Applying Algebraic Specifications on Digital Right Management Systems

Nikolaos Triantafyllou[1], Katerina Ksystra[1], Petros Stefaneas[2] and Panayiotis Frangos[1]

[1] School of Electrical and Computer Engineering, National Technical University of Athens, Heroon Polytechniou 9, 15780 Zografou, Athens, Greece
{nitriant, katksy, pfrangos}@central.ntua.gr
[2] School of Applied Mathematical and Physical Sciences, National Technical University of Athens, Heroon Polytechniou 9, 15780 Zografou, Athens, Greece
petros@math.ntua.gr

Abstract. Digital Right Management (DRM) Systems have been created to meet the need for digital content protection and distribution. In this survey paper we present some of the directions of our ongoing research on the applications of the algebraic specification techniques on mobile DRM systems.

Keywords: DRM, Right Expression Languages, CafeOBJ, Institutions

1 Introduction

Digital Rights Management systems (DRMs) control many aspects of the life cycle of digital contents including consumption, management and distribution. Key component of such a system is the language in which the permissions on contents and constraints are expressed. Such languages are called Right Expression Languages (RELs). In this survey paper we present some of our ongoing research directions aiming to address some of the open problems of the DRM systems [1][2], by using algebraic specifications. Our research has been focused on Open Mobile Alliance [3], a well-known DRM standard.

Our paper is organized as follows: Section 2 gives a brief introduction to the concepts needed. Section 3 gives the outline of an abstract syntax and its specification for OMA REL [4]. OMA presents an algorithm that deals with multiple licenses referring to the same content. In section 4 we refer to the formal specification of this algorithm in the algebraic specification language CafeOBJ, and to the formal verification of a safety property. This algorithm is not the optimal to use as it explained in [2]. In section 5 we suggest a redesign of this algorithm based on Order Sorted Algebra [5] and hint at a formal proof that this algorithm is correct using the methodology presented in [6]. Finally we present some of our future goals.

Andrew V. Jones (Ed.): Proceedings of the 1st Imperial College Student Workshop (ICCSW '11), pp. 94–100. September 29th–30th 2011, London UK.

2 Prerequisites

2.1 Order Sorted Algebra

An Order Sorted Algebra (OSA) is a partial ordering $\leq$ on a set of sorts [5], where by sorts we usually mean a set of names for data types. This subsort relation imposes a restriction on an S-sorted algebra A, by s-sorted algebra we mean a mapping between the sort names and sub sets from the set A called the carries of sort s, that if $s \leq s'$ then $A_s \subseteq A_{s'}$ where A_s denotes the elements of sort s in A. Order sorted algebra (OSA) provides a way for several forms of polymorphism and overloading, error definition, detection and recovery, multiple inheritance, selectors when there are multiple constructors, retracts, partial operations made total on equationally defined sub-sorts, an operational semantics that executes equations as left-to-right rewrite rules and many more applications [5].

2.2 Observation Transition Systems, CafeOBJ, Specification and Verification

An Observation Transition System (OTS) is a transition system that can be written in terms of equations. We assume there exists a universal state space, say Y. Formally, an OTS S is a triplet $S = <O, I, T>$ where I is a subset of Y, the set of initial states of the machine and O is a set of observation operators. Each observer in O is a function that takes a state of the system and possibly a series of other data type values (visible sorts) and returns a value of a data type that is characteristic to that state of the system. Finally, T is the set of transition (or action) conditional functions. Each transition takes as input a state of the system and again possibly a series of data-type values and returns a new state of the system.

CafeOBJ [8] is an executable algebraic specification language, implementing equational logic by rewriting. Equations are treated as left to right rewrite rules. It can also be used as a powerful interactive theorem prover with the proof scores method [11]. With CafeOBJ each module defines a sort. A visible sort is the specification of an abstract data type. Hidden sorts are used to specify state machines. Sort ordering is simply declared using <. Concerning hidden sorts there are two kinds of operators; *action operators*, which change the state of a machine, and *observation operators*, which observe (and return) a specific value in a particular state of the machine. Equations are denoted using the keyword *eq* and conditional equations using the keyword *ceq*. Finally modules can be imported to other modules by either protecting them or extending them. An OTS can be specified in CafeOBJ, in a natural way. The state space corresponds to the values of a hidden sort. The initial states are denoted by a set of constants of the hidden sort. Observation operators are denoted as observers and transitions as action operators.

After creating the specification of a system in the OTS/CafeOBJ approach it is possible to verify that it holds several kinds of properties such as invariant and liveness. The former consists of properties that hold in any reachable state of the system and is the most explored in the bibliography. The latter consists of properties expressing that something will eventually happen in the system. There are few applications of this methodology in CafeOBJ to our knowledge. Finally we should

mention that it is possible to conduct falsification in this approach, meaning that the CafeOBJ system can guide you to find a counter example of the property you desired to verify.

Here we discuss the procedure of verifying invariant properties, liveness properties are discussed further in section 5. To verify an invariant property you first need to express it as a predicate in CafeOBJ terms. Next, show that this predicate holds in any initial state. This is done by asking CafeOBJ to reduce the predicate term in an arbitrary initial state. Then show that the property holds for any transition, the inductive step. Assuming that the predicate holds for an arbitrary state we ask CafeOBJ to reduce whether this implies that it holds for its successor state. The successor state is obtained by applying the transition rules to the above arbitrary state. CafeOBJ will either return true, false or an expression. If it returns true then the predicate holds on that step. When an expression is returned, this means that the machine cannot continue with the reductions. We must then assist CafeOBJ by case splitting the transition providing additional equations. If false is returned then we might need to find a lemma to discard this case, showing that it is not. If however, the state is reachable then the property does not hold and we have a counterexample.

3 Formal Semantics for OMA REL

OMA REL [4] is an XML based language. The part of the language that is responsible for the expression of rights is called the agreement model. Inside this model the constraints and permission of the language are defined. We have given algebraic semantics to the OMA REL component dealing with expressing the permissions and constraints on the contents. To achieve this, we first created an abstract syntax for the language. Then we translated this syntax to the CafeOBJ specification language in order to use its rewriting as a tool for validation.

A longer version of this part of our paper appeared in the proceedings of WiMob 2009 where we proposed the abstract syntax, its specification and some case studies [9]. Here we will just present one example. Assume that Alice has purchased the following license: *Display content named contentID1 as many times as you like, and Display or Print the content named contentID2 as many times as you like*. Having specified the above abstract syntax as rewriting rules in CafeOBJ we can validate sets of licenses. The first step is to specify in a script the license of interest. In our specification this is done by declaring the permission set as; `eq ps1=add (True==>contentID2 print, add(True==> contentID2 display, add(True ==> contentID1 display, em-permset))).`

The `add` operator adds a permission element to a set of permissions. A permission element is a triplet; *constraint* on *content* allows *action.* `em-permset` is a constant denoting an empty permission set. After the permission sets and licenses are created, it is easy to perform e-validation by simply asking the CafeOBJ compiler if the desired permission belongs to the permissions allowed by this license, using the following reduction `red Permitted(print,ebook, contentID2) in permissionSET` . Where `red` is a CafeOBJ command for term rewriting the given

expression and `permissionSET` is an operator denoting the above license, which contains `ps1`.

4 Verifying the OMA Rights Choice Algorithm

We study here the algorithm that comes together with the specification of the OMA REL and is responsible for choosing the most appropriate license to use, when there exist multiple licenses referring to a specific content. This algorithm has been formally specified and in addition, it has been proved to satisfy a minimal safety property in [11]. A longer version of this specification and verification appeared in the proceedings of WINSYS 2010 [10].

The property we proved can be seen in table 1. On the left column L and S are variables representing an arbitrary license and an arbitrary state of the system respectively. bestLic(S) is an observer that returns the best license to use in S and valid(S,L) an operator that checks if licenses L constraints hold in S. The proof of such properties follows the methodology presented in section 2.2. It required four extra lemmas: two were used to discard unreachable states of the OTS and the other two where lemmas on data-types that helped CafeOBJ with the reductions on these visible sorts.

5 Proposing a New Algorithm and its Verification

There exist some cases where we end up losing execution rights by using the algorithm currently in use [2]. Indeed, let us consider the set of licenses seen on table 2. If the user decides to use his right "*listen to song A*", using the above algorithm the DRM agent will choose License 1. But by doing so, License 1 will become depleted since it contains the count constraint denoted by "once". This results in the user losing the right to ever listen to song B with this set of licenses. This would not occur if the agent had decided to use License 2 to execute the right to listen to song A.

This loss has been characterized by monotonicity of licenses in [2] and is proven that any algorithm attempting to solve this problem as is, will be NP-complete. Our approach is based on Order Sorted Algebra [5]. We point out that licenses, as data types, can be represented by ordered sorts [12]. Next we identified that this loss of rights can only occur in some special cases.

Safety property for OMA Rights Choice Algorithm euationally and informally	
eq inv1(S,L) = ((L=bestLic(S)) and not (L= nil)) implies valid(S,L) .	*When a license is chosen, then the license is valid at that specific time.*

Table 1. Minimal Safety property to verify in the original OMA Rights Choice Algorithm, in natural language and CafeOBJ equational notation

Installed Licenses on a DRM agent	
License1: "you may listen to songs A or B once before the end of the month".	*License2: "you may listen to songs A or D ten times."*

Table 2. A set of installed license that can cause a loss of rights

Liveness Property for the OMA Rights Choice Algorithm	
eq lto(S, P) = ((color(S,P) = white) ∧ (P /in allowed(S))) |--> (color(S, P) = black) .	*If a right belongs to the installed licenses and is colored white leads to it being colored black.*

Table 3. Liveness property describing the no loss of rights in CafeOBJ notation and natural language

To capture this we inserted *Labels* on licenses that denote the following three things; Firstly, if the license contains one or more permissions, secondly the dominant constraint based on the original algorithm and finally if the license only allows one more execution. These labels allow us to provide an ordering on licenses that is used to determine what license to choose so that no loss will occur, while respecting the ordering on constraints in the original algorithm. A longer version of this paper, which contains the full algorithm together with case studies and Java implementation, can be found in [12].

5.1 Verification of the New Algorithm

We have proved that our new algorithm does not cause the same loss of rights as the algorithm currently in use. The full proof will be presented elsewhere. In this section we will only sketch our proof. The proving procedure has been broken down into the following steps. First we created a specification of our algorithm as an OTS in CafeOBJ. Next we constructed an OTS, modeling the behavior of installed licenses on a DRM agent, meaning how they evolve when the user executes rights. The two OTSs where composed behaviorally as described in [13] yielding a new OTS. In to order describe and prove the desired property we added to the OTS a coloring on rights via an observer. Initially all rights are white (unused). A right is colored black (used) in two cases. Firstly, if the right corresponds to user request and the algorithm chooses the license containing this right as the optimal. Secondly, a right, say B, should be colored black if the user makes a request, say A different then B, but A only belongs to the license that contains B and that license becomes depleted after the execution of the request A.

At the property describing the no loss of rights condition (table 3), `S,P` are variables denoting an arbitrary state and a permission respectively. `color(S,P)` is an observer that returns the color of permission `P` in state `S` and `|-->` is an operator we used to denotes *leads- to*. The deduction rules for `|-->, ensure` and `unless` are provided in a separate module called OTSLogic. This is a *Liveness* property and particularly a *leads-to* property [6]. The proof followed the methodology of [6]. The *lead-to* predicate was broken down into two *ensure* predicates of the form *p ensure q*, with p and q predicates. These types of properties require proving the "*unless case*; *p unless q"* and the "*eventually case*; *p eventually q"*. For the first we

need to prove that all of the transitions preserve the predicate; *(p(s) and* $\Box$*q(s))* $\Box$ *(p(s') or q(s'))*. While for the second we need to show that there exists an instance of a transition where; *(p(s) and* $\Box$*q(s))* $\Box$*q(s')* holds. Where *s* a state of the OTS and *s'* is derived from *s* by applying a transition rule.

6 Conclusions

We have presented some of our ongoing work on the applications of algebraic specifications to mobile DRM systems. Also, we have shown how various techniques, from rewriting to theorem proving, can help solve some of the open problems of the field and also provide insights that can lead to the development of novel applications to DRMs as already shown with the proposed algorithm.

One of the main concerns with DRM is interoperability. There exist many different REL and DRM systems that cannot work together, so at the moment it is usually not possible to transfer licenses from one environment (mobile) to another (media player).We have started to address this problem by defining an Institution for OMA REL ([7]). Using the well-known abstract model theory tools of Institutions we intend to create a mechanism for translating licenses from one system to another via Institution morphisms in such a way so the meaning of the license is preserved by using semantic techniques.

References

1. Pucella, R., Weissman, V.: A Logic for Reasoning about Digital Rights. In: Proc. 15th IEEE Workshop on Computer Security Foundations (CSFW '02), pp. 282 (2002)
2. Barth A., Mitchell J. C., Managing digital rights using linear logic, 21st Symposium on Logic in Computer Science, pp. 127 – 136, 2006
3. Open Mobile Alliance Digital Rights Management Version 2.1, approved in 2008.
4. Open Mobile Alliance, DRM Rights Expression Language, approved version 2.1, 2010.
5. Goguen, J,A., Meseguer J.: Order-sorted algebra I: equational deduction for multiple inheritance, overloading, exceptions and partial operations, Theoretical Computer Science Volume 105, Issue 2, pp. 217-273 (1992)
6. Ogata, K., Futatsugi, K.: Proof Score Approach to Verification of Liveness Properties. IEICE Transactions on Information and Systems, Volume E91.D, Issue 12, pp. 2804-2817 (2008).
7. Goguen, J,A., Burstall, R., M.: Institutions: abstract model theory for specification and programming
8. Diaconescu, R., Futatsugi, K.: CafeOBJ Report: the Language, Proof Techniques, and Methodologies for Object-Oriented Algebraic Specification. AMAST Series in Computing, Vol. 6, World Scientific, Singapore, 1998
9. Triantafyllou, N., Ouranos I., Stefaneas, P.: Algebraic Specifications for OMA REL Licenses. IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), pp. 376-381 (2009)
10. Tiantafyllou, N., Ouranos, I., Stefaneas, P., Frangos, P.: Formal Specification and Verification of the OMA License Choice Algorithm in the OTS/CafeOBJ Method. Wireless Information Networks and Systems (WINSYS) 2010, pp. 173-180 (2010)

11. Futatsugi, K., Ogata, K.: Proof Scores in the OTS/CafeOBJ Method. In: Najm, E., Nestmann, U., Stevens, P. (eds.) Formal Methods for Open Object-Based Distributed Systems. LNCS, vol. 2884, pp. 170--184, Springer, Berlin, (2003)
12. Tiantafyllou, N., Ouranos, I., Stefaneas, Ksystra, K., P., Frangos, P.: Redesigning OMA Choice Algorithm, submitted for publication - available at CoRR abs/1102.1547: (2011)
13. Iida, S., Futatsugi, K., Diaconescu, R.: Component-based algebraic specification: behavioural specification for component-based software engineering. Seventh OOPSLA Workshop on Behavioral Semantics of OO Business and System Specification (1999)