1 INTRODUCTION

Sharing data among individuals and organizations is becoming easier and easier with the support of highly-connected ICT systems. Data sharing, however, poses several problems, including privacy and data misuse issues, as well as uncontrolled propagation of data. Decades of research have shown how a technical approach based on the definition and enforcement of data sharing policies represents a valid support in automatising, easing, and assuring the process of electronic data sharing (Ferraiolo and Kuhn, 1992; Park and Sandhu, 2004; Damiani et al., 2001; Casassa Mont et al., 2015). This work considers Data Sharing Agreements (DSA), electronic contracts specifying rules for data sharing among the contracting parties and - possibly, third parties. In the real world, several actors, working in different fields and with different expertise, may contribute to DSA definition: a legal expert, familiar with the legal and contractual perspectives of the agreement, could set up basic aspects, like the kind of data whose sharing is being regulated, the purpose of such sharing, and, e.g., the rules specific for international data transfer, as defined by national and international regulations; a business expert at a specific organization - such as a private company, a public administration, a healthcare department - could further refine the agreement, by inserting rules that are peculiar for that organization (e.g., access rights within different units of the same company). Furthermore, for some specific scenarios, like the healthcare one, an end-user (a patient) could both acknowledge the rules regulating her personal data and add some rules, e.g., for delegation matters.

Different actors editing the agreement and the multiplicity of data sharing rules envisaged in medium to large contexts may cause the presence of potential conflicts among the rules. Indeed, at time of requesting the access to (or the usage of) the data whose sharing is regulated by the DSA, an enforcement engine will evaluate the rules in the agreement. The evaluation could result in two - or more - rules, all applicable to that access request, which give different effects: according to some rules, the access/usage would be granted; according to some others, the access/usage would be denied. Here, we propose an analysis tool, the DSA-Analyser, which considers the rules in a DSA and spots potential conflicts before the actual enforcement of the agreement. Conveniently guiding the editor to the kind of conflicts and the rea-
sons which may cause them, the DSA-Analyser is available as a web service application, it exposes its functionalities through APIs, and performs the actual analysis of the rules by means of Maude, a formal engine based on rewriting logic. This paper describes the design and implementation of the DSA-Analyser, also evaluating its performances in terms of execution time, varying the number of rules in the DSA and the number of terms in each rule. For the majority of our experiments, the performances results highlight that up to hundreds of rules are analysed in less than or around half a minute. Quite obviously, such an outcome is appealing, because the automatic analysis of hundreds of rules in a reasonable amount of time outperforms the capability to manually investigate the same set of rules to find conflicts among them. Furthermore, such results improve those of past conflict detection approaches, like the one in (Matteucci et al., 2012a). As illustrated in the following, DSA regulate data sharing in a quite channelled way: each DSA is related to a specific category of data, and tied to a specific purpose of use, thus limiting the number of significant rules that a DSA may include. From our practical experience of real case scenarios inherited from public administration, healthcare, and even large private companies, we argue that the DSA-Analyser can be employed in real-use contexts to check consistency and spot potential conflicts among the exposed data sharing rules.

The paper is structured as follows: Next section presents background notions on DSA and the kind of conflicts we deal with. Section 3 describes the DSA-Analyser and gives performance results. Section 4 highlights how to proceed towards rules enforcement, once the DSA has been analysed. Section 5 discusses related work and Section 6 concludes the paper.

2 BACKGROUND

Hereafter, we recall definition and description of Data Sharing Agreements and report a definition of the type of conflicts we consider in the current work.

2.1 Data Sharing Agreements

DSA are electronic documents regulating how parties share data. Resembling their cousin legal contracts, they consist of:

- the DSA title, a label which could be used to identify the DSA (DSA_ID).
- the parties involved in the DSA. Parties can be either natural or legal persons, and they are specified by means of their names, roles and responsibilities. Borrowing the language from the privacy and data protection context, as roles of the parties a DSA usually involves the Data Controller, the Data Processor, and the Data Subject\textsuperscript{1}. Responsibilities are legal duties of the parties expressed in pure natural language, in terms of gathering, sharing, and storing the data subject of the agreement.
- the validity period of the DSA, stating its start and end date.
- the vocabulary, which provides the terminology for authoring the DSA data sharing rules. The vocabulary is defined by an ontology, i.e., a formal explicit description of a domain of interest (like, for example, a medical or a public administration domain).
- the data classification, describing the nature of the data covered by the DSA, such as personal data (e.g., contact details, medical data, judicial data) and non-personal data (e.g., business data, as corporate strategy development analysis, customer data, product development plans).
- the purpose of the DSA, which is linked with the data classification. Example of purposes are the provision of healthcare services (e.g., for diagnoses), administrative purposes (e.g., for book and payments), marketing (e.g., for proposal of commercials services), and fulfillment of law obligations (e.g., to access data when needed by public authorities).

Finally, a DSA contains the rules regulating data sharing:
- the authorizations section contains rules on permitted operations;
- the prohibitions section contains rules on operations which are not allowed;
- the obligations section contains rules which are mandatory, in relation to the data sharing.

2.2 Conflicts

Intuitively, when editing a set of data sharing rules, conflicts can arise. In particular, in this work, we consider conflicts between authorizations and prohibitions, and between obligations and prohibitions. At time of DSA writing, different rules can be inserted, even by different authors. Let the reader consider,

\textsuperscript{1}Terminology adopted in the European Parliament Directive 95/46/EC and in the new General Data Protection Regulation (which will start to apply in 2018) to indicate the parties involved in an agreement governing the sharing of personal data.
for example, an expert legal user composing legal authorization rules that strictly descend from legislation and regulations. Then, a policy expert at a particular organization may want to add to the DSA specific data sharing rules, which apply to the organization itself. Two, or more, rules composing the DSA could allow and deny the data access under the same contextual conditions, which are a collection of attributes referred to subject, object, and environment describing the conditions under which an action is authorized, prohibited, or obliged. To give the flavour of what a contextual condition is, let the reader suppose that the DSA contains the following simple authorization rule: *Doctors can read radiological reports during office hours.* When a subject tries to access a data, according to that authorization the access will be granted if the following contextual conditions hold: the subject is a doctor, the data is a radiological report, and the time at which the access request is being made is within office hours.

Throughout the paper, we manage the following three sets of conflicting rules - the nomenclature has been inherited from (Jin et al., 2011):

**Contradictions.** Two rules are contradictory if one allows and the other denies the right to perform the same action by the same subject on the same data under the same contextual conditions. In practice, the rules are exactly the same, except for their effect (deny/permit the access).

**Exceptions.** One rule is an exception of another one, if they have different effects (one permits and the other denies) on the same action, but the subject (and/or the data, and/or the contextual conditions) of one rule belongs to a subset of the subject (and/or the data, and/or the conditions) of the other one. As an example: an authorization stating *doctors can read medical data* and a prohibition stating *doctors cannot read radiological reports,* where radiological reports are a subset of medical data.

**Correlations.** Two rules are correlated if they have different effects and the set of conditions of the two rules intersect one with the other. As an example, *doctors can read medical data* and *who is outside the hospital cannot read medical data:* these raise a conflict when a doctor tries to access when she is not inside the hospital.

## 3 DSA-ANALYSER

In our scenario, we imagine a policy expert at an organization (such as a hospital, a public administration, or a private company): she aims at analysing rules in a DSA, and rules have been possibly composed by different actors - like the policy expert herself and a legal expert - who knows legal constraints applicable to the data whose sharing is controlled by that DSA. The DSA-Analyser has been developed using RESTful technology\(^2\). This allows the tool to be reachable through a simple HTTP call, while the execution of the core component can be expressed using a different programming language, such us Java. This way, the interaction with the analyser is quite versatile since it can be directly done from a generic web browser as well as a client software developed to interact with the tool. We develop the core of the DSA-Analyser, as well as its functionalities, using Java v8.

Then, the DSA-Analyser runs as web-application into an Apache Tomcat v7.0.70 server. To call the DSA-Analyser, a simple web-client application specifies the server URI.

The client specifies the call type - for the DSA-Analyser is *POST* - and the DSA ID, which is sent as payload in the call.

The inner analysis process is hidden to the user and it is performed by Maude (Clavel et al., 2007)\(^3\), a popular executable programming language that models distributed systems and the actions within those systems. We let Maude group data sharing rules in authorizations, prohibitions, and obligations. Each set of rules is seen as a process, describing the sequence of authorised, denied, and obliged actions, respectively. Maude is executable and comes with built-in commands allowing, e.g., to search for allowed sequence of actions within a set of rules. We simulate all the access requests which are possible given the application domain of the rules (in Section 3.1 we will detail how the access requests are built). When Maude finds at least two sequences, in different sets, which are equal (e.g., subject with role doctor reads object of type radiological reports). When Maude finishes the computation, the analysis outcome is shown to the user through a graphical interface.

The DSA-Analyser takes as input a DSA ID from an external database, with the available DSAs for a specific organization (Fig. 1). Then, it grabs the DSA through its ID and it starts processing it. As shown in Code 1, the DSA content follows an XML structure. The analyser executes the following steps to check conflicts:


\(^{3}\)maude.cs.illinois.edu
Code 1. Source code of a DSA.

<?xml version="1.0" encoding="UTF−8"?><dsa xmlns="..." encryption−key−schema="organization_default_scheme" governing−law="General Data Protection Regulation (GDPR)" id="DSA−8e9126eb.xml" purpose="Provision of Healthcare Services" status="Customised" title ="Example DSA" user−consent="false" version="1.0" vocabulary−url="http:// testcocodsa . iit . cnr . it:8080 / vocabularies / healthcare_vocabulary .owl#" >
<description />
<expirationPolicy periodInDays="0"><complexPolicy>DenyAll</complexPolicy></expirationPolicy>
<revocationPolicy periodInDays="0"><complexPolicy>DenyAll</complexPolicy></revocationPolicy>
<updatePolicy periodInDays="12"><complexPolicy>DenyAll</complexPolicy></updatePolicy>
<parties>
<organization id="Hospital" name="Hospital Name" responsibilities="..." role="data−controller" />
<organization id="Hospital−Partner" name="Partner Name" responsibilities="..." role="data−controller" />
</parties>
<validity>
<startDate>2016−10−27+02:00</startDate>
<endDate>2016−12−31+01:00</endDate>
</validity>
<data>
<datum id="DATUM_X_2">
<expression language="CNL4DSA">?X_2 is a DelegateOfPatient</expression>
</datum>
<datum id="DATUM_X_3">
<expression language="CNL4DSA">?X_3 is a Medical</expression>
</datum>
<datum id="DATUM_X_5">
<expression language="CNL4DSA">?X_5 is a Patient</expression>
</datum>
...
</data>
<authorizations>
<authorization id="AUTHORIZATION_1">
<expression issuer="Legal Expert" language="NaturalLanguage">DelegateOfPatient CAN Read a Data</expression>
<expression issuer="Legal Expert" language="CNL4DSA">if (hascategory(?X_18,?X_3)) and (hasrole(?X_21,?X_14)) and (haspurpose(?X_18,healthcare)) then { can [?X_21, Read, ?X_18] }</expression>
</authorization>
</authorizations>
<obligations>
<obligation id="OBLIGATION_1">
<expression issuer="Legal Expert" language="NaturalLanguage">AFTER DelegateOfPatient Access a Data THEN the System MUST Log the Event</expression>
</obligation>
</obligations>
<dataClassification >MEDICAL DATA</dataClassification>
</dsa>
Step 1 it checks that the XML file is well formed and it can be properly parsed.

Step 2 it reads the root elements of the DSA, such as the purpose, id, parties, roles, title (Code-1 line 2).

Step 3 it reads all DATUM fields written in the document (Code-1 line 15-23).

Step 4 it converts the rules written in CNL4DSA to the input language of Maude (Code-1 line 27-42).

Step 5 it loads all Terms and Property from the Vocabulary associated to the DSA.

Step 6 it generates the contextual conditions used to evaluate the rules.

Step 7 through Maude, it evaluates each pair in the DSA, consisting of one authorization and one prohibition, and one obligation and one prohibition.

Step 8 it reports the outcome of the analysis.

Code 2. Example of DATUM.

Expression : ?X_18 is-a
  <http://localhost:8080/vocabularies/healthcare_vocabulary.owl#Data>
language : CNL4DSA
Entity : Data
URL : http://localhost:8080/vocabularies/healthcare_vocabulary.owl
idShort : ?X_18

Rules expressed in the DSA are specified both in natural language and in CNL4DSA, a controlled natural language firstly introduced in (Matteucci et al., 2010). In Code 1 (line-31) is shown an example of a CNL4DSA authorization rule. The terms specified in the data section shown in Code 1 will replace the place-holders (?X_i). As an example, ?X_18 refers to the term #Data (see Code 2), while ?X_3 refers to the term #Medical. The first part of Code 1 (line-31) is a condition regarding a datum, which has category medical.

Code 3. Example of rule expressed in CNL4DSA.

<expression issuer="Legal Expert"
  language="CNL4DSA">if (hascategory(data,medical)) and (hasrole(?X_21,?X_14))
  and (haspurpose((data,healthcare)))
then { can [?X_21, read, data] } </expression>

To replace all the place-holders with their corresponding values, the DSA-Analyser parses and stores all the data specified in the data section of the XML DSA file, see, for example, Code 2. Upon replacing each place-holder in the DSA rules, the DSA-Analyser translates the CNL4DSA version of the rules into the Maude syntax, according to the translation process described in (Matteucci et al., 2012b). Maude is the actual tool performing the analysis of DSA rules. For instance, from Code 1 (line-31), we obtain Code 4.

Code 4. A rule in MAUDE.

((hascategory(data,medical)) and
  (hasrole(user,doctor)) and
  (haspurpose(data,healthcare)) ) < user1, 'read, data > . 0

All rules written in Maude are part of a bigger template that is given as input to Maude to evaluate the rules. The DSA-Analyser uses a single template, filled in with the rules parsed from the DSA. The STATMENTS_HERE placeholder in Code 5 is the part in the template where the DSA-Analyser inserts the DSA rules, converted as in previous steps.

Code 5. Template excerpt.

... mod EXAMPLE is
  inc CNL4DSA
  eq dsa-auth = STATMENTS_HERE
  endm

STATMENTS_HERE is updated as in Code 6 (for the sake of simplicity, we have shown only one authorization rule).
3.1 Conflict Detection Algorithm

The data sharing rules are evaluated in Maude under a set of contextual conditions (hereafter called contexts), which mimics valid properties at time of the access request. Thus, contexts instantiate properties of the subject, the data, and the external environment: for a generic access request, when a subject will request the access to some data, such properties will be checked against the DSA rules, to evaluate the right of that subject to access those data. Contexts may specify, e.g., the role of the subject making the request, the category of requested data, the time of the day at which the request is being made, the geographical location (both of data and subjects), and so on. An example of context is in Code 7.

A peculiar feature of the DSA-Analyzer is its ability to simulate all the possible contexts, given the DSA vocabulary and the properties specified in the DSA rules. The algorithm for contexts generation is detailed in the following.

3.1.1 Algorithm for Context Generation

A DSA vocabulary is made up of properties $p$ and terms $t$. A single DSA document may contain a subset of the properties. Examples of properties are hasRole, hasCategory, hasID. Properties have a domain and a codomain. Examples of domains are Subject and Data. Examples of codomains are Doctor (e.g., for property hasRole), Medical (e.g., for property hasCategory).

To automatise the process of generating all the possible combinations of properties, ranged over all the different terms, we first consider the array $P_{[p_q]}$, $0 \leq q \leq n$, shown below. The array $P$ lists all the properties specified in the rules that appear in a DSA.
\[
P[0] \rightarrow T_0[t_0^0,t_0^1,t_0^2,\ldots,t_0^n]
\]
\[
P[1] \rightarrow T_1[t_1^1,t_1^2,\ldots,t_1^n]
\]
\[
\ldots
\]
\[
P[n] \rightarrow T_n[t_n^0,t_n^1,\ldots,t_n^n]
\]

For each position of the array $P$, another array $T_q$ contains the list of terms representing the codomain for the specific property $p_q$. The DSA-Analyzer grabs from the DSA vocabulary the properties and associated terms to form $P$ and $T$, by filtering out those properties that do not belong to the rules of the specific DSA under investigation. Instantiating the above structure with an example, we have:

\[
[\text{hasRole}] \rightarrow \{\text{Role}_1,\text{Role}_2\}
\]
\[
[\text{hasDataCategory}] \rightarrow \{\text{Category}_1\}
\]
\[
[\text{hasID}] \rightarrow \{\text{id}_1,\text{id}_2\}
\]

We then create a matrix $M$ whose values are pointers to a property with an associated term. The number of rows in the matrix is equal to all the possible combinations of properties and terms:

\[M_{max} = |\text{hasRole}| * |\text{hasDataCategory}| * |\text{hasID}|
\]

The number of columns is equal to the number of properties: $M_{columns} = |\text{Properties}|$. In our example, we have $M_{rows} = 2 * 1 * 2 = 4$ and $M_{columns} = 3$.

We start filling the content of the matrix from the last column on the right. This last column will contain pointers to the last item of $P$, i.e., $P_{[p_n]}$ (and to the corresponding values in $T_n$). We initialise a counter, which starts at zero, and stops increasing at $|\text{hasID}|$ length – 1 (in the example, 2=1=1, thus, leading to only two possible values, 0 and 1). Once the counter reaches $|\text{hasID}|$ length – 1, it starts again until the numbers of rows are all filled in. Thus, we get a partially filled matrix, as follows:

\[
M = \begin{bmatrix}
X & X & 0 \\
X & X & 1 \\
X & X & 0 \\
X & X & 1 \\
\end{bmatrix}
\]

Then, the algorithm starts processing the second array from the bottom, $P_{[p_n-1]}$ (in our example, we consider the element [hasDataCategory]). The counter stops at $|\text{hasDataCategory}|$ length – 1 = 1 – 1 = 0. This means that, for all rows of the corresponding column in the matrix, we put 0:

\[
M = \begin{bmatrix}
X & 0 & 0 \\
X & 0 & 1 \\
X & 0 & 0 \\
X & 0 & 1 \\
\end{bmatrix}
\]
As last step in the example, we proceed with the element, \[hasRole\], which still range over two terms: even in this case, the counter can have two possible states, 0 and 1. Differently from the way we acted when filling the right-hand column, the algorithm fills the column by repeating a counter value for a number of times equal to:

\[
\text{hasID.length} \times \text{hasDataCategory.length} = (2 \times 1) = 2
\]

Concluding, the algorithm generates a combination matrix:

\[
M = \begin{bmatrix}
0 & 0 & 0 \\
0 & 0 & 1 \\
1 & 0 & 0 \\
1 & 0 & 1
\end{bmatrix}
\]

The DSA-Analyser uses the matrix and it works at row level for generating all the possible contexts. At the first iteration, the DSA-Analyser generates the context in Code 8:

**Code 8. Automated context generation.**

```plaintext
#1
Subject hasRole Role1 = true;
Data hasDataCategory Category1 = true;
Subject hasID id1 = true;

Then, at the second iteration, the context produced by the DSA-Analyser is:

```plaintext
#2
Subject hasRole Role1 = true;
Data hasDataCategory Category1 = true;
Subject hasID id1 = true;
```

Finally, third and forth iterations are:

```plaintext
#3
Subject hasRole Role2 = true;
Data hasDataCategory Category1 = true;
Subject hasID id2 = true;

#4
Subject hasRole Role2 = true;
Data hasDataCategory Category1 = true;
Subject hasID id2 = true;
```

Algorithm 1 shows the pseudo-code for the generation of the combination matrix \(M\).

### 3.2 Performances

Here, we evaluate the DSA-Analyser execution time, varying i) the number of rules in a DSA, and ii) the dimension of the DSA vocabulary. We test the tool on three real use cases: data sharing i) among different health organizations, ii) through mobile devices within a corporate environment, and iii) among different municipalities. Each scenario is associated to a vocabulary defined by the Web Ontology Language (OWL). An ontology is a formal way to describe taxonomies and classification networks, essentially defining the structure of knowledge for various domains: the nouns representing classes of objects (terms of the vocabulary) and the predicates representing relations between the objects (properties of the vocabulary).

The three vocabularies we considered in our tests have different dimensions in terms of both number of terms and number of properties. We performed a series of experiments on DSA containing a different number of rules (from 3 to 960 rules, see Table 1), for a total of 27 DSA (9 DSAs per 3 different vocabularies). The DSA-Analyser analyses each DSA by evaluating - separately - the authorization, the prohibition, and the obligation rules, with respect to all the possible generable contexts. The healthcare vocabulary (HV) leads to 83 iterations for set of rules, the public administration vocabulary (PAV) leads to 10 iterations, and the mobile vocabulary (MV) leads to 42 iterations. We remind the reader that the test data are practically relevant, since rules and vocabularies are the ones from real use cases. Furthermore, the DSA-Analyser execution time resulting from our experiments is independent from the number of conflicts actually occurring over the tested rules (meaning, the analysis could, e.g., reveal no conflicts, or even one conflict per each pair of rules, the execution time will be the same). Tests were run on a 1.3 GHz Intel Core m7 with 8 GB of RAM and SSD storage. Figures 2 and 3 report the execution time varying the number of rules in the DSA and the vocabulary. Total analysis refers to the whole analysis over the DSA, while single analysis considers the average execution time of the analysis of a DSA evaluated with respect to a single context. Overall, the Maude engine execution time is stable - and reasonably small - until it processes nearly one hundred rules. In particular, from the graphical representation in Figure 2, we observe that the execution time starts growing polynomially when the rules are greater than 120. This is particularly relevant when considering the healthcare scenario: the difference in this scenario is the number of

<table>
<thead>
<tr>
<th>Number of rules</th>
<th>Total Analysis Time (s)</th>
<th>Single Analysis Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>HV 44.5</td>
<td>445</td>
</tr>
<tr>
<td>10</td>
<td>HV 74.1</td>
<td>745</td>
</tr>
<tr>
<td>10</td>
<td>MV 194.5</td>
<td>1945</td>
</tr>
<tr>
<td>480</td>
<td>HV 959.8</td>
<td>9598</td>
</tr>
<tr>
<td>120</td>
<td>MV 420.3</td>
<td>4203</td>
</tr>
<tr>
<td>250</td>
<td>HV 250.2</td>
<td>2502</td>
</tr>
<tr>
<td>480</td>
<td>HV 959.8</td>
<td>9598</td>
</tr>
<tr>
<td>960</td>
<td>HV 959.8</td>
<td>9598</td>
</tr>
</tbody>
</table>
Algorithm 1: Combination Matrix Algorithm.

**Require:** An array with all Properties $P = [P_1, P_2, \ldots, P_N]$ and $n$ different Terms arrays where $T_1 = P_1, T_2 = P_2, \ldots, T_n = P_n$. Each $T$ is an array $T = [t_1, t_2, \ldots, t_K]$.

1: \textbf{global} $P$  
2: \textbf{function} COMBINATIONMATRIX($P$)  
3: \textbf{for} $i = 1$ to $\text{P.length}$ do  
4: \hspace{1em} $T = P[i]$  
5: \hspace{1em} if $T.length > 0$ then  
6: \hspace{2em} numRows = numRows * $T.length$  
\hspace{1em} end if  
\textbf{end for}  
7: \hspace{1em} $\triangleright$ Counting the number of rows that the matrix will have  
8: \hspace{1em} $\triangleright$ Initialization of variables  
9: \hspace{1em} numColumns = $\text{P.length}$  
10: \hspace{1em} $\triangleright$ Numbers of columns of the Matrix  
11: \hspace{1em} combinationMatrix = new [numRows, numColumns]  
\hspace{1em} $\triangleright$ Creating the combination Matrix  
12: \hspace{1em} previousArrayLength = 1  
\hspace{1em} $\triangleright$ It contains the length of the Terms array before that one it is processed  
13: \hspace{1em} iterationNumber = 0  
\hspace{1em} $\triangleright$ It counts how many loops will be done  
14: \hspace{1em} innerIterationNumber = 0  
\hspace{1em} $\triangleright$ it counts how many loops will be done in the inner \textbf{FOR}  
15: \hspace{1em} arraySize = 0  
\hspace{1em} $\triangleright$ The size of the array processed  
16: \hspace{1em} \textbf{for} $i = (\text{numColumns} - 1)$ to 0 do  
17: \hspace{2em} \textbf{if} iterationNumber == 0 then  
18: \hspace{3em} $\triangleright$ $P[i + 1]$ because it is a decrementing \textbf{FOR}  
19: \hspace{3em} previousArrayLength = previousArrayLength * $T.length$  
\hspace{2em} \textbf{end if}  
20: \hspace{1em} else  
21: \hspace{2em} $T = P[i + 1]$  
22: \hspace{2em} previousArrayLength = previousArrayLength * $T.length$  
\hspace{1em} \textbf{end if}  
23: \hspace{1em} \textbf{end for}  
24: \hspace{1em} $\triangleright$ Initializing everything for next loop  
25: \hspace{1em} iterationNumber++  
26: \hspace{1em} innerIterationNumber = 0  
\hspace{1em} \textbf{end function}  

However, the polynomial growth in terms of rules number does not sensitively affect those scenarios where the number of terms in the vocabularies are
The cost of the second function depends on the Maude engine. To evaluate the rules of a DSA, we exploit the Maude built-in command \texttt{red}; the tool performs as many \texttt{red} calls as the number of rules of the DSA. Hence, the computational costs of this functions depends on the number of rules, hereafter denoted by \( n \), and by the computational costs of the \texttt{red} function in Maude.

The computational cost of the third function is of \( O(n^2) \) order, being the pairwise comparison of the evaluation of the rules.

Thus, the DSA-Analyser complexity is strictly related to the complexity of the Maude engine (second function), plus an additional factor that depends on the third function, \( (O(n^2)) \), all multiplied by the number of iterations of the first function \( (O(num\_prop^num\_terms)) \).

4 NOTES ON RULES PRIORITIZATION

The DSA-Analyser outputs either the confirmation that no conflicts arise among the evaluated rules or the complete list of conflicts, each of them associated to the related context. In Figure 4, the alert message highlights two potential conflicts, one between authorization \#1 and prohibition \#1, and one between the same prohibition and authorization \#2. Focusing on the former alert, it says that, at DSA enforcement time, there could be an access request that should be authorized (according to authorization \#1) and should be denied (according to prohibition \#1). The access request is evaluated under the context specified in the figure (i.e., the subject is a patient, s(he) is located in a certain area, the data are medical ones, of type ECG, they are stored within the European area, and so on).

To fix a conflictual situation, it is possible to re-editing the rules which may lead to conflicts, and re-running the analysis phase once again. However, it is also possible to leave the rules as they are, and to define ad hoc resolution strategies that will act at enforcement time. Indeed, well-known standard policy languages, such as XACML (OASIS, 2010), introduce combining algorithms, which solve conflicts by prioritizing the application of the rules according to some strategy. Standard and well known strategies are Deny-Overrides, Permit-Overrides, First-Applicable, and Only-One-Applicable. As an example, if the Deny-Overrides algorithm is chosen to solve the conflict that could arise among the rules of the same policy, the result of the evaluation of the policy is Deny if the evaluation of at least one of the rules returns Deny. Instead, if Permit-Overrides is chosen, the re-

Notes on Complexity. To estimate the complexity of the DSA-Analyser, we consider the steps described at the beginning of this section. The time consuming steps are mainly Step 6 and Step 7, while the other steps have a constant cost that does not depend on the number of rules. Step 6 and Step 7 consist of three main functions:

1. the generation of the context matrix (Algorithm 1);
2. the evaluation of the set of rules by Maude;
3. the pairwise comparison of the Maude evaluation results (between each authorization and prohibition and each prohibition and obligation).

The generation of the context matrix is described in Section 3.1.1 and Algorithm 1. The cost can be overestimated considering the combinations of all the properties of the vocabulary, without repetitions, \( O(num\_prop \times num\_terms) \), where \( num\_prop \) is the number of the properties and \( num\_terms \) is the number of terms in the considered vocabulary.

The cost of the second function depends on the Maude engine. To evaluate the rules of a DSA, we exploit the Maude built-in command \texttt{red}; the tool performs as many \texttt{red} calls as the number of rules of the DSA. Hence, the computational costs of this functions depends on the number of rules, hereafter denoted by \( n \), and by the computational costs of the \texttt{red} function in Maude.

The computational cost of the third function is of \( O(n^2) \) order, being the pairwise comparison of the evaluation of the rules.

Thus, the DSA-Analyser complexity is strictly related to the complexity of the Maude engine (second function), plus an additional factor that depends on the third function, \( (O(n^2)) \), all multiplied by the number of iterations of the first function \( (O(num\_prop^num\_terms)) \).

4 NOTES ON RULES PRIORITIZATION

The DSA-Analyser outputs either the confirmation that no conflicts arise among the evaluated rules or the complete list of conflicts, each of them associated to the related context. In Figure 4, the alert message highlights two potential conflicts, one between authorization \#1 and prohibition \#1, and one between the same prohibition and authorization \#2. Focusing on the former alert, it says that, at DSA enforcement time, there could be an access request that should be authorized (according to authorization \#1) and should be denied (according to prohibition \#1). The access request is evaluated under the context specified in the figure (i.e., the subject is a patient, s(he) is located in a certain area, the data are medical ones, of type ECG, they are stored within the European area, and so on).

To fix a conflictual situation, it is possible to re-editing the rules which may lead to conflicts, and re-running the analysis phase once again. However, it is also possible to leave the rules as they are, and to define ad hoc resolution strategies that will act at enforcement time. Indeed, well-known standard policy languages, such as XACML (OASIS, 2010), introduce combining algorithms, which solve conflicts by prioritizing the application of the rules according to some strategy. Standard and well known strategies are Deny-Overrides, Permit-Overrides, First-Applicable, and Only-One-Applicable. As an example, if the Deny-Overrides algorithm is chosen to solve the conflict that could arise among the rules of the same policy, the result of the evaluation of the policy is Deny if the evaluation of at least one of the rules returns Deny. Instead, if Permit-Overrides is chosen, the re-

Notes on Complexity. To estimate the complexity of the DSA-Analyser, we consider the steps described at the beginning of this section. The time consuming steps are mainly Step 6 and Step 7, while the other steps have a constant cost that does not depend on the number of rules. Step 6 and Step 7 consist of three main functions:

1. the generation of the context matrix (Algorithm 1);
2. the evaluation of the set of rules by Maude;
3. the pairwise comparison of the Maude evaluation results (between each authorization and prohibition and each prohibition and obligation).

The generation of the context matrix is described in Section 3.1.1 and Algorithm 1. The cost can be overestimated considering the combinations of all the properties of the vocabulary, without repetitions, \( O(num\_prop \times num\_terms) \), where \( num\_prop \) is the number of the properties and \( num\_terms \) is the number of terms in the considered vocabulary.
sult of the evaluation of a policy is Permit if the evaluation of at least one of the rules in the policy returns Permit. However, standard combining algorithms are coarse grained, mainly because they take into account the result of the rules evaluation (e.g., Deny-Overrides and Permit-Overrides) and their order in the policy (e.g., First-Applicable). We could envisage other aspects for rule prioritization, such as i) the issuer of the rules, ii) the data category, and iii) the purpose of the data sharing, which is classified according to national and international regulations. Hence, we are able to provide a finer combining algorithm that takes into account not only the result evaluation of the rule itself but also according to the evaluation of properties of the rules. As a simple example, let the reader consider a medical datum, which has to be shared between a hospital and a patient, with the purpose of giving diagnoses. We can imagine that the DSA referred to those data features a potential conflict, between an authorization rule - set by the patient whose data refer to - and a prohibition rule, set by a policy expert working at the hospital. For instance, the conflict could arise when the patient tries to access the data from outside the hospital in which the data have been produced. Being the purpose of data sharing related to giving diagnoses, and being those medical data related to the patient, one possible rule prioritization strategy could be to apply the rule defined by the patient, thus granting the access to data.

5 RELATED WORK

In the last decades, Academia has successfully proposed a huge number of work on data and resource sharing management through privacy policies, also expanding the investigation to aspects like usage control (the monitoring and enforcement of security and privacy policies, not only at time of accessing a resource, but also once the resource has been accessed, see seminal work in (Pretschner et al., 2006; Park and Sandhu, 2004). These studies have been extended across several dimensions and applications, as in (La-

zouski et al., 2014) for the enforcement of usage control policies on Android mobile devices. Thus, the research efforts cited in this section are not intended to constitute an exhaustive list. We would like to highlight that the DSA-Analyser proposed in this paper has the advantage to be fully integrated in a wide and general framework, which manages DSA from its creation, passing through DSA editing, analysis, and enforcement (as the result of a FP7 European project). The kind of data sharing rules in a DSA include also usage control rules, and, even in this case, our general framework allows for analysis and enforcement of such rules. Furthermore, the analyser requires few human intervention, by automatically and exhaustively evaluating the rules in a DSA against all possible contextual conditions that are tied to the DSA vocabulary.

Regarding DSA, work in (Matteucci et al., 2011b) presents a preliminary analysis tool for conflict detection among DSA clauses. The analysis worked by considering one single context at a time, which was manually edited by the user. Successively, the authors of (Casassa Mont et al., 2015) described the integration of a set of tools, for policy authoring and analysis, into a working enforcement framework, specifically tailored for cloud systems. Work in (Matteucci et al., 2011a) is focused on medical data protection policies. Work in (Martinelli et al., 2012) distinguished between unilateral and multilateral DSA (the latter being agreements constituting of data sharing policies coming from multiple actors) and proposed a conflict detection technique with a higher level of automation with respect to (Matteucci et al., 2011b). In (Bicarregui et al., 2008), it was shown an application of the Event-B language\(^5\) to model obliged events. The Rodin platform provides animation and model checking toolset to analyse specifications in Event-B, leading to analysis of obligations (Arenas et al., 2010). Work in (De Nicola et al., 2000) proposed a formal definition of conflicts, together with efficient conflict-checking algorithms. The authors of (Hansen et al., 2008) consider usage control policies to restrict the continuous usage and replication

\(^5\)www.event-b.org
of information, e.g., imposing that certain information may only be used - or copied - a certain number of times. Related to the sharing of data, but not strictly related to analysis, (Lupu and Sloman, 1999; Scalavino et al., 2009; Scalavino et al., 2010) present an evaluation scheme for sharing data in a secure way in a crisis management scenario, through opportunistic networks. Work in (Liang et al., 2013) presents a conflict-detection tool based on first order logic, whose performances are compared to the ones in (Huang and Kirchner, 2011), where the authors use coloured Petri nets process for policy analysis. Our performances outcome are competitive with respect to those two results. A popular and general approach for solving conflicts among privacy rules is the one adopted by the eXtensible Access Control Markup Language (XACML) and its associated policy management framework (OASIS, 2010). XACML policies (or policy sets) include a combining algorithm that defines the procedure to combine the individual results obtained by the evaluation of the rules of the policy (of the policies in the policy set). Work in (Lunardelli et al., 2013; Matteucci et al., 2012a) is an example on how standard XACML combining algorithms can be extended, e.g., evaluating - through well known techniques for multi-criteria decision making (Saaty, 1990) - how much the attributes in a policy are specific in identifying the subject, the object, and the environment of the policy.

6 CONCLUSIONS

In this paper, we have considered electronic contracts consisting of several data sharing rules, possibly edited by more than one actor. Aiming at signaling to the editors potential conflicts among the rules, we have designed and developed an analysis tool, which evaluates set of rules with different effect (access granted/access denied) under all the contextual conditions which may arise from the vocabulary and properties associated to the DSA. The performance results indicate the feasibility of the application of our proposal, for scenarios featuring up to hundreds of rules and up to dozens of terms in the vocabulary (which, to the best of our expertise in the field of healthcare, public administration, and business scenarios, represent realistic numbers for DSA-based practical applications). A possible improvement, which we leave for the future, is to optimise the analysis by paralelling it into three different processes, for authorizations, prohibitions, and obligations.

ACKNOWLEDGEMENTS

Partially supported by the FP7 EU project Coco Cloud [grant no. 610853] and the H2020 EU project C3ISP [grant no. 700294].

REFERENCES


