Redesigning OMA Choice Algorithm

Nikolaos Triantafyllou¹, Petros Stefaneas², Iakovos Ouranos³, Katerina Ksystra¹ and Panayiotis Frangos¹

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

³ Hellenic Civil Aviation Authority, Heraklion International Airport "N.KAZANTZAKIS" 71601 Heraklion Crete, Greece
iouranos@central.ntua.gr

Abstract. We claim that the Open Mobile Alliance (OMA) Order of Rights Object Evaluation algorithm causes the loss of execution permissions under certain circumstances. By introducing an algebraic characterization as well as an ordering on licenses, we redesign this algorithm to minimize possible errors, in a way suitable for the limited computational powers of mobile devices. Our algorithm does not suggest major changes of the existing system, hence its future implementation will have minimal implementation cost.

Keywords: Mobile DRM, Algorithm Redesign, OMA, Order of Rights Object Evaluation

1 Introduction

Open Mobile Alliance (OMA) is an organization responsible for the definition of standards for the Mobile DRM systems [1]. The proposed standards for the DRM systems are called OMA-DRM [2], while the language, in which licenses are written, is called OMA-REL [3]. Like most such languages [4, 5, 6] OMA-REL is XML based and is defined as a mobile profile of ODRL [2]. OMA’s DRM is currently implemented in most mobile and smart phones and is adopted by most vendors for mobile content. We demonstrate that the OMA License Choice Algorithm currently in use suffers from a loss of execution permissions and we present a new and relatively fast algorithm to overcome it.

Barth and Mitchel [7] characterize the loss of rights by introducing a notion of monotonicity in licenses. A license is said to be monotonic, if a sequence of actions allowed under one license is also allowed under a more flexible one. They show that all algorithms that assign actions to rights, as actions occur, are non-monotonic in the context of OMA. They claim that the problem of determining whether a sequence of actions complies with a license is an NP-complete problem. Monotonicity can be achieved by enriching the semantics of licenses with linear negation, so that agents
can revise their past allocation of actions to rights after learning which other actions
the consumers wish to perform.

Our paper is organized as follows: section 2 presents the algorithm and the loss of
execution rights, section 3 gives a brief introduction to order sorted algebra, section 4
presents our new algorithm and some case studies and ends with conclusions and
future goals.

2 The OMA Choice Algorithm

2.1 The OMA Choice Algorithm

The OMA Choice algorithm deals with multiple licenses that refer to the same
content. A license is to be preferred if it contains a constraint of higher ranking in this
order (ordering among licenses). It is described as follows [3]:

- Only the rights that are valid at the given time should be taken into account
  from the algorithm.
- Rights that are not constrained should always be preferred over constrained
  rights.
- 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.
- If there exist more than one rights with a Datetime constraint, the one with
  the further in the future End element should be preferred.
- 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.
- Rights that contain a Count constraint should be preferred after rights that
  contain a Timed- Count constraint.
- This algorithm is indeed suitable for the mobile devices since it requires
  limited computing power.

2.2 Losing Rights

There exist some cases that the user may end up losing rights. Indeed, let us consider
the following two licenses installed in the device of a user.

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.”

These licenses translated in to OMA- REL contain the following constraints:
License 1 contains a Datetime constraint (“before the end of the month”), while
License 2 contains a count constraint (“ten times”). If the user decides to use his right
“listen song A”, the OMA-DRM agent will choose License 1 as the most suitable
license since it contains a Datetime constraint that is ranked higher in the ordering.
But by doing so, License 1 will become depleted since it also contains the count constraint denoted by “once”. This results in the user losing the right whenever listens 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 small design bug could cause major discomfort to the users.

3 Order Sorted Algebra in a Nutshell

An order sorted algebra is a partial ordering ≤ on a set of sorts [11], where by sorts we usually mean a set of names for data types. This sortset 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 ≤ s’ then A_s ⊆ A_s’, where A_s denotes the elements of sort s in A. Order sorted algebra (OSA) provides [11] 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 [11].

Formally, given a partially ordered sort set S, an S-sorted set A is just a family of sets A_s for each sort s ∈ S. A many-sorted signature is a pair (S, Σ) where S is called the sort set and Σ is an S’ × S-sorted family of functions {Σ_w,s | w ∈ S’ and s ∈ S}. An order sorted signature is a triple (S, ≤, Σ) such that (S, ≤) is a many-sorted signature, (S, ≤) is a poset, and the operations satisfy the following monotonicity condition: σ ∈ Σ_{w1,s1} ∩ Σ_{w2,s2} and w1 ≤ w2 imply s1 ≤ s2. Given a many-sorted signature an (S, Σ)-algebra A is a family of sets {A_s | s ∈ S} called the carries of A, together with a function A_σ : A_w → A_s for each σ in Σ_{w,s}, where A_w = A_{s1} × ... × A_{sn}, w=s1...sn. Let (S, ≤, Σ) be an order sorted signature. An (S, Σ)-algebra A such that, s ≤ s’ in S implies A_s ⊆ A_s’ and σ ∈ Σ_{w1,s1} ∩ Σ_{w2,s2} and w1 ≤ w2 imply A_σ : A_{w1} → A_{s1} equals A_σ : A_{w2} → A_{s2} on A_{w1}.

4 Redesigning the Algorithm

4.1 Licenses in OMA REL/DRM

Licenses in Open Mobile Alliance Rights Expression Language (OMA REL) resemble in their structure a forest of trees. More precisely, several sublicenses constitute a license. The structure of a sublicense can be seen in figure 1. The idea behind this structure is that a set of permissions is allowed if its “local” set of constraints is satisfied and the sublicense is valid. A sublicense is supposed to be valid if the set of “top” constraints is valid.

Based on the ideas of order sorted algebra presented in section 3, we can argue that licenses are basically a data type. Consecutively there exist a sort set S, of sort names that can be used to represent these licenses. In addition, based on the rights object evaluation provided by OMA there exists a predefined ordering on these sort names.
The key observation that leads to the redesign of the algorithm is that there exist some special cases that loss of rights can occur. Given a set of licenses, we can lose some rights when:

"A license contains more than one permission elements and after the execution it becomes depleted"

According to the statement above, we can argue that each license should be characterized under the following observations:

a) The License becomes depleted after the execution of a right
b) The License contains more than one permission elements
c) The characterizing constraint based on the OMA constraint ordering

In order for observation (a) to occur, the license needs to contain either a count constraint (with one execution still allowed), or a timed count constraint (with once again one execution left). Observation (b) can be easily checked at the level of the creation of a license. Finally, observation (c) can be achieved by a simple search on the license constraints.

Following again the approach of order sorted algebra we can define a signature for OMA REL licenses to be \( S \times S \times S \times \Sigma \) with \( \Sigma = \emptyset \), \( S_1 = \{ \text{count, date, true, …} \} \) the names of the various constraints allowed by OMA REL, \( S_1 = \{ \text{once, many} \} \) denoting whether the license will allow more than one execution of its permissions and finally \( S_2 = \{ \text{complex, simple} \} \), denoting if the license contains more than one permissions.

So essentially a license in this approach is an element of the carrier set \( A_s \) for each sort \( s \) where \( s \in S_1 \times S_2 \times S_3 \). For example \( \text{once} \times \text{simple} \times \text{count} \), denotes that the license allows only one more execution of a permission, it contains only one permission and the dominant constraint of the license is a count constraint. The ordering comes from the predefined ordering of the rights object evaluation algorithm in conjunction to the fact that \( \text{once} \leq \text{many} \) and \( \text{simple} \leq \text{complex} \). So, formally we have that \( s_1 \times s_2 \times s_3 \leq s_1' \times s_2' \times s_3' \) implies that \( s_1 < s_1' \) or \( s_1 = s_1' \).
and $s_2 < s'_2$ or $(s_2 = s'_2$ and $s_3 \leq s'_3$). Using this ordering on licenses we will present in the next section a novel algorithm for the decision problem of the optimal license.

Our approach does not require any knowledge on behalf of the agent on the future actions of the users. It uses the existing OMA algorithm in its core and its computational weight is moved to the creation of the licenses. Once this is done, the algorithm we propose is no more than a linear search on the licenses and an application of the initial OMA algorithm. We believe that our algorithm is relatively easy to implement since it does not require the reconstruction of all the DRM agents.

4.2 Labeling Licenses

As we can see from the characterizations via sorts mentioned in section 4.1, we could augment a license to contain those sort names by using some labels that will be added to the sublicense as a universal label and to the Constraint Permission Set as a local label. This should be done on the creation level of the license and will denote the existence or not of the above special circumstances. More precisely, we want each of these labels to denote when it comes to a Constraint Permission Set or a sublicense:

a) If it refers to one or more permission elements/Constraint Permission Sets; we define that the label will contain the designated word “Simple” or “Complex” respectively.

b) The optimal constraint Name (based on the OMA ordering), we define that the label will contain the name of the Constraint, e.g. “Count”, “Datetime”, “True” (for unconstraint) and so on.

c) Whether it becomes depleted after the next execution of one of its rights. Here we define that the label will contain the words “Once” or “Many” respectively.

As an example, such a label for a Constraint Permission Set could be “SimpleManyTrue”. We need to note here that these labels need to be created simultaneously with the License itself so as to reduce the computational weight on the mobile device.

Also for the algorithm we propose we assume that the DRM agent is enhanced so as to be able to update these license labels after the execution of permissions as necessary. Meaning that if a license after the execution of permission allows only one more execution its label should be updated to say “SimpleOnceTrue” from “SimpleManyTrue”

4.3 The proposed new Algorithm

Now that we have defined the labels for the Licenses/Constraint Permission Sets we can define the proposed algorithm. The underlying intuition is as follows:

“check the licenses installed on the mobile DRM device for the ones matching the request of the user. See if any of these licenses falls into the special case. If all the matching licenses fit into that category use the OMA Algorithm. If there exists a set of licenses that do not fall on this special category use the OMA Algorithm on them instead of the entire matching license set. Finally update the labels if necessary”.
The algorithm can be seen in terms of pseudo code in the following table.

### Table 1. The algorithm in terms of pseudo code

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Input: action, content, SetofLicenses[]</td>
</tr>
<tr>
<td>2.</td>
<td>i = 0; j = 0;</td>
</tr>
<tr>
<td>3.</td>
<td>While( i&lt;= length.SetofLicense) do</td>
</tr>
<tr>
<td>4.</td>
<td>{ If (action, content) belongs SetofLicenses[i]</td>
</tr>
<tr>
<td>5.</td>
<td>{ referencelic(j) = SetofLincenses[i]; j++; }</td>
</tr>
<tr>
<td>6.</td>
<td>i++; }</td>
</tr>
<tr>
<td>7.</td>
<td>i = 0; j = 0;</td>
</tr>
<tr>
<td>8.</td>
<td>While i&lt;= referencelic.length do</td>
</tr>
<tr>
<td>9.</td>
<td>{ If (lictype.referencelic[i] != once) or ( licotype.referencelic[i] = simple)</td>
</tr>
<tr>
<td>10.</td>
<td>{ posslic[j] = referencelic[i]; j++; }</td>
</tr>
<tr>
<td>11.</td>
<td>i++; }</td>
</tr>
<tr>
<td>12.</td>
<td>i = 0; j = 0;</td>
</tr>
<tr>
<td>13.</td>
<td>While i&lt;= posslic.length do</td>
</tr>
<tr>
<td>14.</td>
<td>{ if (type.findperm(posslic[i]) != once) or (type.findperm(posslic[i]) == simple)</td>
</tr>
<tr>
<td>15.</td>
<td>{ finallic[j] = posslic[i]; j++; }</td>
</tr>
<tr>
<td>16.</td>
<td>i++; }</td>
</tr>
<tr>
<td>17.</td>
<td>If isempty.posslic {return OMA(referencelic);</td>
</tr>
<tr>
<td>18.</td>
<td>else { return OMA(finallic); }</td>
</tr>
</tbody>
</table>

In line 1 we define as the input for the algorithm, the action on a content the user wants to exercise and the set of licenses that are installed on his mobile DRM device. Lines 2 – 6 define a simple search on the installed licenses to find those referring to this pair of action, content. Finally these matching licenses are stored in an array called “referencelic”. Lines 7-11 define a search on the matching licenses for licenses that do not fall on the special case and if any exists, stores them on the “posslic” array. Now, if the “posslic” array is not null then it means that the special case does not appear on the level of the license. Despite we need to check if it appears on the level of the Constraint Permission Set. This is done with a further refinement of the licenses in lines 12-16. Finally, in lines 17 to 18 we apply the OMA Choice Algorithm on the appropriate set of licenses according to whether or not all the matching licenses fall into the special case and return the result as the output of the algorithm.

#### 4.4 Case Studies

We have implemented the above algorithm in Java. We were able to do several case studies so as to check that no loss of rights occurs. In Table 2 we can see the license sets checked. As we can see in the Table 2 the two algorithms agree on licenses where the special cases do not occur while the proposed algorithm correctly chooses the license that will not cause any loss of permissions in the case of these special circumstances.
<table>
<thead>
<tr>
<th>License Set</th>
<th>User request</th>
<th>Proposed Algorithm</th>
<th>OMA Choice Algorithm</th>
</tr>
</thead>
</table>
| License 1: “play songs A or B once before the end of the month”  
License 2: “play songs A or C ten times”       | Play song A                                       | License 2          | License 1            |
| License 1: “for a total of ten times, display content 1 or until the end of the month play content2”  
License 2: “until the end of the month play content 2 three times or display content 1” | Display content 1                                 | License 2          | License 2            |
| License 1: “for a total of ten times, display content 1 or until the end of the month play content2”  
License 2: “until the end of the month play content 2 three times or display content 1” | Play content 2                                    | License 2          | License 2            |
| License 1: “for a total of ten times, display content 1 or play content2”  
License 2: “play content 2 three times or display content 1”  
License 3: “play content 2 or 3 once before the end of the month” | Play content 2                                    | License 1          | License 3            |

5. Conclusions and Work in Progress

We have redesigned the OMA License Choice Algorithm by simply introducing labels on some parts of the licenses. This can be easily done by tweaking the way licenses are created. We claim that this algorithm to be fairly fast thus allows us to use it in as many mobile devices as possible. The extra computational cost has been shifted to the creation of licenses. We do not believe that this will cause any problems since these licenses are written for computers with much higher computational power. We also believe that it will be easier to create automated tools that take as input a license written in OMA REL and give as output licenses containing these labels. In addition, we have produced some case studies through an implementation of the algorithm in Java that we believe demonstrates the benefits of this algorithm over the existing one, as well as provide hints that it behaves in a correct manner. We have created a formal specification of the algorithm we propose and provide a formal proof that in fact no loss of permissions can occur. This is in the spirit of our previous work in the formal verification of mobile systems and algorithms [8, 9 and 10].
References

5. ContentGuard, Xrml 2.0 technical overview version 1.0., 2007.