FORMAL SPECIFICATION AND VERIFICATION OF THE OMA LICENSE CHOICE ALGORITHM IN THE OTS/CAFEOBJ METHOD

Nikolaos Triantafyllou, Iakovos Ouranos
School of Electrical and Computer Engineering, National Technical University of Athens, Athens, Greece

Petros Stefaneas, Panayiotis Frangos
School of Applied Mathematical and Physical Sciences, National Technical University of Athens, Athens, Greece
School of Electrical and Computer Engineering, National Technical University of Athens, Athens, Greece

Keywords: Mobile Digital Rights Systems, OMA-Rights Expression Language, CafeOBJ, Observational Transition Systems, License choice algorithm.

Abstract: OMA-Digital Rights Management System is a standard proposed by the Open Mobile Alliance (OMA) for protecting digital content distribution via mobile networks. To solve the decision problem, in the case that multiple licenses refer to the same content, OMA suggests a license choice algorithm. This algorithm ensures the fine grained consumption of contents. CafeOBJ is a new generation algebraic specification language. We apply the OTS/CafeOBJ method to formally model, specify and verify the above mentioned license choice algorithm. More specifically, we develop the mathematical model of the OMA decision algorithm as an OTS, a kind of transition system expressed in an equational CafeOBJ specification style. Finally, we verify that this algorithm fulfills the following safety property: Whenever a license is chosen for a given content, then the license is valid at that specific time.

1 INTRODUCTION

Digital Rights Management Systems (DRMSs) are apart from a set of cryptographic methods to forbid unauthorized usage, a method for expressing permissions and obligations on a content. A set of such permissions and obligations is called a license. Such licenses are written in languages known as Rights Expression Languages (RELs). The most commonly used such languages today are ODRL (Iannella, 2002), XrML (ContentGuard, 2007) and MPEG REL (Rightscom, 2007)

Open Mobile Alliance (OMA) is an organization created to be the center of mobile service enabler specification work. OMA REL (OMA-TS-DRM-REL-V2_0-020060303-A, 2007) is a digital rights expression language that specifies the syntax and the semantics of rights governing the usage of DRM contents in the OMA DRM system (OMA-TS-DRM-REL-V2_0-020060303-A, 2007). It is based on ODRL and is defined as a mobile profile of it. Together with the specification of the language, in (OMA-TS-DRM-REL-V2_0-020060303-A, 2007), OMA proposes an algorithm that comes to lift the burden of the user when he faces the problem of having more than one license that refers to the same content. This algorithm takes into consideration the constraints each license contains, and decides for the user the "best" suited license to use for a desired action on the content. Since this algorithm does not have an official name, we will refer to it for the rest of the paper as the "Choice Algorithm".

Digital Right protected contents are a commodity and as such their success highly depends on their acceptance by the market. DRM systems need to protect the interests of both the publisher and the end user. As a result, licenses intended to work in one manner but end up working differently cause discomfort to both of the parties involved. The end user is dissatisfied because they may end up purchasing a service that does not respond to the advertised manner and the producer might end up

Triantafyllou N., Ouranos I., Stefaneas P. and Frangos P. (2010).
FORMAL SPECIFICATION AND VERIFICATION OF THE OMA LICENSE CHOICE ALGORITHM IN THE OTS/CAFEOBJ METHOD.
In *Proceedings of the International Conference on Wireless Information Networks and Systems*, pages 173-180
DOI: 10.5220/0002944501730180

losing the consumption control of his contents if the licenses behave unexpectedly.

For these reasons we believe that formal semantics and specification in order to achieve the unambiguous licenses and software that control them, is the key to ensure the success and longevity of these commercial DRM systems. In this paper we formally analyze the Choice Algorithm and verify that it behaves in a correct manner using the Observational Transition System (OTS) / CafeOBJ method (Diaconescu, Futatsugi, 1998, CafeOBJ home page, 2009), thus showing both how such techniques can be applied to the field of Mobile DRM and provide at the same time a proof of a fundamental property for the algorithm

The rest of the paper is organized as follows: Section 2 gives a brief introduction to the OTS/CafeOBJ method (Ogata, Futatsugi, 2006, Ogata, Futatsugi, 2003). In section 3 we describe the Choice Algorithm and its specification as an OTS in CafeOBJ while in section 4 we present the invariant property and the corresponding proof scores. Finally, section 5 concludes the paper.

2 THE OTS/CAFEOBJ METHOD

2.1 Observational Transition Systems

An Observational Transition System, or OTS (Ogata, Futatsugi, 2006, Ogata, Futatsugi, 2003), is a transition system that can be written in terms of equations. We assume that there exists a universal states space called Υ and we also assume that each data type we need to use in the OTS, including the equivalence relationship for that data type, has been declared in advance. An OTS S is the triplet $< O, I, T >$ where:

- O: is a finite set of observers. Each $o \in O$ is a function $o : \Upsilon \to D$, where D is a data type that may differ from observer to observer. Given an OTS S and two states $u_1, u_2 \in \Upsilon$, the equivalence $(u_1 = s = u_2)$ between them wrt. S is defined as $\forall o \in O, o(u_1) = o(u_2)$.
- I: is the set of initial states such that $I \subseteq \Upsilon$.
- T: A set of conditional transitions. Each $\tau \in \mathrm{T}$ is a function $\tau : \Upsilon \to \Upsilon$, such that $\tau(u_1) =_S \tau(u_2)$ for each $u_1, u_2 \in \Upsilon / =_S$. For each $u \in \Upsilon, \tau(u)$ is called the successor state of u wrt τ. The condition c_τ of τ is called the effective condition. Also for each $u \in \Upsilon, \tau(u) = u$ if $\neg C_\tau(u)$.

Observers and transitions may be parameterized. Generally observers and transitions are denoted as $o_{i_1,\ldots,i_m}$ and $\tau_{i_1,\ldots,i_n}$ respectively provided that $m, n \geq 0$ and there exist data types D_k, D where $k = i_1,\ldots,i_m, j_1,\ldots,j_n$.

2.2 OTSs in CafeOBJ

An OTS *S* is written in CafeOBJ. The universal state space *Y* is denoted by a hidden sort, say *H*. An observer *oi*1 , ...,*im* ∈ *O* is denoted by a CafeOBJ observation operator. We assume that there exist visible sorts *Vk* and *V* corresponding to the data types *Dk* and *D*, where *k* = *i*1, ..., *im*. The CafeOBJ observation operator denoting *oi*1, ...,*im* is declared as follows: bop o : Vi1 ... Vim H -> V. Any state in *I*, namely any initial state, is denoted by a constant, say *init*, which is declared as follows: op init :-> H

The initial value returned by *oi*1 , ...,*im* is denoting by the term *o*(*Xi*1 , ..., *Xim* , *init*) where *Xk* is a CafeOBJ variable whose sort is *Vk*, where *k* = *i*1, ..., *im*. A transition τ*j*1 , ..., *jn* ∈ *T* is denoted by a CafeOBJ action operator as follows: bop a : Vj1 ... Vjn H -> H with *Vk* a visible sort corresponding to the data type *Dk*, where *k* = *j*1, ..., *jn*.

Each transition is defined by describing what the value returned by each observer *oi*1 , ...,*im* in the successor state becomes when τ*j*1 , ..., *jn* is applied in a state *u*. When *c* − τ*j*1 , ..., *jn* (*u*) holds, this is expressed generally by a conditional equation that has the form ceq o(Xi1,...,Xim, a(Xj1,...,Xjn,S)) = e-a(Xj1,...,Xjn, Xi1,...,Xim, S) if c-a(Xj1,...,Xjn, S) . S is a CafeOBJ variable whose sort is *H* and Xk is a CafeOBJ variable whose sort is *Vk*, where *k* = *j*1, ..., *jn*. a(*Xj*1 , ..., *Xjn* , *S*) denotes the successor state of S w.r.t. τ*j*1 , ..., *jn* plus *Xj*1 , ..., *Xjn* . e-a(*Xj*1 , ..., *Xjn* , *Xi*1 , ..., *Xim*, *S*) denotes the value returned by *oi*1 , ...,*im* in the successor state. c-a(*Xj*1 , ..., *Xjn* , *S*) denotes the effective condition *c* − τ*j*1 , ..., *jn* . The value returned by *oi*1 , ...,*im* is not changed if τ*j*1,..., *jn* is applied in a state *u* such that ¬*c* − τ*j*1, ...,*jn* (*u*), which can be written generally as follows: ceq o(Xi1,Xim, a(Xj1,Xjn,S)) = o(Xi1,Xim, S) if not c-a(Xj1,Xjn, S) .

3 OMA LICENSE CHOICE ALGORITHM

Because the process of manually choosing which license to use, when dealing with multiple licenses referring to the same contents, may cause discomfort to the user, the following algorithm is proposed by OMA (OMA-TS-DRM-REL-V2_0-020060303-A, 2007):

1) Only rights that are valid at the given time should be taken into account from the algorithm.
2) Rights that are not constrained should always be preferred over constrained rights.
3) Any right that includes a Datetime constraint, and possibly others, should be preferred over rights that are constrained but do not have such a restriction.
4) If there exist more than one rights with a Datetime constraint, the one with the further in the future End element should be preferred.
5) If there exist a choice between many rights and none of them contains a Datetime constraint the one containing an Interval constraint should be preferred if there exists such.
6) Rights that contain a Count constraint should be preferred after rights that contain a Timed-Count constraint.

This algorithm basically states an ordering on the constraints of the licenses. Based on this ordering, the decision for the best license to use is made between licenses that are valid at the given time, and refer to the desired right.

We will clarify the above algorithm through the use of an example. We assume that Alice has the following two licenses installed in her DRM agent:

License 1: You can listen before the tenth of the month songs A or B one time

License 2: You can listen to songs A or C up to ten times

Notice that the first license contains a datetime constraint, which is stated above as the constraint "*before the tenth of the month*". The right to listen to songs A or B is further constrained using a count constraint stated by "*up to one time*", while the second license contains only a count constraint stated as "*up to ten times*".

We assume the case where Alice decides to use these licenses installed in her DRM agent to listen to song A. The DRM agent is supposed (OMA-TS-DRM-REL-V2_0-020060303-A, 2007) to include an implementation of the above License Choice Algorithm.

When Alice gives the command to listen to song A, the algorithm will search the licenses installed in her DRM agent to find a corresponding right that is valid at that given time based on their constraints, say "*listen song A*", will check the constraints of the "matching" licenses and based on the ordering provided by the OMA Choice Algorithm it will select a license as the best to use the "*listen song A*" right, and finally exercises it. In our example this license will be the first one.

3.1 Basic Data Types

Before introducing the OTS model of the OMA Choice Algorithm, we define the basic data types we need to use.

1) `Action`: Defines the data type of an action that a user wants to apply to a content, as they are defined by OMA REL.
2) `Content`: Is the data type that represents the unique identification of a specific content.
3) `Use`: represents an action enforced to some content. This is defined in our model as $use := action, content$. The only actions allowed by OMA REL are *play*, *display*, *print*, *execute* and *export*.
4) `Cons`: Defines the data type of the constraints defined by OMA REL.
5) `Right, RightSET`: define a granted right by a license and a set of such rights respectively. A right is an expression of the form `U under C`, where U is a Use and C a Cons. RightSET is defined as a super type (Right is a sub sort of RightSET, in CafeOBJ terms) of the data type Right. It has added functionalities such as a "belong" operator (in) and the union of sets. In addition to those, in the definition of these data types we define operators "`check`", "`belong`" and "`?`". The "`check`" operator takes a Use and a Right element and checks whether that Use belongs to that Right. The "`belong`" operator works in a similar way but checks if a Use belongs to a RightSET. Finally the "`?`" operator takes a RightSET and returns true or false if the constraints of it hold or not respectively.
6) `Lic, LicSET`: These data types define a license and a set of licenses respectively. As in Right and RightSET, Lic is a sub sort of LicSET. A Lic is constructed by operator, `C about R`, where C is a Cons and R is a RightSET. Lics are defined likewise, to

capture the fact that in a license written in OMA REL a constraint can apply to a set of rights that can be further more constrained and refer to different contents. LicSET is again defined as a set of Lic data types with the usual set functions. Again, we use the operators "belong", "belongLS", "existLi", "exist" and "?". The "belong" operator determines whether or not a Use element belongs in a Lic element and "belongLS", if a Use element belongs to a set of licenses. The operator "existLi" checks whether there exists a suitable right for a Use element in a license, while "exist" performs the same check for a set of licenses. Finally the "?" operator determines if a license is valid at a given time. As an example of this specification in CafeOBJ, we present the corresponding code for Use data type:

Table 1: Definition of the Use data type.

```
mod* USe { pr(ACtion + COntent)
        [Use]
op _,_ : Action Cont -> Use {constr}
op none : -> Use
ops play display print execute
export: -> Action
op _ = _ : Use Use -> Bool {comm}
var U : Use
eq (U = U) = true .  }
```

3.2 OTS Model and its Specification

According to the methodology presented in section 3 the hidden state space is denoted by a hidden sort, **[Sys]**.

The observers we will use are: *lics, chosenSet, bestLic, user and valid.* The *lics* observer is specified by the behavioral operator `bop lics: Sys -> LicSet` and returns the installed licenses on the agent at any given time. The observer *chosenSet* defined in CafeOBJ by the behavioral operator `bop chosenSet: Sys -> LicSet` denotes the set of licenses that refer to the same content and the same action as the users choice at the current time. The *bestLic* observer defines the most appropriate license to use for a user request according to the OMA Choice Algorithm. This observer is defined in CafeOBJ by the following behavioral operator `bop bestLic: Sys -> Lic.` The observer *user* denotes the current user request and it is specified in CafeOBJ as `bop user: Sys -> Use.` Finally the *valid* observer shows whether or not a license is valid at a given time and is denoted as `bop valid: Sys Lic -> Bool`

The actions we used in order to specify the OMA Choice Algorithm are the following: *request*, *use*, *hold*, *NoHold*.

1) The action *request* is declared through the action operator `bop request: Sys Action Cont -> Sys` and defines the request by a user to use an action on some content. The transition can only occur if there is such a license in the set of currently installed license of the agent. This is specified using the effective condition for this transition

```
eq            c-request(S,A,CONT)=
existsLi((A,CONT);lics(S)).
```

2) The action *use* defines the consumption of a usage right on some content. This is denoted by the following action operator: `bop _ use _ with _: Sys Use Lic -> Sys.` This transition can only occur if the license is valid at that given time, is the best license as that is defined by the Choice Algorithm and finally the usage right about that content belongs to that license. This is defined in CafeOBJ again through the effective condition of the transition rule:

```
eq       c-use(S,A       ,CONT),L)
=exist((A,CONT);L) and valid(S ,L)
and (L=bestLic(S)) .
```

3) *Hold* is used to specify the fact that after the execution of a usage right, that right remains valid, i.e. its constraints are not depleted. This is defined by the action operator: `bop hold: Sys Lic -> Sys.` Since the only time a usage right can be exercised is if it belongs to the best license, defined by the OMA Choice Algorithm, and for this transition to occur it must remain valid the effective condition of this transition rule is defined as:

```
eq c-hold(S,L)=(L in lics(S)) and
(L)? and (L= bestLic(S)) .
```

4) *NoHold* is the opposite transition of hold. That is after the execution of the usage right, it is no longer valid. This transition is defined in CafeOBJ as: `bop NoHold: Sys Lic -> Sys.` The only difference with the above effective condition is that for this transition to

occur the license must no longer be valid. The is specified as:

```
 eq     c-NoHold(S,L)=     (L     in
lics(S))andnot(L)?              and
(L=bestLic(S)).
```

As stated in section 3, a state of an OTS is characterized by the values the observation operators return. Here we present such an example for our specification, for the case of the *NoHold* transition:

Table 2: Definition of the NoHold action in CafeOBJ.

```
1.op c-NoHold : Sys Lic -> Bool
2.eq c-NoHold(S,L)=(L in lics(S))
       and not(L)? and
L=bestLic(S)).
3.ceq NoHold(S,L) = S
       if not c-NoHold(S,L) .
4.ceq lics(NoHold(S ,L))
= (lics(S) del L) if c-NoHold(S,L) .
5.ceq chosenSet(NoHold(S,L)) = empty
       if c-NoHold(S,L) .
6.ceq bestLic(NoHold(S,L))=nil
       if c-NoHold(S,L) .
7.ceq user(NoHold(S,L))=none
       if c-NoHold(S,L) .
8.ceq valid(NoHold(S,L),L)= false
       if c-NoHold(S,L) .
```

In the above table the numbers at the beginning of each line are not part of the code. In line 1 the signature of the effective condition for the transition rule is declared, as a predicate that takes a system state and a license. In line 2 the equation defining the effective condition is declared as: the license belongs on the set of licenses installed in the agent in S, its constraints no longer hold in S and it is the license chosen by the OMA Choice Algorithm in S. In line 3 we declare that the successor state of the application of the `NoHold` transition is S if the effective condition does not hold.

Lines 4 to 8 denote the values returned by the observers of the OTS when the `NoHold` transition is applied to the arbitrary state S for an arbitrary license L.

In line 4 we declare that the `lics` observer returns the same set of licenses as the one in S where license L is removed from the set, if the effective condition of the transition holds. In line 5 the observer `chosenSET` observes the empty set, if the effective condition of the transition holds. In line 6 the observer `bestLic` observes nil, a dummy license constant that denotes the fact that no license exists. In line 7 the `user` observer, observes again a dummy use constant to denote that no user choice is made in that state. Finally in line 8 the `valid` observer for the license L returns false, when the effective condition holds.

4 VERIFICATION OF OMA CHOICE ALGORITHM

The invariant property that corresponds to the proper function of the algorithm is the following: *Whenever a license is chosen for a given content, then the license is valid at that specific time.* In order to prove such a property, several steps need to be taken (Futatsugi, Goguen, Ogata 2005, Ogata, Futatsugi, 2008).

Express the property in a formal way as a predicate, say *invariant pred(p,**x**)*, where *p* is a free variable for states and ***x*** denotes other free variables of *pred*.In a module, usually called INV, *pred(p,**x**)* is expressed in CafeOBJ

```
op inv : H V -> Bool
eq inv(P, X) = pred(P, X) .
```

where V the list of visible sorts corresponding to x, P is a CafeOBJ variable for H, the state space and X is a list of CafeOBJ variables for V.

In a proof score we show that our predicate holds at any initial state, say *init*.

```
open INV
red inv(init, x) .
close
```

red is a command that reduces a given term by regarding declared equations as left-to-right rewrite rules.

We write a module, usually ISTEP, where the predicate to prove in each inductive case is expressed in CafeOBJ, using two constants p, p' denoting any state and the successor state after applying a transition in the state:

```
op istep : V -> Bool
eq istep(X)=inv(p,X) implies inv(p',X).
```

For each case we write the proof score. It usually looks like the following:

```
open ISTEP
Declare constants denoting arbitrary
objects.
Declare equations denoting the case.
Declare equations denoting the facts if
necessary.
```

```
eq p' = a(p, y) .
red istep(x) .
close
```

Where y is a list of constants that are used as the arguments of CafeOBJ action operator a, which are declared in this proof score and denote arbitrary objects for the intended sorts. If istep(x) is reduced to true, it is shown that the transition preserves pred(p, x) in this case. Otherwise, we may have to *split the case*, may need some invariants that will be used as lemmas (*lemma discovery*), or we may show that the predicate is not invariant to the system.

Following the procedure presented above several lemmas where required to prove the desired invariant safety property. Their informal description is presented in table 3, where property 1 is the main property. The formal definition of invariant 1 in terms of CafeOBJ specification is:

```
op inv1 : Sys Lic -> Bool
eq inv1(S,L) = ((L = bestLic(S)) and
not (L = nil) ) implies valid(S,L) .
```

Table 3: Properties to verify.

No.	Informal definition of Properties to be proven
1	*Whenever a license is chosen for a given content, then the license is valid at that specific time.*
2	*If a license L is the chosen license by the OMA Choice Algorithm for a given set S and that license exits, i.e. is not nil then L belongs to the set S.*
3	*If the choice made by the OMA choice algorithm for the set R union S, where R is an arbitrary license containing one usage right and S is a set of Licenses, is not R nor is it a choice made solely on S then the chosen license is nil, i.e. not valid license is available*
4	*If the set of licenses contains only a single license, say L and the choice made by the OMA Choice Algorithm is not nil, i.e. there exists a valid license, then the choice is this license L*
5	*If the choice made by OMA Choice Algorithm when the license set contains two licenses L and L' is not nil, and if the choice made is not that made based on the second license L' then the chosen license is L*

Some of the properties presented above are stateless, meaning that they do not depend on the state of our OTS. We can prove such properties in a similar manner, the only difference relies on the fact that the induction does not occur on the states of our OTS but on the complexity of the data types,e.g. on the way a set is constructed

In module ISTEP, the following operator denoting the predicate to prove is declared and defined:

```
op istep1 : Lic  -> Bool
eq istep1(l)=inv1(s,l)implies
inv1(s',l)).
```

Using the follow proof passage we prove that the predicate holds for any initial state, say init.

```
open INV
red inv1(init,l) .
close
```

This proof passage returns true, so the base case is proven.

Next we write the proof passage for each of the transition rules used in the OTS. For the case of the `request,` transition rule.

```
open ISTEP
op s' : -> Sys .
op a : -> Action .
op c : -> Cont .
eq s' = request(s,a,c) .
red istep1(l).
close
```

The above case returns neither true nor false. In this case we need to split the case to help the CafeOBJ system reduce it. The most natural choice is to split the effective condition of the transition rule based on whether it holds or not.

```
open ISTEP
op s' : -> Sys .
op a : -> Action .
op c : -> Cont .
eq s' = request(s,a,c) .
eq c-request(s,a,c) = false .
red istep1(l).
close
```

The above refers to the case that effective condition of the transition is false, that is not `c-request(s,a,l)` and CafeOBJ returns true. Now that this case is covered we must cover its symmetrical one, i.e. when `c-request (s,a,l)` is true. This is shown in the following proof passage:

```
open ISTEP
op s' : -> Sys .
op a : -> Action .
op c : -> Cont .
eq s' = request(s,a,c) .
eq existsLi((a,c);lics(s)) = true .
red istep1(l).
close
```

Here we replaced the equation `c-request(s,a,c)= true` with the definition of `c-request(s,a,c)`, that is the equation `existsLi((a,c);lics(s)) = true`, to help CafeOBJ with the reductions.

In this case CafeOBJ returns again nor true nor false and we need more case splitting based on the returned formula (interactive computer – human proof).

```
open ISTEP
op s' : -> Sys .
op a : -> Action .
op c : -> Cont .
eq s' = request(s,a,c) .
eq existsLi((a,c) ; lics(s)) = true .
eq (l=chooseLic(belongLS((a,c);
lics(s)) ))=false .
red istep1(l).
close
```

The above proof passage corresponds to the case `existsLi((a,c);lics(s))` ∧ ¬ `l= chooseLic( belongLS((a,c);lics(s)))`.

This proof passage returns true, so we must continue with the symmetric case once again.

```
open ISTEP
op s' : -> Sys .
op a : -> Action .
op c : -> Cont .
eq s' = request(s,a,c) .
eq existsLi((a,c) ; lics(s)) = true .
eq l=chooseLic(belongLS((a,c);lics(s)))
.
red istep1(l).
close
```

This case corresponds to `existsLi((a,c); lics(s))` ∧ `l=chooseLic(belongLS((a,c); lics(s)))` and CafeOBJ returns again nor true nor false. Once again we need to split the case based on the formula returned by CafeOBJ.

Following this procedure we reach state `existsLi((a,c);lics(s))` ∧ `l=chooseLic( belongLS((a,c);lics(s)))` ∧ ¬ `(chooseLic( belongLS((a,c);lics(s)))=nil)` ∧ ¬ `(chooseLic(belongLS((a,c);lics(s))inlic s(s))` where CafeOBJ has returned true to all the symmetrical subcases.

Here CafeOBJ returns nor true nor false again. Normally we would, and can, apply more case splitting, but we notice that these predicates cannot hold simultaneously in our OTS (another example of the interactive proving procedure). So we can use these contradicting predicates to conjuncture a lemma and discard this case. These predicates constitute lemma 2 we described in table 3 and it is formally defined as:

```
eq  inv2(LS,L,A,C)=((L=chooseLic
(belongLS ((A,C);LS)))and not(L=nil))
implies(L in LS).
```

Using this lemma we can discard this case as is show in the following proof passage:

```
open ISTEP
op s' : -> Sys .
op a : -> Action .
op c : -> Cont .
eq s' = request(s,a,c) .
eq existsLi((a,c) ; lics(s)) = true .
eq
l=chooseLic(belongLS((a,c);lics(s))).
eq  (chooseLic(belongLS((a,c);lics(s)))
= nil) = false .
eq  (chooseLic(belongLS((a,c);lics(s)))
in lics(s)) = false .
red          inv2(lics(s),l,a,c)implies
istep1(l) .
close
```

CafeOBJ returns true for the above proof passage and hence, once we prove lemma 2, this concludes the proof for the request transition rule of our safety property.

Applying the same technique, CafeOBJ returned true for all transitions. Finally, all the lemmas presented in table 3 were proven using the same technique and thus our proof concludes.

5 CONCLUSIONS AND FUTURE WORK

We have modeled OMA Choice Algorithm as an Observational Transition System in CafeOBJ, and verified that the algorithm possesses an important invariant property using CafeOBJ system as an interactive theorem prover. We are not the first to use algebraic specification techniques for modeling and verification of Digital Rights Management Systems (Xiang, Bjørner, Futatsugi, 2008). This

paper is a part of our work in modeling, specification and verification of algorithms and protocols used in mobile settings, using algebraic specification techniques (Ouranos, Stefaneas, 2007).

We have also proposed an abstract syntax for OMA Rights Expression Language in (Triantafyllou, Ouranos, Stefaneas, 2009). Some problems for the OMA Choice Algorithm are presented in (Barth, Mitchell, 2006). More specifically, let us consider the example of section 3.In this example if the user tries to exercise the right *"play song A"* the OMA Choice Algorithm will decide that the best license to use is license A. By doing so, the user is deprived of the right to listen to song B because license A will no longer be valid after the execution of the above right. So, the user ends up losing some of the rights the initial license set contained without exercising them. This malfunction could have been avoided if the OMA Choice Algorithm decided the most fitting license to use for the right "*play song A*" was license B. After the execution of the right the user would retain the rights to play songs A, B and C.

We intend to redesign the OMA Choice Algorithm so that problems like the ones presented in (Barth, Mitchell, 2006) do not occur. The redesign method will include Falsification techniques (Ogata, Nakano, Kong, Futatsugi, 2006) for CafeOBJ together with the OTS/CafeOBJ method.

REFERENCES

Iannella, R., 2002. Open Digital Rights Language (ODRL) version 1.1. Available at: *http://odrl.net/1.1/ODRL-11.pdf* .

ContentGuard, 2007. XrML 2.0 Technical Overview version 1.0. Available at: *http://www.xrml.org/ Reference/ XrMLTechnicalOverviewV1.pdf*

Rightscom, 2007. The MPEG-21 Rights Expression Language - A Whitepaper. Available at: *http://www.xrml.org/reference/MPEG21_REL_ whitepaper _ Rightscom.pdf*".

Diaconescu, R, Futatsugi, K., 1998. CafeOBJ Report. *World Scientific*.

Open Mobile Alliance, 2006. OMA-TS-DRM-REL-V2_ 0-020060303-A. Available at: *http://www. openmobilealliance.org.*

CafeOBJ home page, 2009, *http://www.ldl.jaist.ac.jp/ cafeobj/.*

Ouranos, I., Stefaneas, P., Frangos, P., 2007. An Algebraic Framework for Modeling of Mobile Systems, In: *IEICE Trans. Fund.,* Vol. E90-A, No. 9, pp. 1986-1999.

Ouranos, I., Stefaneas, P., 2007. Verifying Security Protocols for Sensor Networks using Algebraic Specification Techniques. In: *Proc. CAI 2007, Thessalonica, Greece, May 2007,* LNCS 4728, pp. 247-259, Springer.

Barth, A., Mitchell, J.C., 2006. Managing Digital Rights using Linear Logic. In: *21th IEEE Symposium on Logic in Computer Science (LICS*), pp. 127-136.

Futatsugi, K., Goguen, J.A., Ogata, K., 2005. Verifying Specifications with Proof Scores in CafeOBJ. In: *B. Meyer, J. Woodcock (Eds.), Verified Software: Theories, Tools, Experiments, First IFIP TC 2/WG 2.3 Conference,* VSTTE, LNCS 4171, pp. 277-290.

Futatsugi, K., Ogata, K., 2008. Simulation-based Verification for Invariant Properties in the OTS/CafeOBJ Method. In: *Electronic Notes Theor. Comp. Science 201,* pp. 127-154.

Futatsugi, K., Ogata, K., 2006. Some Tips on Writing Proof Scores in the OTS/CafeOBJ Method. In: *K. Futatsugi, J.-P. Jouannaud, J. Meseguer (Eds.), Algebra, Meaning, and Computation, Essays Dedicated to Joseph A. Goguen on the Occasion of His 65th Birthday, LNCS 4060,* pp. 596-615, Springer.

Futatsugi, K., Ogata, K., 2003. Proof Scores in the OTS/CafeOBJ Method. In: *Proc. of the 6th IFIP WG6.1 Intl. Conf. on Formal Methods for Open Object-Based Distributed Systems,* LNCS 2884, pp.170--184, Springer .

Diaconescu, R. 2000. Behavioral Coherence in Object - Oriented Algebraic Specification. *J. Universal Computer Science. 6(1*), pp. 74—96

Ogata, K., Nakano, M., Kong, W., and Futatsugi, K., 2006. Induction-Guided Falsification. *Formal Methods and Software Engineering,* LNCS 4260, pp. 114-131, Springer.

Triantafyllou, N., Ouranos, I., Stefaneas, P., 2009. Algebraic Specifications for OMA REL Licenses. In Proc: *IEEE International Conference on Wireless and Mobile Computing, Networking and Communications.* wimob, pp.376-381.

Xiang, J., Bjørner, D., Futatsugi, K., 2008 Formal digital license language with OTS/CafeOBJ, method. *IEEE/ACS International Conference on Computer Systems and Applications 2008,* pp. 652 – 660.