

A Design Phase for Data Sharing Agreements^{*}

Ilaria Matteucci[#], Marinella Petrocchi[#], Marco Luca Sbodio^b, and Luca Wiegand[#]

[#] IIT-CNR, Pisa, Italy

E-mail: name.surname@iit.cnr.it

^b HP Innovation Center, Italy

E-mail:marco.sbodio@hp.com

Abstract. The number of factories, service providers, retailers, and final users that create networks and establish collaborations for increasing their productivity and competitiveness is constantly growing, especially by effect of the globalization and outsourcing of industrial activities. This trend introduces new complexities in the value supply chain, not last the need for secure and private data sharing among the collaborating parties. A Data Sharing Agreement (DSA) represents a flexible means to assure privacy and security of electronic data exchange. DSA is a formal document regulating data exchange in a controlled manner, by defining a set of policies specifying what parties are allowed, or required, or denied to do with respect to data covered by the agreement. A key factor in the adoption of DSAs is their usability. Here, we propose an approach for a consistent and automated design phase of the agreements. In particular, we present an authoring tool for a user-friendly and cooperative editing of DSA and an analysis tool to identify possible conflicts or incompatibilities among the DSA policies.

Keywords: Data Protection, Policy Specification, Policy Authoring, Policy Analysis.

1 Introduction

In the last twenty years, European and American manufacturing have experienced a deep restructuring, which has been increasingly characterized by globalization and outsourcing of industrial activities. These transformations contribute to maintaining and even improving the profitability of the manufacturing enterprises. However, they also introduce new complexities in the value supply chain:

- Some core manufacturing activities no longer happen within the central premises of manufacturing firms, but are more and more decentralized to partners and remote plants. Consequently, the relationships with partners and suppliers, as well as coordination and governance of the activities among them, become critical factors;
- Greater dependence on logistics and transport services for raw materials, components, and finished products;

^{*} Work partially supported by the EU project FP7-257930 Aniketos: *Ensuring Trustworthiness and Security in Service Composition*, by the EU project FP7-256980 Nessos: *Network of Excellence on Engineering Secure Future Internet Software Services and Systems*, and by the IIT-funded project Mobi-Care.

- Need to execute operational processes at industrial plants located in areas where the availability of highly skilled workers and spare parts for reparation of sophisticated industrial devices (*e.g.*, robots, programmable logic controllers, measurement devices) is quite limited.

In summary, the organization of many manufacturing enterprises has dramatically changed and it is now based on decentralization and hybridization of productive models. This influences the management and sharing of sensitive data across the boundaries of the originating manufacturing enterprise. In this scenario, it is of utmost importance to ensure that data exchange happens in accordance with well defined and automatically manageable policies.

Data Sharing Agreements (DSA), which are formal agreements regulating how parties share data, enable secure, controlled, and collaborative data exchange. Consequently, infrastructures based on DSA become an increasingly important research topic and promise to be a flexible mechanism to ensure protection of critical data.

The main components of a DSA are the following (see also [1,2] for details):

Title gives a title to the DSA.

Parties defines the parties making the agreement.

Period specifies the validity period.

Data lists the data covered by the DSA.

Authorizations defines authorizations covered by the DSA.

Obligations defines obligations covered by the DSA.

Prohibitions defines prohibitions covered by the DSA.

Date and Signatures contains the date and (digital) signatures of the *Parties*.

In this paper, we focus on the DSA design phase. In order to define a DSA, the involved parties negotiate the respective authorizations, obligations, and prohibitions on data covered by the agreement. The design phase is iterative: the authoring of the DSA is followed by analysis of its content in order to identify possible inconsistencies and conflicts among the clauses. This process is iterated until all incompatibilities are solved, and parties have reached agreement on the content of the DSA.

In [2], we develop CNL4DSA, an authoring language for data sharing policies. In this paper, we are going to build on this language. In particular, we present an authoring tool for editing DSA authorization, obligation, and prohibition policies in CNL4DSA, and we provide a DSA analysis tool for detecting anomalies of the policies with respect to the author's intent. Indeed, through the analysis tool, it is possible to form a set of queries related to the DSA under investigation, regarding the possibility, or the necessity, to perform some actions under a list of contextual conditions. The authoring tool consists of a web application with a user interface that allows to write policies by choosing controlled terms from a drop-down menu. Such terms are predefined in a vocabulary configurable by the author. The authoring tool automatically encodes significant sections of a DSA in the CNL4DSA language. The analysis tool consists of two parts: a formal analysis engine allowing automated reasoning on the DSA policies, and a graphical interface through which the user can select the DSA and the contextual conditions in which that DSA is going to be analysed, can formulate a set of queries and

can visualise the answers. Moreover, we present an executable version of CNL4DSA in the Maude language, which serves as the Maude input.

Authoring and analysis examples are presented through a reference agreement related to a scenario in which a set of industrial manufacturers and service providers need to share data in a controlled way.

The paper is structured as follows. Section 2 shows the reference scenario. In Section 3, we present the design and the implementation of the authoring tool. Section 4 introduces the design and implementation of the analysis tool. In Section 5, we comment on a usability study related to the proposed tools. Section 6 discusses related work in the area. Finally, Section 7 concludes with final remarks and insights for future work.

2 Example Scenario

We consider a scenario where a car manufacturer stocks produced cars using an outsourced parking service. The data related to production, stocks, and sale of cars flow across organizational boundaries.

The *XYZ car manufacturer* produces *custom-built* cars, *runabout* cars, and *station-wagons*. XYZ outsources stocking of produced cars to various parking services: XYZ custom-built cars are stored at *VeryExclusiveParking*, runabout cars are stored at *OrdinaryParking*, and station-wagons are stored at *HappyFamilyParking*. The kind of data we consider are: a) production data, *i.e.*, how many cars are produced by the car manufacturer within a certain period of time; b) sale data (how many cars are sold by the manufacturer); and c) employee's salary data (sensitive information related to the salary of XYZ manufacturer's employees).

The management of sensitive data across organisational boundaries and their different scope (production, sale, salaries) require some sort of agreement between the manufacturer and the parking providers. The set of data policies which can apply to each kind of data can be influenced by contextual factors like, *e.g.*, the role on an actor, her geographical location, the kind of data, and time.

In this scenario, we may consider the following information sharing policies.

- Authorizations
 - A1 *OrdinaryParking* has access to the *production* data of XYZ runabout cars related to the next 6 months;
 - A2 *HappyFamilyParking* has access to the past year *sale* data of XYZ station-wagons. Also, *HappyFamilyParking* can share the *sale* data of XYZ station-wagons cars after 1 year of receiving it;
 - A3 Access to both XYZ *production* and *sale* data is only allowed to car manufacturers that are partners of XYZ;
 - A4 Access to XYZ *sale* data on custom-built cars is allowed to car parks.
- Obligations
 - O1 XYZ manufacturer will delete XYZ *employee's salary* data after 2 years of storing them.
 - O2 After a car park accesses XYZ custom-built cars *sale* data, then XYZ must be notified.
- Prohibitions
 - P1 Access to XYZ *sale* data on custom-built cars is not allowed to car parks outside the European Community.

3 DSA Authoring

In this section, we recall the language that we use for specifying Data Sharing Agreements and we describe the design and implementation of the authoring tool.

3.1 Authoring Language

In [2], a Controlled Natural Language for Data Sharing Agreements, namely CNL4DSA, has been proposed. The language allows to formally specify security policies without loosing simplicity of use for end-users. Aiming at making this paper as self-contained as possible, we briefly recall CNL4DSA.

The core of CNL4DSA is the notion of *fragment*, a tuple $f = \langle s, a, o \rangle$ where s is the subject, a is the action, o is the object. The fragment expresses that “the subject s performs the action a on the object o ”, e.g., “Bob reads Document1”. It is possible to express authorizations, obligations, and prohibitions by adding the CAN / MUST / CANNOT constructs to the basic fragment. Fragments are evaluated within a specific *context*. In CNL4DSA, a *context* is a predicate c that usually characterizes environmental factors, such as time and location. Some examples of simple contexts are “more than 1 year ago” or “inside the European Community”. Contexts are predicates that evaluate either to *true* or *false*. In order to describe complex policies, contexts need to be composable. Hence, we use the boolean connectors *and*, *or*, and *not* for describing a *composite context* C which is defined inductively as follows.

$$C := c \mid C \text{ and } C \mid C \text{ or } C \mid \text{not } c$$

The syntax of the *composite authorization fragment* F_A , used for expressing authorization policies, is inductively defined as follows.

$$F_A := \text{nil} \mid \text{can } f \mid F_A; F_A \mid \text{if } C \text{ then } F_A \mid \text{after } f \text{ then } F_A \mid (F_A)$$

The intuition is the following:

- *nil* can do nothing.
- *can f* is the atomic authorization fragment that expresses that f is allowed. Its informal meaning is *the subject s can perform the action a on the object o* .
- $F_A; F_A$ is a list of composite authorization fragments (*i.e.*, a list of authorizations).
- *if C then F_A* expresses the logical implication between a context C and a composite authorization fragment: if C holds, then F_A is permitted.
- *after f then F_A* is the temporal sequence of fragments. Informally, after f has happened, then the composite authorization fragment F_A is permitted.

Similarly, the syntax of a composite obligation fragment, used for expressing obligation policies, is inductively defined as follows.

$$F_O := \text{nil} \mid \text{must } f \mid F_O; F_O \mid \text{if } C \text{ then } F_O \mid \text{after } f \text{ then } F_O \mid (F_O)$$

The intuition is the same as for F_A , except for *must f* that represents the atomic obligation: *the subject s must perform the action a on the object o*. *must f* expresses that *f* is required.

Finally, the syntax of a composite prohibition fragment, used for expressing prohibition policies, is inductively defined as follows.

$$F_P := nil \mid cannot\ f \mid F_P; F_P \mid if\ C\ then\ F_P \mid after\ f\ then\ F_P \mid (F_P)$$

The atomic prohibition is represented by *cannot f*: *the subject s cannot perform the action a on the object o*. *cannot f* expresses that *f* is not permitted.

CNL4DSA has an operational semantics based on a modal transition system, able to express *admissible* and *necessary* requirements to the behaviour of the CNL4DSA specifications [2,3].

3.2 Examples

Here, we show some simple examples of policy specification in CNL4DSA. Such examples refer to the scenario depicted in Section 2.

Authorization A1: OrdinaryParking has access to the production data of XYZ runabout cars related to the next 6 months.

We can rephrase it as

if a car parking service has role *OrdinaryParking*, and some data has category *Production*, and that data refer to *Runabout*, and that data refer to *the next six months*, and that data refer to *XYZ CarManufacturer* then that car parking service can access that data.

We can now use CNL4DSA syntax to express the authorization:

IF (c1 AND c2 AND c3 AND c4 AND c5) THEN CAN f1

- c1 = {Car Park has role OrdinaryParking} is a context;
- c2 = {Datum has data category Production} is a context;
- c3 = {Datum refers to Runabout} is a context;
- c4 = {Datum refers to Next6Months} is a context;
- c5 = {Datum refers to XYZ CarManufacturer} is a context;
- c1 AND c2 AND c3 AND c4 AND c5 is a composite context;
- f1 = {Car Park has access to Datum} is an atomic fragment with subject "Car Park", action "access", and object "Datum".

Obligation O2: After a car park accesses XYZ custom-built cars sale data, then XYZ must be notified.

After rephrasing the sentence, we can express it using CNL4DSA as:

IF (c1 AND c2 AND c3) THEN (AFTER f1 THEN MUST f2)

- c1 = {Datum refers to Custom-Built} is a context;
- c2 = {Datum has data category Sale} is a context;
- c3 = {Datum refers to XYZ} is a context;
- f1 = {Car Park has access to Datum} is an atomic fragment with subject “Car Park”, action “access”, and object “Datum”.
- f2 = {System notifies XYZ} is an atomic fragment with subject “System”, action “notify”, and object “XYZ”.

Prohibition P1: Access to XYZ sale data on custom-built cars is not allowed to car parks outside the European Community.

In CNL4DSA:

IF (c1 AND c2 AND NOT c3 AND c4) THEN CANNOT f1)

- c1 = {Datum refers to Custom-Built} is a context;
- c2 = {Datum has data category Sale} is a context;
- c3 = {Car Park has location EuropeanCommunity} is a context;
- c4 = {Datum refers to XYZ} is a context;
- f1 = {Car Park has access to Datum} is an atomic fragment with subject “Car Park”, action “access”, and object “Datum”.

3.3 Authoring tool

The DSA Authoring tool supports users in the creation of DSAs. Its main functionality consists of the simplified and controlled editing of a DSA, and the automatic encoding of its key sections in the CNL4DSA language. The resulting DSA document is in a XML file. CNL4DSA expressions are embedded into the XML representation of the DSA document, and they encode the sections Authorizations, Obligations and Prohibitions. The output of the DSA Authoring tool (*i.e.*, the XML file with the embedded CNL4DSA expressions) is the machine-processable document that enables the automation of the DSA lifecycle ¹.

The DSA Authoring tool is a Web application that displays the DSA sections by using editable widgets, thus allowing the author to edit sections Authorizations, Obligations, and Prohibitions in a controlled way. The Web application guides the user in building semi-natural statements by using predefined terms taken from a controlled vocabulary. Such statements are automatically encoded in CNL4DSA and stored in the XML representation of the DSA. Additionally, the DSA Authoring tool automatically builds the Data section of XML DSA, which, for the sake of simplicity, is not displayed to the end user. Thus the DSA Authoring tool displays a human-readable version of the DSA, and allows the user to interactively edit its section in an assisted way.

Figures 1, 2, and 3 are snapshots of the DSA Authoring tool during the editing phase of our fictitious XYZ car manufacturer DSA. Fig. 1 shows the Authorizations and Obligations section of the DSA, where CNL4DSA is rendered in semi-natural language.

¹ The DSA Authoring tool is the subject of the international patent application PCT/EP2011/058303 filed by Hewlett-Packard Development Company LP.

Additionally, Fig. 1 shows the “highlight references” functionality of the DSA Authoring tool, which helps the end user in better understanding the statements structure. References are used when the subject or object of a CNL4DSA fragment are actually a reference to a previously used variable (either in the same, or in other statements). For example, Fig. 1 shows that the three expressions “those data” appearing in the first statement (highlighted in light blue), actually refer to previously used term “a datum” (highlighted in dark blue). When a DSA is complex and contains many authorizations and/or obligations and/or prohibitions, the “highlight references” functionality is very useful for end users authoring and/or reading the DSA.

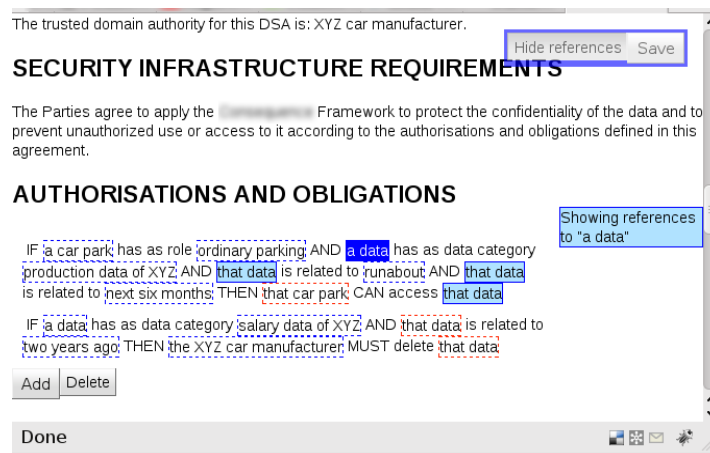


Fig. 1. Showing references.

Fig. 2 shows how the DSA Authoring tool guides the user in writing a new CNL4DSA statement. The user is currently adding one of the example statements for the XYZ car manufacturer DSA. The vocabulary window on the bottom-right side of the screenshot shows a list of available terms. Terms are taken from a predefined controlled vocabulary, and their availability changes depending on the evolving structure of the statement currently being edited. In the example screenshot the user has previously selected the term “a datum”, and therefore the tool is now suggesting to use one of the available predicates, in order to build a valid CNL4DSA statement.

Finally, Fig. 3 shows how the DSA Authoring tool helps the DSA author in using references. Following the previously described example statements, the DSA author should now enter the context ... AND { that datum is related to station wagon }, where the term “that datum” refers to a previously defined “datum”. To achieve this, the DSA author select “REFERENCE” from the vocabulary window, and the DSA Authoring tool automatically shows with a light green background all the previously used terms that the user can refer to. When the user moves the mouse over these terms, the DSA Authoring tool highlights them in dark green. Finally, when

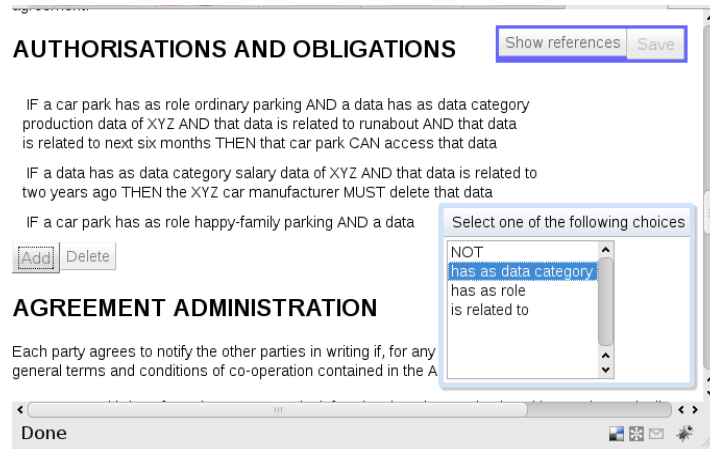


Fig. 2. Adding a predicate.

the user clicks on the currently highlighted one (“a datum”), the DSA Authoring tool builds a reference to it (that will be displayed as “that datum”).

Fig. 4 shows a fragment of the XML serialization of our example DSA for XYZ car manufacturer. The green box highlights an authorization statement. Each authorization (or obligation, or prohibition) has an automatically generated identifier, which is unique within the scope of the DSA; the DSA Authoring tool generates these identifiers. An authorization consists of several equivalent expressions based on different languages. Besides the `UserText` version, the figure shows the corresponding `CNL4DSA` expression, and also the `CNL4DSA-E` expression. The latter is a simplified version of `CNL4DSA`, which is generated on the fly by the DSA Authoring tool while the user is building the authorizations/obligations/prohibitions statements. The DSA Authoring tool takes care of generating the formal `CNL4DSA` expressions when the DSA is saved and serialized in XML.

From the implementation perspective, the DSA Authoring tool is a lightweight Web application, in which the computation tasks (mostly related to `CNL4DSA` generation and processing) are carefully partitioned between the client-side and the server-side. Implementation-wise, there are two interesting aspects of the DSA Authoring tool: the agile web-based application, and the syntax driven editor.

Implementation of the Web application The Web application is based on AJAX techniques, which allows the creation of rich client-side user interaction.

The client-side Web application (running inside a Web browser) asynchronously communicates with the server to retrieve data (for example the vocabulary), and to delegate computation-intensive tasks (for example the serialization of the DSA into XML with embedded `CNL4DSA`).

The server-side part, that runs in a Java application server, manages a DSA repository, that contains the XML serializations of the DSA documents. Additionally, the server-side part manages the vocabularies of terms that the DSA author can use when editing authorizations, obligations, and prohibitions. Such vocabularies are currently

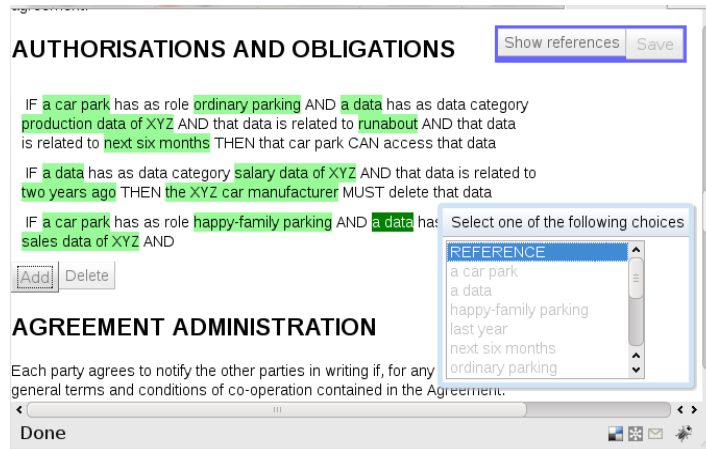


Fig. 3. Adding a reference.

defined by XML files, that give the definitions of actions, subjects, objects and predicates. The architecture is flexible, and it allows for adding and configuring new vocabularies, which become instantly available for the DSA authors.

Implementation of the syntax driven editor The most sophisticated part of the DSA Authoring tool is the editor guiding the user in building new statements. This editor is based on the formal definition of the CNL4DSA grammar. Based on an ANTLR² grammar definition we derived a Finite State Machine (FSM) that implements the CNL4DSA grammar, and we implemented the FSM on the client-side of the DSA Authoring tool. The type of the term selected by the user (action, subject, object, predicate) determines the state transitions of the FSM. The syntax driven editor essentially ensures that the user builds syntactically correct policies.

4 DSA Analysis

Here, we present the DSA analysis process, which allows to answer questions related to the allowance, or the necessity, to perform some particular actions, like the following.

- *Action list*: what are all the authorised actions in the investigated set of policies, under a set of contextual conditions?
- *Answer to specific authorization-related queries*: is it true that subject x is authorised to perform action z on object y, under a set of contextual conditions?
- *Answer to specific obligation-related queries*: is it true that subject x is obliged to perform action z on object y, under a set of contextual conditions, after that a subject w performs an action a on object o?
- *Check conflicts*: is it true that subject x can perform action z on object y and that, under the same context, subject x cannot perform action z on object y?

² <http://www.antlr.org/>

```

--<datum id="DATUM_X79">
  --<expression language="CNL4DSA">
    ?X79 is-a <http://www.consensus-project.eu/ontology/ontology/ #Data>
  </expression>
</datum>
</data>
--<authorizations>
--<authorization id="AUT_1271750718935">
  --<expression language="CNL4DSA-E">
    IF ?X77:CarPark has_role ?X78:OrdinaryParking AND ?X79:Data has_data_category ?X80:XYZProduction AND ?X79
    is_related_to ?X81:Runabout AND ?X79 is_related_to ?X82:NextSixMonths THEN ?X77 CAN access ?X79
  </expression>
  --<expression language="UserText">
    IF a car park has as role ordinary parking AND a data has as data category production data of XYZ AND that data is
    related to runabout AND that data is related to next six months THEN that car park CAN access that data
  </expression>
  --<expression language="CNL4DSA">
    IF has_role(?X77,?X78) AND has_data_category(?X79,?X80) AND is_related_to(?X79,?X81) AND
    is_related_to(?X79,?X82) THEN CAN [?X77, access, ?X79]
  </expression>
</authorization>

```

Fig. 4. XML serialization of the XYZ car manufacturer DSA.

The main goal is to check if the authorization, obligation, and prohibition policies have been specified accordingly to the author intent. The analyser checks the execution trace of such policies, to understand if a particular action happens or not. This means checking if a certain safety property holds or not. It is possible to define a set of *security relevant actions*, *i.e.*, a set of actions that, when executed, may affect the security of the system. The safety properties we analyse are those stating that a certain security relevant action appears in an execution trace of the set of policies under investigation.

4.1 The Analysis Tool

The analysis tool consists of two parts:

- a formal engine that actually performs the analysis of the policies;
- a graphical user interface that allows the user to dynamically load contextual conditions and queries, launch the analysis process, and visualize the results.

The Engine CNL4DSA has been designed with a precise formal semantics, based on a modal labelled transition system [3]. Thus, the language is governed by rules regulating states and transitions between these states. This allows for a precise translation of CNL4DSA in Maude. Maude is a programming language that models distributed systems and the actions within those systems [4]. Systems are specified by defining algebraic data types axiomatizing systems states, and rewrite rules declaring the relationships between the states and the transitions between them. Maude is executable and comes with a toolkit that allows formal reasoning of the specifications produced. In particular, the Maude facilities can be exploited to search for allowed traces, *i.e.*, sequence of actions, of a policy specified in CNL4DSA. These traces represent the sequences of actions that are authorised, or required, or denied by the policy.

CNL4DSA has been made executable by translating its syntax and formal semantics in Maude. An excerpt of the translation is shown in Fig. 9. The Maude template

used for the analysis of DSA authorizations, obligations, and prohibitions is available online at: www.iit.cnr.it/staff/marinella.petrocchi/template.maude. This template contains static parts defining the translation from CNL4DSA to Maude and logic operators. Also, some modules are dynamically loaded depending on the kind of policies, contextual conditions, and queries that the user is going to deal with.

The Graphical User Interface The GUI is deployed as a Web Application and it allows the user to query the analysis engine and visualize its results. The analysis engine exposes its functionalities as Web Service methods. The GUI is in charge of retrieving the set of policies that a user wants to analyse and the related vocabulary from a repository. Each vocabulary is implemented as an ontology and the inner logic of the GUI exploits it in order to create and show a set of menus whose information is consistent with the vocabulary.

The interface helps the user to create dynamic contexts, which represent the environment under which the analysis will be performed. The inner logic of the GUI updates the information according to the selected context. Furthermore, it is possible to compose different types of queries, related to authorizations, obligations, and prohibitions. Once the user has selected both context and queries, the GUI sends all the inputs, *i.e.*, the vocabulary, the CNL4DSA/Maude specification of the policies, the context defining the conditions on which the policies have to be evaluated, and the set of queries to the engine that performs the analysis. When the analysis has been performed, the results are shown through the GUI.

4.2 Analysis Example

Here, we show some example analyses over the policies of reference³. Fig. 5 and Fig. 6 show some snapshots of different phases of the analysis process.

The user can select the contextual conditions under which the analysis is carried out from a drop-down menu (see Fig. 5). The menu is dynamically created according to the vocabulary of the loaded policies. All the selected contexts are automatically set to *true*. We assume that everything that it is not explicitly specified does not hold. Hence, the user shall select each context that is supposed to be true.

Then, the user can define the set of queries. In Fig. 6, we show an example of query composition. The user can select queries representing either authorizations, or obligations, or prohibitions. All the selected queries are shown to the user in the English natural language (see Fig. 7). Once that the user has selected both context and queries, she can start the analysis process by pressing the Submit button. This launches the inner analysis engine. At the end of the process, the GUI shows the analysis result to the user.

Finding the traces allowed by a set of policies is particularly useful to detect conflicts before the actual enforcement of those policies. As an example, we show how an authorization and a prohibition of our reference scenario leads to a conflict. They are authorization A4 and prohibition P1 listed in Section 2. Indeed, at the same time, they

³ The GUI is available online at <http://dev4.iit.cnr.it:8080/DsaAnalyzerWebGUI-0.1/?dsaID=cars.xml>. The interested reader should press the *Submit for the Analysis* button in order to load our reference policies and the related vocabulary.

```

Context:
1 eq eval (hasdatacategory(data,sale) = true .
2 eq eval (isreferredto(data,custombuilt) = true .
3
4
5
6
7
8
9

```

1 Delete contextLine

Insert Context

The property: isReferredTo

Has datacategory: hasDataCategory

Has codomain: Ca

isReferredTo

Add contextLine

Fig. 5. Selection of context for the XYZ car manufacturer DSA analysis.

Select Query

MUST

The action: Notify

Being performed by the subject: System

On the object: Carmanufacturer -> XYZ

After: Access

Being performed by the subject: Carparking

On the object: Data

?

Expected: True

Add query

Fig. 6. Selection of queries for the XYZ car manufacturer DSA analysis.

give and deny to car parkings the possibility of accessing sale data of custom-built cars manufactured by XYZ. This happens when the following contextual conditions are set:

- datum has data category *sale*
- datum refers to *custom-built*
- datum refers to *XYZ*

Indeed, according to authorization A4, a car park is allowed to access sale data of XYZ car manufacturer. On the other hand, prohibition P1 is activated since it is not explicitly stated that *the car parking has location European Community*. Recall that everything that is not explicitly said it is not true, thus the car park is located outside the European Community. A conflict occurs and an alert message is shown to the user. Thus, she can decide to go back to the authoring phase to modify the clauses that give the conflict, before their enforcement. The conflict detection is shown in Fig. 7. Currently, the system needs an explicit creation of queries that might capture conflicts. In the final remarks, we give some hints for a possible improvement of the conflict detection process.

Query	Expected	Result	Statement
Can a carparking access a data?	true	true	
Cannot a carparking access a data?	false	true	

 Table of Access

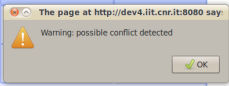


Fig. 7. Conflict detection.

Finally, through the user interface it is possible to save the current configuration (*i.e.*, a set of contextual conditions and a set of queries) for successive elaborations (see Figure 8). This functionality allows to load a saved session without redefining contexts and queries. This is useful when the user, that possibly detects a conflict among the policies, modifies those policies. When checking the correctness of the modified clauses, there is no need to reformulate the contextual conditions and the queries.

Load / Save Context and Queries

Fig. 8. Load and save a configuration.

5 Discussion

This paper presents two tools that can be exploited as the front-end for a data sharing agreements management system. To assess the degree of the user-friendliness of such tools, a first usability study has been carried out by external evaluators, both technicians and industrial managers, within the FP7 European Project Consequence

```

COMPOSITE CONTEXT
C := c | C and C | C or C | not c

COMPOSITE FRAGMENT
FA := nil | can f | FA; FA |
      if C then FA | after f then FA | (FA)
FO := nil | must f | FO; FO |
      if C then FO | after f then FO | (FO)
FP := nil | cannot f | FP; FP |
      if C then FP | after f then FP | (FP)

fmod FRAGMENT-CONTEXT is
--- this module declares fragments and contexts
...
sorts Term Action Basicfragment .
sorts Contesto CompCont . subsort Contesto < CompCont .
vars CC CC' : CompCont . var C : Contesto .
op eval : CompCont -> Bool .
--- logic operators for composing contexts:
op _ and _ : CompCont CompCont -> CompCont [ass comm prec 57] .
op _ or _ : CompCont CompCont -> CompCont [ass comm prec 59] .
op not_ : Contesto -> CompCont [prec 53] .
ceq eval(CC and CC') = true if eval(CC) ^ eval(CC') .
ceq eval(not C) = true if eval(C) == false .
...
--- Basicfragment declaration:
op <_,_,> : Term Action Term -> Basicfragment .
--- Context declaration:
op -(_,_) : Assertion Term Term -> Contesto . endfm

fmod CNL4DSA-SYNTAX is
inc FRAGMENT-CONTEXT .
sort Fragment .
--- syntax:
--- can/cannot/after
op ... : Basicfragment Fragment -> Fragment [frozen prec 25] .
--- list
op ;_ : Fragment Fragment -> Fragment [frozen assoc comm prec 30] .
--- if
op @_ : CompCont Fragment -> Fragment [frozen prec 25] .
--- obl
op * _ * _ : Basicfragment Fragment -> Fragment [frozen prec 25] .
endfm

```

Fig. 9. The syntax of CNL4DSA (top) and its translation in MAUDE (bottom).

(www.consequence-project.eu)⁴. Consequence designed and developed an integrated framework for DSA authoring, analysis, and enforcement. In particular, the authors of this paper worked on the DSA authoring and analysis phase. The Consequence results have been evaluated on two testbeds, and the results of those evaluations are available as public reports [5,6]. These two final deliverables accurately detail advantages and drawbacks of all the Consequence architecture components, including authoring and analysis. It was our intention to inherit the suggestions of the evaluators in order to improve the usability of the tools. For the time being, some updates have been done, like, *e.g.*, the insertion of a *help on line* facility that may help non expert users, like industrial managers. This facility is integrated in the online analysis GUI. However, we aim at fixing some further issues and to carry out a second usability survey.

6 Related Work

In the literature, there are proposals for policies authoring and analysis. We compare our framework with existing work in the area.

Work in [7] investigates platform-independent policy frameworks to specify, analyze, and deploy security and networking policies. The authors describe a prototype for usable and effective policy authoring through either natural language or structured lists that manage policies from the specification to the possible enforcement. Proposals in [8,9] specifically focus on DSA, by modeling the agreement as a set of obligation constraints. Obligations are expressed as distributed temporal logic predicates (DTL), a generalization of linear temporal logic including both past-time and future-time temporal operators. The authors of [10] proposes SPARCLE as an authoring language and the SPARCLE Policy Workbench as an application for editing privacy policies. Even if the proposed language is not based on a formal syntax and semantics, this work identifies some grammars for parsing natural language into SPARCLE. In [11], the authors propose a Datalog-like authoring language as the input for a policy editor. Focus is given on social and environmental aspects that can influence the interpretation and specification of the policies, like, *e.g.*, trust and privacy aspects. In [12], the authors offer a user-friendly, visual interface for the specification of the underlying concepts of privacy-preserving authorizations, such as roles, data types, actions, rules and contextual information, providing the appropriate level of abstraction. With respect to these work, we also deal with the problem of (automated) DSA verification: the formal foundation of CNL4DSA let us leverage existing analysis frameworks like Maude.

Other alternative approaches are possible for analysing DSA. Binder [13] is an open logic-based security language that encodes security authorizations among components of communicating distributed systems. It has a notion for context and provides flexible low-level programming tools to express delegation, even if Binder does not directly implement higher-level security concepts like delegation itself. Also, the Rodin platform provides animation and model-checking toolset, for developing specifications based on the Event-B language (www.event-b.org). In [14], it was shown that the Event-B language can be used to model obliged events. This could be useful in the case of analysing

⁴ The project ended in January, 2011. The website and all its content will be available online until Dec 31, 2013.

obligations in DSA. In [15], the authors present a formalization of DSA clauses in Event-B and the ProB animator and model checker are exploited in order to verify that a system behaves according to its associated DSA. The main difference with our approach is that CNL4DSA captures the events (or actions) that a system can perform, the order in which they can be executed and it can be easily extended for dealing with other aspects of this execution, such as time and probabilities. On the other hand, Event-B is a “state-oriented” language that models a state of a system rather than its transition.

Even if not specifically DSA-related, [16] presents a policy analysis framework which considers authorizations and obligations, giving useful diagnostic information. Also, a relevant work in [17] proposes a comprehensive framework for expressing highly complex privacy-related policies, featuring purposes and obligations. Also, a formal definition of conflicting permission assignments is given, together with efficient conflict-checking algorithms. Finally, the Policy Design Tool [18] offers a sophisticated way for modeling and analysing high-level security requirements in a business context and create security policy templates in a standard format.

To conclude, there exists generic formal approaches that could *a priori* be exploited for the analysis of some aspects of DSA. As an example, the Klaim family of process calculi [19] provide a high-level model for distributed systems, and, in particular, exploits a capability-based type system for programming and controlling access and usage of resources. Also, [20] consider policies that restrict the use and replication of information, *e.g.*, imposing that a certain information may only be used or copied a certain number of times. The analysis tool is a static analyser for a variant of Klaim.

7 Conclusions and Future Work

We focused on the authoring and analysis phase of DSAs. We developed a user-friendly authoring tool that can exploit capabilities of a background analysis tool. The combination of such tools allows to dynamically define DSAs ensuring privacy and security properties, and to detect conflicts before the actual enforcement of the resulting policies.

We leave some work for the future. Currently, the vocabularies collecting the terms used in a DSA do not carry semantic information, but we plan to evolve them towards more formal ontological definition of terms. Also, there are some open issues related to the detection of conflicts. First, in the current implementation, the conflict detection between an authorization and a prohibition is based on a user query. The analysis tool could be easily extended by automatically looking for all the possible actions that are at the same time allowed and prohibited under a certain context. Secondly, when a conflict is detected, the user manually re-edits the DSA and modifies the policies responsible for that conflict. We are currently working on supporting the user in automatically solving conflicts, once detected. Finally, as it is common for tools based on state exploration, the underlying analysis engine suffers from the problem of the state explosion. Thus, it may be convenient to further investigate the feasibility of using this engine for more complex DSA specifications.

References

1. The Consequence Team: D2.1: Methodologies and Tools for Data Sharing Agreements Infrastructure. http://www.consequence-project.eu/Deliverables_Y1/D2.1.pdf (2008)
2. Matteucci, I., Petrocchi, M., Sbodio, M.L.: CNL4DSA: a Controlled Natural Language for Data Sharing Agreements. In: SAC: Privacy on the Web Track, ACM (2010)
3. Larsen, K.G., Thomsen, B.: A modal process logic. In: LICS. (1988) 203–210
4. Clavel, M., et al., eds.: All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic. In Clavel, M., et al., eds.: All About Maude. Volume 4350 of LNCS., Springer (2007)
5. The Consequence Team: D6.4: Final Evaluation of the Sensitive Data Test Bed. http://www.consequence-project.eu/Deliverables_Y3/D6.4.pdf (2011)
6. The Consequence Team: D5.4: Final Evaluation of the Policy-Based Security for Crisis Management Test Bed. http://www.consequence-project.eu/Deliverables_Y3/D5.4.pdf (2011)
7. Brodie, C., et al.: The Coalition Policy Management Portal for Policy Authoring, Verification, and Deployment. In: POLICY. (2008) 247–249
8. Swarup, V., et al.: A Data Sharing Agreement Framework. In: ICISS. (2006) 22–36
9. Swarup, V., et al.: Specifying Data Sharing Agreements. In: POLICY. (2006) 157–162
10. Brodie, C., et al.: An Empirical Study of Natural Language Parsing of Privacy Policy Rules using the SPARCLE Policy Workbench. In: SOUPS, ACM (2006) 8–19
11. Fisler, K., Krishnamurthi, S.: A Model of Triangulating Environments for Policy Authoring. In: SACMAT, ACM (2010) 3–12
12. Mousas, A.S., et al.: Visualising Access Control: The PRISM Approach. Panhellenic Conference on Informatics (2010)
13. Abadi, M.: Logic in Access Control. In: LICS, IEEE (2003) 228
14. Bicarregui, J., et al.: Towards Modelling Obligations in Event-B. In: ABZ. (2008) 181–194
15. Arenas, A., et al.: An Event-B Approach to Data Sharing Agreements. In: Integrated Formal Methods, Springer (2010) 28–42
16. Craven, R., et al.: Expressive Policy Analysis with Enhanced System Dynamicity. In: ASI-ACCS. (2009)
17. Ni, Q., et al.: Privacy-aware Role-based Access Control. ACM Transactions on Information and System Security **13** (2010)
18. Policy Design Tool: <http://www.alphaworks.ibm.com/tech/policydesigntool> (2009)
19. De Nicola, R., Ferrari, G.L., Pugliese, R.: Programming Access Control: The KLAIM Experience. In: CONCUR. (2000) 48–65
20. Hansen, R.R., Nielson, F., Nielson, H.R., Probst, C.W.: Static Validation of Licence Conformance Policies. In: ARES. (2008) 1104–1111