ABSTRACT
A Data Sharing Agreement (DSA) is an agreement among contracting parties regulating how they share data. A DSA represents a flexible mean to assure privacy of data exchanged on the Web. As an example, a set of intelligent user agents may interact with each other, and by means of DSA, may negotiate privacy requirements on behalf of human users. However, a key factor for the adoption of privacy’s technologies is not only their reliability, but also their usability. Here, we propose CNL4DSA, a Controlled Natural Language for DSA aiming at lowering the barrier to adoption of DSA, and, at the same time, ensuring mapping to formal languages that enable the automatic verification of agreements.

Keywords
Privacy Policy, Data Sharing Agreements, Controlled Natural Language, DSA authoring, DSA analysis.

1. INTRODUCTION
Data sharing is what makes the Web unique. Since its early days, the Web has conveyed an ever growing explosion of data into our everyday life. Recent trends, such as blogging, social networking, on-line chat, and service-based computing have increased the amount of personal data that users share or put on the Web. Additionally, the Web is an open, heterogeneous and constantly changing environment. Ensuring privacy on the Web does not only require standards and technologies, but also the ability to combine them into a simple and extensible framework that supports both the users needs, and the constantly evolving Web infrastructure.

In this scenario, Data Sharing Agreements (DSA) become an increasingly important research topic. A DSA is a formal legal agreement among two or more parties regulating how they share data, and usually consists of a set of authorizations and obligations clauses. The purpose of a DSA is exactly that of defining what parties are allowed or required to do with respect to data covered by the agreement. Note that data sharing agreements are usually subject to a lifecycle consisting (at least) of the following phases: definition, enforcement, disposal1. During the definition phase, the parties negotiate the respective authorizations and obligations on data covered by the agreement. During the enforcement phase, the DSA clauses are enacted by an appropriate infrastructure ensuring that data exchange among parties comply with the DSA clauses. Finally, a DSA may be disposed of when its validity expires, or when one of the contracting parties decides to recede from it. We observe that the definition phase is iterative: authoring of the DSA is followed by analysis of its content in order to identify possible conflicts or incompatibilities among authorizations/obligations clauses. This process is iterated until all conflicts are solved, and parties have reached agreement on the content of the DSA.

DSA promise to be a flexible mechanism to ensure privacy on the Web: an intelligent user agent may in fact autonomously negotiate privacy requirements on behalf of a human user, based, for example, on a set of user preferences, or on a dynamic user interaction. Noticeably, such an intelligent user agent may interact both with other user agents (when negotiating user-to-user privacy requirements), and with Web site agents (when negotiating privacy requirements of the user with respect to the service provided by the Web site). Another interesting aspect of DSA is their multilateralism. Whereas a privacy policy is usually understood as a legal document unilaterally stating how a service provider retains, processes, distributes, and deletes service consumer’s data, a DSA is intrinsically a multilateral agreement in which all parties actively participate.

From a user perspective, a key factor for the adoption of privacy technologies, is not only their reliability, but also their simplicity and usability. Currently, there are various policy specification languages (for example the W3C Recommendation P3P [7], Rei [13], and EPAL [21]) that address different aspects, but generally fail to meet the usability requirement, because they often rely on a formal and unintuitive syntax. In order to increase the readability of a policy language, some researchers ([3, 15, 16], and more recently [6]) have proposed the adoption of controlled natural languages (CNL). Following this approach, we propose here a Controlled Natural Language for Data Sharing Agreements
2. CNL4DSA: SYNTAX AND SEMANTICS

Some examples of Data Sharing Agreements can be found in [9, 10, 11, 20]. In general, a DSA consists of various parts, among which a Title, a validity Period, the list of Data covered by the agreement, the list of involved Parties, their respective Signatures, and (possibly empty) Authorizations and Obligations sections. In this paper we focus on the Authorizations and Obligations sections, illustrating the use of CNL4DSA to define their content in a human understandable way, and, at the same time, to derive a formal specification suitable for automated verification. When the Authorizations and Obligations sections are empty, we assume that the DSA does not impose any obligation, and that it has only the following implicit authorization: “all entities (users, groups, etc.) belonging to Parties are not authorized to make any action on Data during Period”.

Authorizations and obligations refer to data and involve parties specified in the Data and Parties sections of the DSA respectively. Although the definition of the Data and Parties sections is out of scope for this paper, we remark that data items and parties are referred to by using unique identifiers (possibly a URI), and that data types, and related attributes, like, e.g., author, size, creation date, .. are defined in formal vocabularies (ontologies).

In order to specify authorizations and obligations we introduce the notion of fragment denoted as \( f, f_1, \ldots \), ranged over the set \( \mathcal{F} \). The fragment is a tuple consisting of four elements, \( f = (s, a, o, [v]) \), where \( s \) is the subject, \( a \) is the action, \( o \) is the object and \( v \) is a variable. Square brackets denote that \( v \) is an optional element.

The fragment \( f = (s, a, o, [v]) \) expresses that “the subject \( s \) performs the action \( a \) on the object \( o \)”. The optional value \( v \) typically represents the date and/or time, characterizing the occurrence of \( f \). Finally, we observe that the action \( a \) is taken from a list of predefined actions (possibly deriving from an ontology), and that each action can be decorated with additional information (e.g., the action “send” is decorated with the additional information “to …” specifying the recipient).

An example of a fragment using an action is “Bob reads Alice’s profile”, where “Bob” is the subject, “reads” is the action, and “Alice’s profile” is the object”. Our assumption is that everything that is not explicitly expressed is forbidden. Thus, we consider that authorizations and obligations are always given in a positive form, e.g., “Alice can parse the content of personal e-mails”.

More complex expressions are generated by combining fragments. We refer to such expressions as composite fragments, and we denote them as \( F \) (ranged over the set \( \mathcal{F} \)). We distinguish two disjoint sets of composite fragments: authorization fragments, denoted by \( F_A \) and ranged over the set \( \mathcal{A} \), and obligation fragments, denoted by \( F_O \) and ranged over the set \( \mathcal{O} \).

Usually, fragments are evaluated within a specific context. In CNL4DSA, a basic context is a predicate \( c \) that characterizes environmental condition, such as time and location. Some examples of simple contexts are “more than 1 year ago” or “inside the facility”. In order to describe complex agreements, contexts need to be composable. Hence, starting from the basic context \( c \), we use the boolean connectors and, or, and not for describing a composite context \( C \) (ranged over the set \( \mathcal{C} \)) which is defined inductively as follows:

\[
C := c \mid C \land C \mid C \lor C \mid \neg c
\]

For modelling the behaviour of our DSA, we adopt a labelled transition system [2] which add modalities to the derivations of DSA, by exploiting a modal transition system MTS [18, 17].

In its original version, MTS is a structure \((\mathcal{S}, \mathcal{A}, \rightarrow_{\mathcal{O}}, \rightarrow_{\mathcal{C}})\), where \( \mathcal{S} \) is a certain set of specifications (e.g., process expressions, within the framework of Process Algebras), \( \mathcal{A} \) is the set of actions which specifications may perform, and \( \rightarrow_{\mathcal{O}}, \rightarrow_{\mathcal{C}} \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S} \) are the two modal transition relations expressing admissible and necessary requirements to the behaviour of the specifications. In particular, \( S \xrightarrow{a} S' \) with \( S, S' \in \mathcal{S} \) and \( a \in \mathcal{A} \) means that it is admissible that the implementation of \( S \) performs \( a \) and then behaves like \( S' \). Dually, \( S \xrightarrow{\mathcal{O}} S' \) represents a transition in which the implementation of \( S \) is required to perform \( a \) and then behaves like \( S' \). It is assumed that all the required transitions are admissible transitions.

**Authorization Fragment.**

The syntax of a composite authorization fragment is inductively defined as follows:

\[
F_A := \text{nil} \mid \text{can } f \mid \text{FA} \mid \text{if } C \text{ then } F_A \mid \text{after } f \text{ then } F_A \mid \langle F_A \rangle (F_A)
\]

The intuition for the composite authorization fragment is the following:

- \text{nil} can do nothing.
- \text{can } f is the atomic authorization fragment. Its formal meaning is the subject \( s \) can perform the action \( a \) on the object \( o \), with optional value \( v \). can \( f \) expresses that \( f \) is allowed, but not required.
- FA is a list of composite authorization fragments.
- if \( C \) then \( F_A \) is a list of composite authorization fragments.
- the list constitutes the authorization section of the considered DSA. Whenever one term of the list performs a \( f \)-transition, then that term evolves to the correspondent derivative.
- after \( f \) then \( F_A \) expresses the logical implication between a composite context \( C \) and a composite authorization fragment: if \( C \) holds, then \( F_A \) is permitted.
- after \( f \) then \( F_A \) represents the temporal sequence of fragments. Informally, after \( f \) has happened, then the composite authorization fragment \( F_A \) is permitted.

Also, we assume that (after \( f \) then \( F_A \))) \((\text{after } f \text{ then } F_A)\) behaves as after \( f \) then \((F_A; F_A)\). Finally, round brackets indicate precedences among fragments.
Figure 1: Operational semantics of $F_A$, where the symmetric rule for $;$ is omitted.

Figure 1 shows the operational semantics of $F_A$ in terms of a modified modal transition system $MTS_{A\text{auth}} = (\mathcal{A\text{UTH}}, \mathcal{F}, \rightarrow^{\circ}, \mathcal{C})$. As usual, rules are expressed in terms of a set of premises, possibly empty (above the line) and a conclusion (below the line).

$MTS_{A\text{auth}}$ deals with authorized transitions only, and it considers also the set of contexts because the DSA transitions may depend also on the value of such contexts, see rule if in Figure 1. The introduction of $C (= \{a \text{ set of predicates}\}$ in a labelled transition system is a standard practice [2]. We observe that the if operator implies the binding of variable appearing in the context $C$.

Obligation Fragment.
Similarly, the syntax of a composite obligation fragment is inductively defined as follows:

$F_O := \text{nil} | \text{must } f | F_O \& C \text{ then } F_O \text{ | after } f \text{ then } F_O \text{ | } (F_O)$

The intuition for the composite obligation fragment is the following:

- $\text{nil}$ expresses no obligation.
- $\text{must } f$ is the atomic obligation fragment. Its meaning is the subject $s$ must perform action $a$ on the object $o$, with optional value $v$. Thus, the $f$-transition is required.
- $F_O; F_O$ represents a list of composite obligation fragments. The list constitutes the obligation section of the considered DSA. Whenever one term of the list performs a $f$-transition, then that term evolves to the correspondent derivative.
- if $C$ then $F_O$ expresses the logical implication between a context $C$ and a composite obligation fragment. It means that if $C$ holds, then $F_O$ is required.
- after $f$ then $F_O$ represents the temporal sequence of fragments. It means that after that $f$ is performed, then $F_O$ is required.
- ROUND brackets denote precedence.

The operational semantics of $F_O$ is expressed in terms of the modal transition system $MTS_{OBL} = (\mathcal{OBL}, \mathcal{F}, \rightarrow^{\circ}, \mathcal{C})$. The axioms and rules are similar to the ones presented for $F_A$, apart from changing the transition relation, that becomes $\rightarrow^{\circ}$. For example, the axiom for the atomic obligation fragment is: (must) $\text{must } f \rightarrow^{\circ} \text{ nil}$

3. SOME EXAMPLES OF PATTERNS

Here, we introduce some illustrative examples of authorizations and obligations, and we show their CNL4DSA representation. In the first example we consider the case in which Alice and Bob are members of the same community portal, and they agree on sharing some data:

Alice must send the list of her new contacts to Bob, every time that Bob logs into the portal.

We can express this using CNL4DSA as:

\[ \text{AFTER } f1 \text{ THEN MUST } f2 \]

- $f1 = \{\text{Bob logs into the portal}\}$ is an atomic fragment with subject “Bob” and action $a = \text{“logs into the portal”}$;
- $f2 = \{\text{Alice sends the list of her new contacts to Bob}\}$ is an atomic fragment with subject “Alice”, action “send to Bob”, and object “the list of her new contacts”.

In the second example we consider the case in which Alice is registering as a user of the on-line travel agent NiceTravel, and she wants to specify how NiceTravel can share her preferences:

NiceTravel can share Alice’s preferences after one year of receiving them.

We can express this using CNL4DSA as:

\[ \text{AFTER } f1[v] \text{ THEN } (\text{IF } c1 \text{ THEN CAN } f2) \]

- $f1[v] = \{\text{NiceTravel receives preferences from Alice at } v\}$ is an atomic fragment with action $a = \text{“receives from Alice”}$, subject $s = \text{“NiceTravel”}$, and object $o = \text{“preferences”}$; the variable $v$ is a marker characterizing the occurrence of $f1$;
- $c1 = \{\text{after } 1 \text{ year of } v\}$ is a context;
- $f2 = \{\text{NiceTravel send preferences to ANY}\}$ is an atomic fragment with subject $s = \text{“NiceTravel”}$, action $a = \text{“send to ANY”}$, and object $o = \text{“preferences”}$; we observe that ANY is a predefined identifier representing any subject.

Finally, we consider an example involving a so called “responsive forwarding” statement [22, 23]. Here Bob is the customer of the on-line bookstore ReadMore, and he wants to specify obligations concerning his purchases and the way that ReadMore communicates with Bob’s credit card company SecureCard:

ReadMore will delete the credit card details received from Bob, after two years of storing them. Also, ReadMore will send to SecureCard each purchase order that it receives from Bob.

We can express these statements with a list of $F_O$. 

3In [18, 17], the axioms defining the behaviour of $\square$ are also given in terms of $\rightarrow^{\circ}$, stating that all the required transitions are also admissible.
AFTER \( f1 \) THEN (AFTER \( f2 \) THEN (IF \( c1 \) THEN MUST \( f3 \)))

\[ P_1 \text{par} A \rightarrow P_2 \]

\( P_1 \text{par} A = P_2 \) represents concurrent activity requiring synchronization between the policies \( P_1 \) and \( P_2 \). In particular, any action belonging to the set of actions \( A \) can only occur when both \( P_1 \) and \( P_2 \) permit that action.

\( Z \) is the constant process used for identifying processes. Indeed, we write \( Z \) \( \rightarrow \) \( P \) to use the identifier \( Z \) to refer to the process \( P \).

As usual for (process) description languages, derived operators may be defined. For instance, by using the constant definition, the sequence and the parallel operators, the replicate operator \( \text{r}(P) \) can be derived. Informally, \( \text{r}(P) \) is the parallel composition of \( P \) an unbounded number of times. With \( Z = P\text{par} Z \), \( \text{r}(P) = Z \).

In POLPA, a process \( P \) expresses the admissible behaviour of a system. Obligations are explicitly identified by using special start and end tags: \( \text{Ob} = \text{start}_{\text{Ob}}, P, \text{end}_{\text{Ob}} \). These tags are only syntactic and they serve for distinguishing obligation and authorization processes.

We sketch a possible mapping from CNL4DSA to POLPA for the Authorization Fragment. Let \( M \) be the mapping function, then \( M \) is inductively defined as follows:

\[ M(\text{can}fr) \Rightarrow \alpha(\vec{x}).\text{stop}; \]
\[ M(\text{FrA} \text{FrA}) \Rightarrow M(\text{FrA})\text{par}_B M(\text{FrA}) \]
\[ M(\text{if} \ C \text{ then} \ P \text{A}) \Rightarrow p(\vec{x}).M(\text{FrA}) \]
\[ M(\text{after fr then} \ P \text{A}) \Rightarrow \alpha(\vec{x}).M(\text{FrA}) \]

The replicate operator may simulate the persistence of the authorizations during the DSA validity period. If an authorization says: “a webmail service can archive my read email after 11:00 PM”, then the context “after 11:00 PM” must be evaluated every day during the DSA validity period.

The mapping function for the Obligation Fragment is essentially the same as the one for the Authorization Fragment, with the exception that the mapping of the composite \( FrO_b \) is enclosed into the \( \text{start}_{\text{Ob}}, \text{end}_{\text{Ob}} \) tags.

Each atomic context \( C \) corresponds to a predicate \( p(\vec{x}) \), expressing attributes either of location, or of time, or of roles, etc... Contexts, as well as predicates, are combined by logical connectives with exactly the same logical meaning.

The atomic fragment \( fr \) corresponds to the action \( \alpha(\vec{x}) \), with parameters \( \vec{x} \) representing the subject, the object, and other optional parameters. Hence, the fragments’ set given by the ontology corresponds to the actions’ set \( \text{Act} \) in POLPA.

## 5. RELATED WORK

Here we compare our proposal with existing frameworks. [4] investigates platform-independent policy frameworks to specify, analyze, and deploy security and networking policies. In particular the authors describe a scenario-based demo of a portal prototype for usable and effective policy authoring through either natural language or structured lists that manage policies from the specification to the possible enforcement. However, they do not present any formal language for authoring. Here, we describe a language by giving its syntax and semantics. The main goal is an easy mapping into a high level language for analysing and enforcing any agreements we are able to write.

[22, 23] specifically focus on DSA. They model 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. With respect to...
the logical language in [22, 23], our CNL4DSA aims at a more friendly DSA authoring phase, by letting even the non expert users to easily type their DSA in a structured but yet understandable way. Nevertheless, we also take into account the problem of (automated) DSA verification: CNL4DSA translation to a formal language such as POLPA let us leverage existing analysis frameworks like, e.g., Maude [5].

[14] advocates the idea of using controlled English as a Semantic Web language allowing the creation of Semantic Web content in a user-friendly and yet logically precise way. The resulting framework is quite flexible and extensible. CNL4DSA has a more limited scope: rather than providing a comprehensive controlled natural language, we focus on a reduced set of constructs that allows users to define authorizations and obligations statements. On the other side, we observe that due to the specific domain requirements, CNL4DSA expressiveness goes beyond that of [14] in at least the definition of temporal constructs such as \( \text{after } f \text{ then } F_{A/O} \).

Also [8] presents a logic-based policy analysis framework which (i) is expressive, (ii) considers obligations and authorizations, (iii) includes a dynamic system model, and (iv) gives useful diagnostic information. Our language considers authorizations and obligations, although it is not directly thought for their analysis. Instead, the main aim is to help the user to write DSA in a structured way. CNL4DSA plays the role of an authoring user oriented language.

6. CONCLUSIONS

We focused on the definition of CNL4DSA, and on its translation to a process algebra based formalism (POLPA). Compared to other approaches, our work aims at defining a language that is (i) simple and human readable, (ii) sufficiently expressive to specify common authorizations and obligations in an agreement, and (iii) translatable to one (or more) formal language(s) enabling the DSA analysis. This approach is intended to facilitate the development of user-friendly editing tools that can exploit capabilities of background analysis engines. The combination of such tools would allow users to dynamically define DSA ensuring appropriate privacy conditions, and detect/solve conflicts before the actual enforcement of the resulting policies.

We are currently developing an executable specification of POLPA with Maude [5], that offer built-in model checking capabilities for the analysis of DSA clauses. We intend to exploit Maude features to develop a DSA analysis engine that can detect possible conflicts within a DSA. We plan to interface this back-end analysis engine with a user-friendly DSA authoring tool, which will enable end-users to easily write CNL4DSA fragments, and to see possible conflicts detected by the analysis tool. Finally, we also plan to extend the expressiveness of CNL4DSA by adding, e.g., a while construct to deal with context conditions having a duration.

7. ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their helpful comments. We also thank MarcoCasassaMont (HP Labs, Bristol) and Fabio Martinelli (IIT-CNR) for their precious comments and hints.

8. REFERENCES