyerBids and the list of
    prices
-- for the auction.
  DoubleAuction: seq of Price, nat1, nat1 ==> ()
  DoubleAuction(ps, noBuyers, noSellers) == (
    prices := ps;
   for all i in set inds ps do sellerBids := sellerBids ^ [0];
                for all i in set inds ps do buyerBids := buyerBids ^
                    [0]
                numberOfBuyers := noBuyers; numberOfSellers :=
                    noSellers;
                )
```

```
        public AnnouncePrices () r:Price
        pre true
        post r = prices

        public AddSeller(bid: seq of Quantity)
        pre len(bid) = len(prices) and forall b in set elems bid @ b
            >= 0
            and numberOfSellers > 0
        post (forall i in set inds prices @ sellerBids(i) = sellerBids
            ~(i) + bid(i))
            and numberOfSellers = numberOfSellers~ - 1

        public AddBuyer(bid: seq of Quantity)
        pre len(bid) = len(prices) and forall b in set elems bid @ b
            <= 0
            and numberOfBuyers > 0
        post (forall i in set inds prices @ buyerBids(i) = buyerBids~(
            i) + bid(i))
             and numberOfBuyers = numberOfBuyers~ - 1

        public AnnounceMarketClearingPrice () r:Price
        pre numberOfBuyers = 0 and numberOfSellers = 0
        post (exists j in set inds prices @
                 (forall i in set inds prices @
                    sellerBids(j) + buyerBids(j) <= sellerBids(i) +
                        buyerBids(i))
                and r = prices(j))

end
```

Listing 4.1: Single System Double Auction

Correctness in requirement **R5** is handled by the `AnnounceMarketClearingPrice` operation, and the two state-variables `numberOfBuyers` and `numberOfSellers` which are set initially by the constructor. Requirements **R0** and **R2** are implicitly represented in that multiple sellers and buyers can be added. In fact, this model and description of the requirements are slightly more general than necessary for the sugar beet auction because it allows multiple buyers. Requirements **R1**, **R3** and **R4** are met by defining the functions `AnnoucePrices`, `AddSeller` and `AddBuyer`.

4.4 Ideal Distributed Model

The single system model presented in the previous section focuses on describing correct functionality. In this section the single system model is *refined* into a reactive model where users interact through one device each. In turn, their devices communicate with an idealised functionality over perfectly secure channels.

Model Purpose In the single system model requirements **R6** – **R8** are achieved by assumptions on the environment external to the system. The purpose with that model was to demonstrate a functional overview. The *purpose* with this model is to demonstrate the behavioural aspects of the protocol, i.e. focus is on highlighting communication aspects.

Level of Abstraction In this section security is shown by assuming an incorruptible ideal functionality $AMPC$ as in the ideal execution of [Can00]. In this setting the party process instances communicate securely with the functionality which behaves ideally (e.g. it is the specification of how the protocol should behave).

The Model This model is created to model the situation in an ideal execution of the BeDOZa protocol in [BDOZ11]. In Listing 4.3 the ideal functionality is modelled as a process. This process is instantiated with the circuit to be computed. The circuit is described by a sequence of records `Inp, Add, Mul, Min, Out`. In Listing 4.2 this instantiation takes place on line 1. It is instantiated with the circuit to compute the double auction for four players, one buyer and three sellers. The encoding of the double auction in the `MpcInstructionSet` can be found in Listing 4.6. This scenario is set up in lines 7 – 12 of Listing 4.2. The construct `|| partyID in set ...` is known as a generalised replicated parallelism and means one instance of the process Party is instantiated and run in parallel for each party id in $0, 1, 2, 3$. The `union` channel construct in lines 9 and 10 sets up a synchronisation between the ideal functionality `IdealF` and the party with id `partyID`, allowing the ideal functionality and each party to communicate (privately) on dedicated channels.

```
process IdealF  = AMPC(ThreePartyDoubleAuction)                          1

values                                                                   3
 inputs : seq of (seq of int) = [ [8,4, 2],[3,5,6],
                                  [0,1,10],[2,4,8] ]                     5

process Auction = || partyID in set {0,1,2,3} @ [{ }]                    7
                   ( Party(partyID,inputs(partyID),1)
                       [| {| input.i  | i in set {partyID} |} union      9
                          {| output.i | i in set {partyID} |} |]
                    IdealF                                               11
                   )
```

Listing 4.2: Auction setup

The ideal functionality `AMPC` in Listing 4.3 takes as input a `Circuit` (line 8) and then immediately enters the `Ready` action/state (line 35). The body of the `AMPC` process is its `Ready` action. The `Ready` action functions as an execution loop, executing one instruction from the circuit at a time. The *Inp* instruction is carefully explained in the following. For the remaining instructions see the full model[3]. An input instruction *Inp* is a record of two values: The id of the party (`pid`) from whom input is coming and the id (`vid`) by which the input can be referred in following instructions (lines 3 – 5). The first step `AMPC` does for an `Int` instruction is to receive input from the party with the id specified in the instruction (line 24). For *Add*, *Mul* and *Min* operations no communication is needed. The *AMPC* functionality stores the values in its own memory enabling it to compute these instructions without interaction.

[3]Available at http://www.cs.au.dk/~rwl/progrep/bedoza.zip

```
process AMPC = val circuit : Circuit @
begin
 state
   memory       : map ValueID to Value
   mycircuit    : Circuit := circuit

 operations
   NextInstr: () ==> MpcInstructionSet
   NextInstr() ==  <...>

 actions
   Ready = if (len(mycircuit) = 0)
   then Skip -- Computation done
   else  (dcl curinstr : MpcInstructionSet := NextInstr() @
    cases curinstr :
    -- Input instruction
    mk_Inp(pid,vid) ->
       input?pid?v ->
        memory(vid) := v ; Ready,
    -- Output instruction
    <...>
    -- Add instrucion
       mk_Add(vid1,vid2,vid3) ->
         (dcl v1 : Value := memory(vid1),
              v2 : Value := memory(vid2) @
          memory(vid) := v1 + v2,
    -- Mul instruction
    <...>
       others -> Stop
        end
      )
  @
    Ready
end
```

Listing 4.3: The Ideal functionality.

Informal Refinement of the Single System Model The refinement considered here is for the property of indistinguishability inspired by [Can00]. The goal is to show that by "executing" the idealised model above, no more information is revealed about the inputs (the secure ones indicated by requirements **R6** – **R8**) and the intermediate states than in an "execution" of the single model. In this case it is not hard to see intuitively. However, mapping the operations *AddSeller* and *AddBuyer* from the single system model to the model above is not trivial. Each invocation of *AddSeller* is mapped into a small cirtuit of *Inp*, *Add* instructions. Finally the *AnnounceMarketClearingPrice* is mapped into two min invocations in the concrete case here of three prices. Also interesting to observe here is that in the specification of AddSeller, another party is implicitely added to the computation. When this happens the arithmetic circuit needs to be rebuilt and the "Raw Material" needs to be generated again.

This model is very different from the Single Model system. The fundamental approach is different in that the Single Model System is specialised towards solving the double auction directly. This model, on the other hand, has the flavor of an interpreter. The Double Auction is encoded in the data given as input to the model rather than having the operations explicitly. This indicates some interesting challenges since the kind of refinement required would not only put restrictions on the refined system being built but also determines the allowed inputs.

4.5 Model for Double Auction Protocol

In this section the idealise distributed model is *refined* further into the real protocol description from [BDOZ11]. For simplicity only the on-line phase will be modelled. The full result from modelling the on-line phase of the BeDOZa protocol can be downloaded from `http://www.cs.au.dk/~rwl/progrep/bedozamodel.zip`. The current refinement is done by hand and is informal.

Model Purpose The *purpose* with this model is to illustrate feasibility of modelling the on-line phase of the BeDOZa protocol in CML and use this modelling for better comprehension. Also, as part of the ongoing process the purpose is to show that the requirements **R6** – **R8** are both in the model and fulfilled by this protocol. In summary the purposes are:

1 Demonstrate feasibility of modelling the BeDOZa protocol in CML, through argumentation for the security requirements **R6** – **R8**.

2 Provide a good benchmark model for CML tools to analyse. The intention is to run simulations on this model when the simulator is matured enough. Therefore, some readability and simplicity is scarified as all actions, functions and operations are made explicit.

Level of Abstraction In this model the focus is on the actions and communications going on in the protocol and the conceptual correctness of the values sent. To capture this goal, the level of abstraction chosen here will compute on "small" values represented by the *nat* data type in CML. The underlying assumption is that there exists methods and tools for computing on large numbers. Hence, for some translation of this model into code realising BeDOZa , basic operators of the model like $+, *,$ - need to be translated into appropriate use of a library for large numbers in $\mathbb{Z}_p$. With this it is a choice that low level integer computations are not in the scope for this model.

The Model Any of the details not included here can be found in the paper [Lau13] or in the comments of the model which can be downloaded from `http://www.cs.au.dk/~rwl/progrep/bedozamodel.zip`. At top level, the protocol is described in terms of a generic party process able to execute the instruction `Inp, Add, Mul, Min, Out` (Listing 4.5). Four instances of the generic party process are created, see Listing 4.4. The entry point of a model is a process. In the `Auction` process is the entry point for the model in this work. `Auction` is the parallel composition (on all channels) of the three sellers and the buyer process instantiated.

```
process Buyer   = Party(0,ThreePartyDoubleAuction,[ 8,4, 2],
                        singles,triples, keys, {0,1,2,3})
process Seller1 = Party(1,ThreePartyDoubleAuction,[ 3,5, 6]),
                        singles, triples, keys, {0,1,2,3})
process Seller2 = Party(2,ThreePartyDoubleAuction,[ 0,1,10],
                        singles, triples, keys, {0,1,2,3})
process Seller3 = Party(3,ThreePartyDoubleAuction,[ 2,4, 8],
                        singles, triples, keys, {0,1,2,3})
-- ENTRY POINT, RUN Auction
process Auction = Buyer || Seller1 || Seller2 || Seller3
```

Listing 4.4: Double Auction Circuit

The `Party` process is the main component and represents either a seller or a buyer, determined by the circuit and its inputs. The structure of the `Party` is presented in Listing 4.8. As arguments it takes:

- A party id making this party uniquely identifiable
- A circuit to compute
- A sequence of input values for the circuit
- A sequence of singles (shared representations from the pre-processing phase)
- A sequence of triples (shared representations from the pre-processing phase)
- A key map mapping MAC keys to shares and triples (from pre-processing)
- The set of all parties

PartyIDs are represented by *nat*'s. However, for readability one could consider using the token type instead. Each arithmetic instruction in the circuit is described by a record. The information needed in an input instruction is the id of the player that is instructed to input a value and the id under which the value should be stored. In Listing 4.5 the input instruction *Inp* is displayed with the definition of *MpcInstructionSet* and *Circuit*. Notice that since the parties are stateful, a sequence of instructions can describe a binary circuit referencing "memory" (ValueIDs) locations rather than building a tree-like structure. To see how this works out for the double auction consider Listing 4.6.

```
-- [*] vid  - is the id the inputted value shall have
-- [*] pid  - is the party inputting the value
Inp ::
        vid : ValueID
        pid : PartyID

-- An MPC circuit consists of any combination of the
-- above instructions.
MpcInstructionSet = Inp | Out | Add | Mul | Min

-- An MPC Curcuit is defined to be a sequence of
-- MpcInstructionSet
Circuit =  seq of MpcInstructionSet
```

Listing 4.5: Definition of MpcInstructionSet omitting Out Add Mul and Min instructions

The circuit in Listing 4.6 computes the double auction for four parties: one buyer and three sellers. The circuit is defined as a sequence of instructions. The circuit starts with twelve input instructions three for each of the four parties. The *Inp* instruction takes two operands: the id by which future computation can refer to the value and the id of the party who should provide the input, i.e. `mk_Inp(vid,pid)` instructs the protocol to let the party with id `pid` input his next value under id `vid`. In this way, the input values for the four parties are stored with ids 0 through 11 such that later instructions can refer to these ids as operands. The following two addition instructions compute the total supply at price 10 (line 9). The addition instruction takes three parameters: id of the left operand, id of the right operand and an id for the result. In this way the market deficiency (i.e. the difference between total supply and demand at each price, representing supply not bought or buying power not utilised) [4] is stored in the three values with ids 20, 21 and 22.

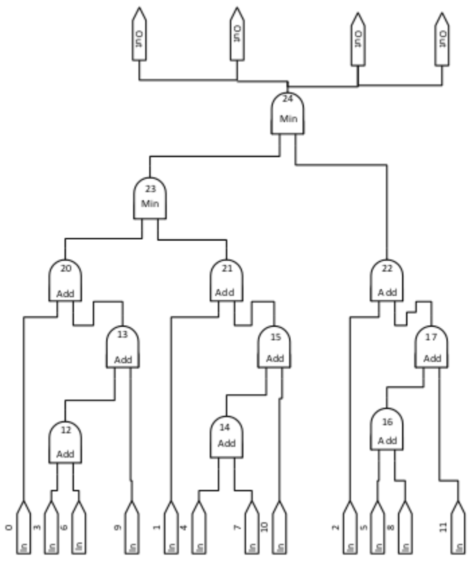
[4]For simplicity it is assumed that buyers and sellers supply negative and positive inputs respectively. This is different than from [BCD^{+}08].

Line 20 computes two min operations[5] storing the final result that is 24. The following four out instructions open this value to each of the four parties.

```
1   ThreePartyDoubleAuction: Circuit =
    [
3   -- instruct parties to input values
    mk_Inp( 0,0),mk_Inp( 1,0),mk_Inp( 2,0),
5   mk_Inp( 3,1), mk_Inp( 4,1), mk_Inp( 5,1),
    mk_Inp( 6,2), mk_Inp( 7,2), mk_Inp( 8,2),
7   mk_Inp( 9,3), mk_Inp(10,3), mk_Inp(11,3),
    -- Compute the total supply for price 10:
9   mk_Add(3,6,12), mk_Add(12,9,13),

11  -- Compute the total supply for price 20:
    mk_Add(4,7,14), mk_Add(14,10,15),
13
    -- Compute the total supply for price 20:
15  mk_Add(5,8,16), mk_Add(16,11,1
    -- Compute sum of demands and supplies
17  mk_Add(13,0,20), mk_Add(15,1,21), mk_Add
        (17,2,22),

19  -- Compute minimum distance
    mk_Min(20,21,23), mk_Min(23,22,24),
21
    -- Everybody learns the output
23  mk_Out(24,0), mk_Out(24,1), mk_Out(24,2)
    mk_Out(24,3)
25  ]
```



Listing 4.6: Double Auction Circuit

[5]In [BCD$^+$08] binary search is used to compute as few min operations as possible. This optimisation is not included in this description.

The instructions are carried out as the protocol between the four parties. Consider Listing 4.8 showing an overview of the `Party` process. It has the form of an execution loop taking instructions from the circuit until there are no more (lines 10 – 14 and 21). The `cases` construct branches on the kind of instruction to be executed (line 15 – 20). In the following, operation of the input instruction (`mk_Inp`) from Listing 4.5 is carefully described .

```
process Party =                                                              1
    -- Inputs:  partyID, circuit, inputs, singles, triples, keys,
        parties
    <...>                                                                    3
  @
begin                                                                        5
  state: -- myid,  mycircuit, myinputs,  mysingles,  mytriples, mykeys
     ,
        -- memory                                                            7
  operations -- helper functions
  actions                                                                    9
    ExecuteInstr = if (len(mycircuit) = 0) then Skip else
    (dcl instr : MpcInstructionSet := (hd mycircuit)                         11
     @
      mycircuit := tl mycircuit  ;                                           13
      ExecutionStep.instr ->
       cases instr :                                                         15
        mk_Inp(vid,pid) -> <...>
        mk_Out(vid,pid) -> <...>                                             17
        mk_Mul(op1id, op2id, rid) -> <...>
        mk_Add(op1id, op2id, rid) -> <...>                                   19
        mk_Min(op1id, op2id, rid) -> <...>
    ) ; ExecuteInstr                                                         21
  @ ExecuteInstr
end                                                                          23
```

Listing 4.7: Overview of the party process

The input instruction is asymmetric in that one `Party` is providing input whereas the remaining parties assist. The behaviours are conditioned on whether the `pid` of the `Inp` instruction is the same as the id of the process instance (line 2).

Consider the case for the process providing input. In lines 11 – 13 the private input `x` is computed by taking the next available input from the input sequence. Then, an opening of one of the pre-processed singles is carried out. In line 7, the input providing party is in parallel waiting for a share `shij` and its MAC `shijmac`. If the MAC check in line 11 passes, the share is stored in the map `receievdShares`. If the MAC check does not pass, the party stops and the system is deadlocked. If the *CheckMac* parses for all parties, then the mapping `receivedShares` contains all the shares necessary to construct the clear value of the single, which is done on line 20. The δ (`delta`) is computed as in the protocol and broadcast to all players in parallel in lines 22 – 23 via the *SendValue* channel. Now each party can update their state to represent $[x]$.

The case where a party participates in letting another party input a value is carried out in lines 41 – 48 of Listing 4.8.

```
mk_Inp(vid,pid) ->
  if (myid = pid) -- I am the one inputting x
  then
   (dcl x : Value := NextInput(),
     receivedShares : map PartyID to Value,
     single : Share := NextSingle() @
     (( [| {RequestOpen} |] i in set parties \ {myid} @ [{
         receivedShares }]
       (dcl j : ValueID := single.id @
                RequestOpen?i?myid?shij?shijmac ->
                 (dcl a : Value := shij @
                 if (CheckMac(shij,shijmac,mykeys(i)(j)))
                 then
                   receivedShares(i) := a
                 else
                   Stop
               ))
             )
             ;
             (dcl
                openedSingle : Value := ComputeClearSingle(single,
                    receivedShares)
              @
               ( [| {SendValue} |] i in set parties \ {myid} @ [{
                   openedSingle }]
                   SendValue!i!(ComputeDeltaForAdd(openedSingle,x))
                       -> Skip
               ) ; UpdateStateInp(vid,single,ComputeDeltaForAdd(
                   openedSingle,x))
             )
         )
       )
      else -- I'll participate in inputting x
      (dcl
         single : Share := NextSingle()
        @
         RequestOpen?myid!pid!(single.value)!(single.mac(pid)) ->
         SendValue!pid?delta ->
         AddConstantToKey(pid,single.id,delta)
```

Listing 4.8: The Input Instruction

Informal Refinement of the Ideal Model The ideal $AMPC$ functionality is replaced by a protocol such that only the `Party` process remains. The actions are significantly more complex than for the ideal versions: Addition and multiplication are now also handled directly as a result of communications between the parties. The circuit and party ids are exactly the same. The parameter for the number of expected outputs is replaced by the "raw material" needed to conduct the computation, which also captures the fact that the number of parties that can participate is determined in the off-line phase.

To show that the protocol model here is a secure refinement of the idealised distributed model it is argued that anyone observing an "execution" of the protocol model attains no more information about the secret inputs than could be attained from observing the ideal distributed model. The input instruction causes a quadratic number of shares and MACs to be communicated initially for the opening of a single. These values are guaranteed to be random and thus conveys no information. The party providing the input then broadcasts δ with his input masked by subtracting the value of the random single[6]. No more communication is carried out during the input phase. The communication is asymmetric in the sense that the party providing input is not sending any messages. However, this only conveys the information that this party is the input provider which is already public information that can be derived from the circuit. A similar argumentation can be made for the multiplication instruction.

The kind of refinement needed for this work should be able to show that the protocol model reveals no more information about the secret input than the ideal model. One idea is to introduce a three valued judgement $G = \mathit{Public}, \mathit{Random}, \mathit{Secret}$ for every state in the program. All (private) inputs in both models are initially judged as Secret. In the ideal model everything else is judged as Public. In the protocol model Singles and Triples are judged Random and everything not input or Triple/Single are judged Public. Then, if some analysis on the models could maintain this judgement step by step for every language construction, one could check whether inputs judged as Secret in the ideal model is a subset of the inputs judged as Secret in the protocol model. For example, the main reasoning above is that the result of adding or subtracting a random value with a secret reveals nothing about the secret to someone ignorant of the random value. Hence, a taste on how the analysis above could be realised through a structured operation semantics [Plo81] is the following rules for addition and subtraction:

$$[\mathit{SUB}]\ \frac{\Gamma \vdash \mathit{exp}1\colon \mathit{Random} \qquad \Gamma \vdash \mathit{exp}2\colon \mathit{Secret}}{\Gamma, \mathit{exp}1\colon \mathit{Secret} \vdash (\mathit{exp}1\text{-}\mathit{exp}2)\colon \mathit{Random}} \qquad [\mathit{PLUS}]\ \frac{\Gamma \vdash \mathit{exp}1\colon \mathit{Random} \qquad \Gamma \vdash \mathit{exp}2\colon \mathit{Secret}}{\Gamma, \mathit{exp}1\colon \mathit{Secret} \vdash (\mathit{exp}1 + \mathit{exp}2)\colon \mathit{Random}}$$

The SUB rule is read bottom-up and states: In the variable environment Γ extended with the judgement $\mathit{exp}1\colon \mathit{Secret}$ it is the case that $(\mathit{exp}1 + \mathit{exp}2)$ looks random to anyone ignorant of Secret values. The PLUS rule is read in a similarly way. Analysis with the similar flavor of reasoning can be found [Mor07].

[6]Note that the share of the party providing input has not been sent on any channel and therefore the value of the single is still unknown to anyone except for the party providing the input.

5 Conclusions and Future Work

An MBSE process is a branch of System Engineering where wanted properties with a system can be validated before attempting an implementation. Tools supporting a modelling formalism can assist in such processes by helping engineers check their designs for consistency and perform validation against requirements on models. The results from Chapter 3 show that modelling can help in bringing the combinatorial hassle with cryptographic parameters under control. Lifting this work to a CML model is an obvious next step for the collaboration with B&O. The result from Chapter 4 demonstrates that the COMPASS tool suite is ready to support MBSE processes and to be extended for further analysis. This section presents three ideas for future directions based on the experience in the previous sections.

5.1 Refinement

Refinement enables program construction via a number of separate correctness-preserving stages, which ideally can be understood in isolation. This is attempted informally in Chapter 4 through the construction of three models. The idea is to formalise an approach such that for some notion of executing the models, the following can be shown on the model level: the information about the private inputs and the intermediate states that can be deduced from observing the protocol model is at most what can be deduced in the idealised distributed model. Also, what can be deduced about the private inputs and the intermediate states in the idealised distributed model should be at most what can be deduced from the single system model.

Work on ignorance preserving refinement by Carroll Morgan in [Mor07] seems promising as he presents a weakest precondition calculus for preserving ignorance during refinement. As an example he describes a simple specification of another cryptographic primitive called Oblivious Transfer. He uses his logic to establish, via refinement, the correctness of Rivest's construction of Oblivious Transfer.

A natural and relevant next step is therefore to try to apply Morgan's techniques to reach the refinement expressed above for the work in Chapter 4. Then, another intriguing challenge is to generalise this to support an analysis on CML models in general (possibly by annotating the language). In recent work by Banks [Ban12], related work to this idea is done for Circus[1] where also the ideas by Morgan are lifted to UTP. As CML has a UTP semantics, carrying over Banks work to CML looks promising.

[1]Circus is very much like COMPASS combining another formalism (Z) with CSP http://www.cs.york.ac.uk/circus/index.html

5.2 Model Cryptographic Primitives

The MPC protocol presented in Chapter 4 is rather complex. To get a better understanding of modelling the underlying ideas, another future task would be to model simpler cryptographic primitives and/or other MPC protocols. Concrete constructions to look at could be protocols for Commitment Schemes or Oblivious Transfer (Oblivious Transfer is actually complete for MPC). Other constructions for MPC could for instance be the Yao construction MPC [Yao86] . Also, modelling well-established protocols like SSL/TLS would show immediate relevance for these tools in industry.

5.3 Proof of Concept

Yet another natural step would be to look at some applications, for instance of MPC (preferably an industry-driven ones), and capture requirements for a system with SoS characteristics. The idea would be to compare two developments, one using the COMPASS technologies developed here and one using established technologies in order to measure the effect of COMPASS methodology. People at Alexandra Institutet in Aarhus have already implemented the sugar beet auction and they are continuously implementing cryptographic (mostly MPC) protocols. A collaboration with some of these people has already been established so that the COMPASS technology can be tried out (and evaluated) for future implementations.

Bibliography

[Bal07] Laurent Balmelli. An overview of the systems modeling language for products and systems development. 2007.

[Ban12] Michael J. Banks. On confidentiality and formal methods. 2012.

[Bat09] Nick Battle. VDMJ User Guide. Technical report, Fujitsu Services Ltd., UK, 2009.

[BCD$^+$08] Peter Bogetoft, Dan Lund Christensen, Ivan Damgaard, Martin Geisler, Thomas Jakobsen, Mikkel Kroeigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. 2008.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *EUROCRYPT*, pages 169–188, 2011.

[Ber64] Brian J.L. Berry. Cites as Systems within System of Cites. *Papers and Proceedings of the Regional Science Association*, 13(1):149–163, January 1964.

[Bou56] Kenneth E. Boulding. General Systems Theory–The Skeleton of Science. *Management Science*, 2(3):197–208, April 1956.

[Can00] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. 2000. http://eprint.iacr.org/.

[CML$^+$12] Joey W. Coleman, Anders Kaels Malmos, Peter Gorm Larsen, Jan Peleska, Ralph Hains, Zoe Andrews, Richard Payne, Simon Foster, Alvaro Miyazawa, Cristiano Bertolini, and André Didier. COMPASS Tool Vision for a System of Systems Collaborative Development Environment. In *Proceedings of the 7th International Conference on System of System Engineering, IEEE SoSE 2012*, volume 6 of *IEEE Systems Journal*, pages 451–456, July 2012.

[CMNL12] Joey W. Coleman, Anders Kaels Malmos, Claus Ballegaard Nielsen, and Peter Gorm Larsen. Evolution of the Overture Tool Platform. In *Proceedings of the 10th Overture Workshop 2012*, School of Computing Science, Newcastle University, 2012.

[Col13] Joey W. Coleman. Second release of the COMPASS tool — tool grammar reference. Technical report, COMPASS Deliverable, D31.2c, January 2013.

[COM11] Description of Work: Comprehensive Modelling for Advanced Systems of Systems, 2011. Grant agreement no: 287829.

[Dam11] Ivan B. Damgaard. Definitions and results for cryptosystems. *Lecture Notes for CRYPTOGRAPHY Course at Computer Science Aarhus University.*, 2011.

[Est08] Jeff A. Estefan. Survey of model-based systems engineering (mbse) methodologies. 2008.

[FL98] John Fitzgerald and Peter Gorm Larsen. *Modelling Systems – Practical Tools and Techniques in Software Development*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK, 1998. ISBN 0-521-62348-0.

[FL09] John Fitzgerald and Peter Gorm Larsen. *Modelling Systems – Practical Tools and Techniques in Software Development*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK, Second edition, 2009. ISBN 0-521-62348-0.

[FLM$^+$05] John Fitzgerald, Peter Gorm Larsen, Paul Mukherjee, Nico Plat, and Marcel Verhoef. *Validated Designs for Object–oriented Systems*. Springer, New York, 2005.

[FLS08] John Fitzgerald, Peter Gorm Larsen, and Shin Sahara. VDMTools: Advances in Support for Formal Modeling in VDM. *ACM Sigplan Notices*, 43(2):3–11, February 2008.

[FLV08] J. S. Fitzgerald, P. G. Larsen, and M. Verhoef. Vienna Development Method. *Wiley Encyclopedia of Computer Science and Engineering*, 2008. edited by Benjamin Wah, John Wiley & Sons, Inc.

[HFK$^+$10] Cecilia Haskins, Kevin Forsberg, Michael Krueger, David Walden, and R. Douglas Hamelin. *SYSTEMS ENGINEERING HANDBOOK*. 2010.

[Hoa85] Tony Hoare. *Communication Sequential Processes*. Prentice-Hall International, Englewood Cliffs, New Jersey 07632, 1985.

[Jon99] Cliff B. Jones. Scientific Decisions which Characterize VDM. In J.M. Wing, J.C.P. Woodcock, and J. Davies, editors, *FM'99 - Formal Methods*, pages 28–47. Springer-Verlag, 1999. Lecture Notes in Computer Science 1708.

[Kle99] Paul Klemperer. Auction theory: A guide to the literature. Microeconomics, EconWPA, 1999.

[Lau13] Rasmus Lauritsen. A case study applying systems of systems modelling to multi-party computation. *In preparation for the 11th Internation Conference on Software Engineering Formal Methods*, 2013.

[LFW09] Peter Gorm Larsen, John Fitzgerald, and Sune Wolff. Methods for the Development of Distributed Real-Time Embedded Systems using VDM. *Intl. Journal of Software and Informatics*, 3(2-3), October 2009.

[LH$^+$96] P. G. Larsen, B. S. Hansen, et al. Information technology – Programming languages, their environments and system software interfaces – Vienna Development Method – Specification Language – Part 1: Base language, December 1996. International Standard ISO/IEC 13817-1.

[LL12] Rasmus Lauritsen and Lasse Lorenzen. Towards an extensible core model for Digital Rights Management in VDM. In *Proceedings of the 10th Overture Workshop 2012*, School of Computing Science, Newcastle University, 2012.

[Mai96] Mark W Maier. Integrated modeling: A unified approach to system engineering. *Journal of Systems and Software*, 32(2):101–119, 1996.

[Mai98] Mark W. Maier. Architecting principles for systems-of-systems. *Systems Engineering*, 1(4):267–284, 1998.

[Mor07] Carroll Morgan. The shadow knows: Refinement and security in sequential programs. 2007.

[Nie10] Claus Ballegaard Nielsen. Dynamic Reconfiguration of Distributed Systems in VDM-RT. Master's thesis, Aarhus University, December 2010.

[NLF$^+$13] Claus Ballegaard Nielsen, Peter Gorm Larsen, John Fitzgerald, Jim Woodcock, and Jan Peleska. Model-based engineering of systems of systems. *In preparation for submission to ACM Computing Surveys*, January 2013.

[PCC$^+$04] Popescu, Bogdan C., Crispo, Bruno, Tanenbaum, Andrew S., Kamperman, and Frank L.A.J. A drm security architecture for home networks. In *Proceedings of the 4th ACM workshop on Digital rights management*, DRM '04, pages 1–10, New York, NY, USA, 2004. ACM.

[PL92] Nico Plat and Peter Gorm Larsen. An Overview of the ISO/VDM-SL Standard. *Sigplan Notices*, 27(8):76–82, August 1992.

[Plo81] Gordon D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, 1981.

[RHB97] A. W. Roscoe, C. A. R. Hoare, and Richard Bird. *The Theory and Practice of Concurrency*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1997.

[Soh07] David Sohn. Understanding drm. *Queue*, 5:32–39, November 2007.

[Sys10] OMG Systems Modeling Language (OMG SysML$^{\text{TM}}$). Technical Report Version 1.2, SysML Modelling team, June 2010. http://www.sysml.org/docs/specs/OMGSysML-v1.2-10-06-02.pdf.

[Tof07] T. Toft. *Primitives and applications for multi-party computation*. PhD thesis, Aarhus University, 2007.

[WBB$^+$12] Jim Woodcock, Victor Bandur, Jeremy Bryans, Ana Cavalcanti, and Andy Galloway. Compass modelling language definition 1 (d23.2). 2012.

[WC04] J. C. P. Woodcock and A. L. C. Cavalcanti. A Tutorial Introduction to Designs in Unifying Theories of Programming. In E. A. Boiten, J. Derrick, and G. Smith, editors, *IFM 2004: Integrated Formal Methods*, volume 2999 of *Lecture Notes in Computer Science*, pages 40 – 66. Springer-Verlag, 2004. Invited tutorial.

[WK04] Chi-Hung Chi William Ku. Survey of the technological as[ects of digital rights management. LNCS(3225):391–403, 2004. National University of Singapore.

[Yao82] Andrew C. Yao. Protocols for secure computations. pages 160–164, 1982.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). pages 162–167, 1986.

Rasmus Winther Lauritsen, Systems og Systems with Security, 2013

Department of Engineering
Aarhus University
Edison, Finlandsgade 22
8200 Aarhus N
Denmark

Tel.: +45 4189 3000