An Event-B approach to Data Sharing Agreements

Alvaro E. Arenas, Benjamin Aziz, Juan Bicarregui
and Michael D. Wilson

Published version information

Citation: AE Arenas et al. 'An Event-B approach to Data Sharing Agreements.' In 8th International Conference on Integrated Formal Methods (IFM 2010), Nancy, France, 11-14 Oct 2010, (2010): 28-42. Lecture Notes in Computer Science, vol. 6396. Springer, Berlin, Heidelberg.

DOI: 10.1007/978-3-642-16265-7_4

An Event-B Approach to Data Sharing Agreements

Alvaro E. Arenas, Benjamin Aziz, Juan Bicarregui, and Michael D. Wilson

e-Science Centre, STFC Rutherford Appleton Laboratory
Oxfordshire OX11 0QX, U.K.

{ alvaro.arenas, benjamin.aziz, juan.bicarregui, michael.wilson }@stfc.ac.uk

Abstract. A Data Sharing Agreement (DSA) is a contract among two or more principals regulating how they share data. Agreements are usually represented as a set of clauses expressed using the deontic notions of obligation, prohibition and permission. In this paper, we present how to model DSAs using the Event-B specification language. Agreement clauses are modelled as temporal-logic formulas that preserve the intuitive meaning of the deontic operators, and constrain the actions that a principal can execute. We have exploited the ProB animator and model checker in order to verify that a system behaves according to its associated DSA and to validate that principals' actions are in agreement with the DSA clauses.

Key words: Data Sharing Agreements; Formal Analysis; Event-B

1 Introduction

Data sharing is increasingly important in modern organisations. Every organisation requires the regular exchange of data with other organisations. Although the exchange of this data is vital for the successful inter-organisational process, it is often confidential, requiring strict controls on its access and usage. In order to mitigate the risks inherent in sharing data between organisations, Data Sharing Agreements (DSAs) are used to ensure that agreed data policies are enforced across organisations [11].

A DSA is a legal agreement among two or more parties regulating who can access data, when and where, and what they can do with it. DSAs either include the data policies explicitly as clauses, or include existing organisational data policies by reference. DSA clauses includes deontic notions stating permissions for data access and usage, prohibitions on access and usage which constrain these permissions, and obligations that the principles to the agreement must fulfil. DSA can be created between an organisation and each of many collaborators. A single data set may result from multiple sources, so that the clauses in the DSA with each contributor need to be combined together and applied to it. DSAs are represented in natural language with its concomitant ambiguities and potential conflicts, which are exacerbated by DSA combination. Natural language DSA clauses can be enforced by transforming them into executable policy languages [5]. However, support is required to ensure this transformation conveys the intention of the natural language, while avoiding its potential problems. The formal representation and analysis of agreement clauses could provide this support.

This paper presents a formalisation of DSAs using the Event-B specification language [2]. This involves incorporating normative deontic notions (obligations, permissions and prohibitions) into the Event-B modelling process. The main contribution of the paper is the development of an approach to model agreements using the Event-B specification language. The process itself forces the explicit statement of implicit assumptions underlying the natural language which are required in the formal modelling. The contribution includes a method to transform natural-language data-sharing agreement clauses, including deontic notions, into linear temporal logic predicates suitable for verification and validation.

The paper is structured as follows. Section 2 introduces DSAs and their main components. Then, section 3 summarises the Event-B method. Section 4 presents our approach to model contracts in Event-B, and section 5 applies that approach to a DSA for scientific collaborations. Section 6 discusses the verification and validation of contracts in Event-B. Finally, section 7 relates our work with others, and section 8 presents concluding remarks and highlights future work.

2 An Overview of Data Sharing Agreements

To introduce DSAs, we take as an example data sharing in scientific collaborations [4]. Large-scale research facilities such as particle accelerators and synchrotrons are used by teams of scientists from around the world to produce data about the structure of materials which can be used in many ways, including to create new products, change industrial methods or identify new drugs. There are many individuals and organisations involved in these processes who make agreements with each other to share the data from the facilities, within specified time limits, and for use in particular ways.

Data sharing requirements are usually captured by means of collaboration agreements among the partners, typically established during the scientific proposal preparation. They usually contains clauses defining what data will be shared, the delivery/transmission mechanism, the processing and security framework, among others. Following [12], a DSA consists of a definition part and a collection of agreement *Clauses*. The definition part usually includes the list of involved *Principals*; the start and end dates of the agreement; and the list of *Data* covered by the agreement. Three type of clauses are relevant for DSAs: *Authorisation*, *Prohibitions* and *Obligation* clauses. Authorisations indicate that specified roles of principal are authorised to perform actions on the data within constraints of time and location. Prohibitions act as further constraints on the authorisations, prohibiting actions by specific roles at stated times and locations. Obligations indicate that principals, or the underlying infrastructure, are required to perform specified actions following some event, usually within a time period. The DSA will usually contain processes to be followed, or systems to be used to enforce the assertions, and define penalties to be imposed when clauses are breached.

For instance, in the case of scientific collaborations, principals typically involve *investigators* and *co-investigators* from universities or industrial research departments, and a *scientific facility provider*, whose facilities provide the infrastructure for performing scientific experiments. Data generated from experiments is usually held on a *exper-*

imental database, which is exposed via a web service API. The following are examples of DSA clauses, taken from existing DSAs:

1. During the embargo period, access to the experimental data is restricted to the principal investigator and co-investigators.
2. After the embargo period, the experimental data may be accessed by all users.
3. Access to data must be denied to users located in un-secure locations such as the cafeteria.
4. System must notify principal investigator when a public user access experimental data, within 2 days after the access.
5. User must renew use license within 30 days if it has expired before the three year embargo period.

Clauses 1 and 2 correspond to authorisation clauses. Clause 3 is a prohibition. Clauses 4 and 5 are examples of obligation clauses.

2.1 Main components of Data Sharing Agreements

The set-up of DSAs requires technologies such as DSA authoring tools [8], which may include controlled natural language vocabularies to define unambiguously the DSAs conditions and obligations; and translators of DSAs clauses into enforceable policies. Our work aims at providing formal reasoning support to DSA authoring tools such as the one presented in [8], focusing mainly on the clauses part of a DSA.

We represent DSA clauses as guarded actions, where the guard is a predicate characterising environmental conditions, such as time and location, or restrictions for the occurrence of the event, such as "*user is registered*" or "*data belongs to a project*".

Definition 1. (**Action**). *An action is a tuple consisting of three elements* $\langle p, an, d\rangle$*, where* p *is the principal,* an *is an action name, and* d *is the data.*

Action $\langle p, an, d\rangle$ expresses that the principal p performs action name an on the data d. Action names represent atomic permissions, where actions are built from by adding the identity of the principal performing the action name and the data on which the action name is performed. Actions are analogous to *fragments* defined in [8], We assume that actions are taken from a pre-defined list of actions, possibly derived from an ontology. An example of an action is "Alice accesses experimental data", where "Alice" is the principal, "accesses" is the action and "experimental data" is the data.

We are interested in four types of of clauses: permissions, prohibitions, bounded obligations, and obligations. Clauses are usually evaluated within a specific context, which is represented by a predicate characterising environmental conditions such as location and time.

Definition 2. (**Agreement Clause**). *Let* G *be a predicate,* n *an integer denoting a time unit, and* $a = \langle p, an, d\rangle$ *be an action. The syntax of agreement clauses is defined as follows:*

$$\begin{array}{rcl} \mathcal{C} ::= & \mathtt{IF}\ G\ \mathtt{THEN}\ \mathcal{P}(a) & \mid \quad \mathtt{IF}\ G\ \mathtt{THEN}\ \mathcal{F}(a) \\ \mid & \mathtt{IF}\ G\ \mathtt{THEN}\ \mathcal{O}(a) & \mid \quad \mathtt{IF}\ G\ \mathtt{THEN}\ \mathcal{O}_n(a) \end{array}$$

In the following, we provide an intuitive explanation of the clause syntax. A more precise meaning will be given later when representing agreement clauses in the Event-B language. A permission clause is denoted as `IF` G `THEN` $\mathcal{P}(a)$, which indicates that provided the condition G holds, the system may perform action a. A prohibition clause is denoted as `IF` G `THEN` $\mathcal{F}(a)$, which indicates that the system must not perform action a when condition G holds. An obligation clause is denoted as `IF` G `THEN` $\mathcal{O}(a)$, which indicates that provided the condition G holds, the system eventually must perform action a. Finally, a bounded-obligation clause is denoted as `IF` G `THEN` $\mathcal{O}_n(a)$, which indicates that provided the condition G holds, the system must perform action a within n time units.

A data sharing agreement can be defined as follows.

Definition 3. (**Data Sharing Agreement**). *A DSA is a tuple* $\langle Principals,\ Data,\ ActionNames,\ fromTime,\ endTime,\ \mathbb{P}(\mathcal{C})\rangle$.

$Principals$ is the set of principals signing the agreement and abiding by its clauses. $Data$ is the data elements to be shared. $ActionNames$ is a set containing the name of the actions that a party can perform on a data. $fromTime$ and $endTime$ denotes the starting and finishing time of the agreement respectively; this is an abstraction representing the starting and finishing date of the agreement. Finally, $\mathbb{P}(\mathcal{C})$ is the set of clauses of the agreement.

3 Event-B with Obligations

3.1 Introduction to Event-B

Event-B [2] is an extension of Abrial's B method [1] for modelling distributed systems. In Event-B, machines are defined in a context, which has a unique name and is identified by the keyword `CONTEXT`. It includes the following elements: `SETS` defines the sets to be used in the model; `CONSTANTS` declares the constants in the model; and finally, `AXIOMS` defines some restrictions for the sets and includes typing constraints for the constants in the way of set membership.

An Event-B machine is introduced by the `MACHINE` keyword, it has a unique name and includes the following elements. `VARIABLES` represents the variables (state) of the model. `INVARIANT` describes the invariant properties of the variables defined in the clause `VARIABLES`. Typing information and general properties are described in this clause. These properties shall remain true in the whole model and in further refinements. Invariants need to be preserved by the initialisation and events clauses. `INITIALISATION` allows to give initial values to the variables of the system. `EVENTS` cause the state to change by updating the values of the variables as defined by the *generalised substitution* of the event. Events are guarded by a *condition*, which when satisfied implies that the event is permitted to execute by applying its generalised substitution in the current state of the machine.

Event-B also incorporates a refinement methodology, which can be used by software architects to incrementally develop a model of a system starting from the initial most abstract specification and following gradually through layers of detail until the model is close to the implementation.

In Event-B, an event is defined by the syntax: `EVENT` e `WHEN` G `THEN` S `END` , where G is the guard, expressed as a first-order logical formula in the state variables, and S is any number of generalised substitutions, defined by the syntax $S ::= \ x := E(v) \mid x := z : \mid P(z)$. The deterministic substitution, $x := E(v)$, assigns to variable x the value of expression $E(v)$, defined over set of state variables v. In a non-deterministic substitution, $x := z : \mid P(z)$, it is possible to choose non-deterministically local variables, z, that will render the predicate $P(z)$ true. If this is the case, then the substitution, $x := z$, can be applied, otherwise nothing happens.

3.2 Linear Temporal Logic in Event-B

The Event-B Rodin platform[1] has associated tools, such as ProB[2] and Atelier B[3] for expressing and verifying Linear Temporal Logic (LTL) formulae properties of Event-B specifications. LTL formulae are defined based on a set of propositional variables, $p_1, p_2 \ldots$, the logical operators, $\neg$ (not), $\wedge$ (and), $\vee$ (or), $\rightarrow$ (implication), as well as future temporal operators: $\Box$ (always), $\Diamond$ (eventually), $\bigcirc$ (next), $\mathcal{U}$ (until), $\mathcal{W}$ (weak until) and $\mathcal{R}$ (release). It also includes dual operators for modelling the past.

The propositions themselves can be *atomic*, which include predicates on states written as $\{\ldots\}$, and the event enabled predicate (in the case of ProB), written as `enabled`$(\overline{a})$, which states that event $\overline{a}$ is enabled in the current state. They can also be *transition* propositions, such as $[\overline{a}]$, to state that the next event to be executed in the path is $\overline{a}$. Finally, $(P \Rightarrow Q)$ is often written to denote the formula $\Box(P \rightarrow Q)$.

3.3 Modelling Obligations in Event-B

Classical Event-B does not include a notion of obligation. When the guard of an event is true, there is no obligation to perform the event and its execution may be delayed as a result of, for example, interleaving it with other permitted events. The choice of scheduling permitted events is made non-deterministically. In [3], we describe how obligations can be modelled in Event-B as events, interpreting the guard of an event as a *trigger* condition. An obliged event is written as `EVENT` e `WHEN` T `WITHIN` n `NEXT` S `END` , where T is the trigger condition such that when T becomes true, the event must be executed within at most $n + 1$ number of time units, provided the trigger remains true. If the trigger changes to false within that number, the obligation to execute the event is canceled. This type of event represents a bounded version of the leads-to modality, represented by the obligation $(T \ \Rightarrow \ \Diamond_{\leq n}(T \rightarrow e))$. The obliged event is a syntactic sugar and can be encoded and refined in the standard Event-B language (see [3]). In the rest of the paper, we adopt Event-B with obligations as our specification language.

4 Formal Modelling of Data Sharing Agreements in Event-B

This section introduces our method for modelling DSAs in Event-B. The method consists of a number of stages, as shown in Figure 1.

[1] www.event-b.org/platform.html
[2] www.stups.uni-duesseldorf.de/ProB
[3] www.atelierb.eu

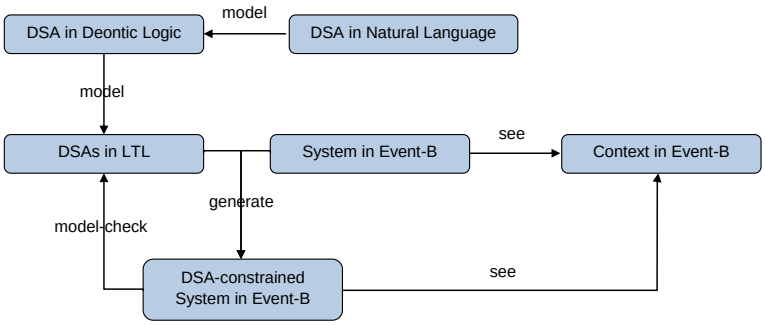

Fig. 1. A Lifecycle for Modelling DSAs in Event-B.

First, the system domain and variables are defined, and actions are identified (see subsections 4.1 and 4.2). Second, once defined the system vocabulary, the agreement clauses are modelled using the deontic operators introduced in definition 2 (see subsection 4.3). Finally, each agreement clause is modelled as a logical formula that holds as an invariant in the Event-B system (see subsections 4.4, 4.5 and 4.6).

4.1 Defining System Domains

The first step in the modelling of DSAs in Event-B consists of defining the domains to be used in the model. This corresponds to the definition of primitive sets and constant values. A DSA includes three main sets: *PRINCIPALS* indicating the principals participating in the DSA; *DATA* indicating the date to be shared, and *PACTIONNAMES* denoting the actions that principals may perform on data. Following definition 1, an action is defined as a tuple consisting of a principal, an action name, and a data. In addition, we are interested in registering the actions performed by the system, therefore we introduce set *SACTIONNAMES* to indicate such actions. The main motive in distinguishing user and system action names stems from our later treatment of obligations where we distinguish between the non-enforceable obligations on users and the enforceable obligations on systems.

Our example follows a role-based model, hence we identify the roles to be used in the system; *LOCATIONS* indicates the set of location of a principal may be located. Below, we present the context for our DSA example. We are using a slightly different syntax to the one used in Event-B for readability purpose, associating each set with its constant elements.

CONTEXT

$PRINCIPALS = \{ Alice, Bob, Charlie, \cdots \}$
$PACTIONNAMES = \{ \texttt{access}, \texttt{renewlicence}, \cdots \}$
$DATA = \{ ExpData, \cdots \}$
$PACTION = PRINCIPALS \times PACTIONNAMES \times DATA$

$SACTIONNAMES = \{ \texttt{notify} \}$
$MESSAGE = PRINCIPALS \times DATA$
$SACTION = SACTIONNAMES \times PRINCIPAL \times MESSAGE$

$LOCATIONS = \{ room1, room2, cafeteria, \cdots \}$
$ROLES = \{ PI, CoI, PublicUser \}$

4.2 Modelling System Variables

The following are the main variables associated to a DSA. $action$ is the input to the system. $actionLog$ represents the actions performed by principals, where each action is labelled with the occurrence time. Those actions performed by the system are logged into the $systemLog$. Variables $fromTime$, $endTime$ and $currentTime$ denote the starting time of the agreement, final time, and the current time respectively. In addition, there are domain-specific variables related to specific DSAs. For our example of DSA in the scientific domain, we require variables such as $embargoTime$, indicating the end of the embargo on the data being shared; $location$ associates each principal with his current location and $safeLocation$ indicates if a location is safe. The relation $roleAssig$ associates principals with their role in the system; and $getPI$ is a function that given a data returns its associated principal investigator. Finally, $licenceExpTime$ indicates the time when the licence of a principal for accessing a data expires. Below, we present the variables of our DSA, their domain, and their main properties.

INVARIANTS

$action \in PACTION \cup SACTION$
$actionLog \in \mathbb{P}(\mathbb{N} \times PACTION)$
$systemLog \in \mathbb{P}(\mathbb{N} \times SACTION)$

$fromTime \in \mathbb{N}$
$endTime \in \mathbb{N}$
$currentTime \in \mathbb{N}$

$location \in PRINCIPALS \rightarrow LOCATIONS$
$safeLocation \in LOCATIONS \rightarrow \mathbb{B}$
$embargoTime \in DATA \rightarrow \mathbb{N}$
$getPI \in DATA \rightarrow PRINCIPALS$
$roleAssig \in (PRINCIPALS \times DATA) \rightarrow ROLES$
$licenceExpTime \in (PRINCIPALS \times DATA) \rightarrow \mathbb{N}$

$fromTime \leq currentTime \leq endTime$
$\forall t \in \texttt{ran}\ licenceExpTime \cdot t \leq endTime$
$\forall d \in DATA \cdot roleAssig(getPI(d), d) = PI$

4.3 Initial Modelling of Agreement Clauses

Having defined system variables enables us to model the guard associated to each clause, and to identify the actions of the system. The next step consists in representing agreement clauses in the style presented in Definition 2. For instance, the clauses described in Section 2 can be modelled as follows.

$$
\begin{array}{l}
\mathcal{C}_1 : \forall p \in PRINCIPALS,\ d \in DATA \cdot \\
\quad \texttt{IF}\ (\ d = ExpData\ \wedge \\
\quad\quad currentTime \leq embargoTime(d)\ \wedge\ roleAssig(p,d) \in \{PI, CoI\}\ ) \\
\quad \texttt{THEN}\ \ \mathcal{P}(\langle p, \texttt{access}, d\rangle) \\
\mathcal{C}_2 : \forall p \in PRINCIPALS,\ d \in DATA \cdot \\
\quad \texttt{IF}\ (\ d = ExpData\ \wedge\ currentTime > embargoTime(d)\ ) \\
\quad \texttt{THEN}\ \ \mathcal{P}(\langle p, \texttt{access}, d\rangle) \\
\mathcal{C}_3 : \forall p \in PRINCIPALS,\ d \in DATA \cdot \\
\quad \texttt{IF}\ \neg\ safeLocation(location(p)) \\
\quad \texttt{THEN}\ \ \mathcal{F}(\langle p, \texttt{access}, d\rangle) \\
\mathcal{C}_4 : \forall p \in PRINCIPALS,\ d \in DATA \cdot \\
\quad \texttt{IF}\ (\langle p, \texttt{access}, d\rangle\ \in\ \texttt{ran}\ (actionLog)\ \wedge\ roleAssig(p,d) = PublicUser) \\
\quad \texttt{THEN}\ \ \mathcal{O}_2(\langle \texttt{notify}, getPI(d), (p,d)\rangle) \\
\mathcal{C}_5 : \forall p \in PRINCIPALS,\ d \in DATA \cdot \\
\quad \texttt{IF}\ currentTime > licenceExpTime(p,d) \\
\quad \texttt{THEN}\ \mathcal{O}_{30}(\langle p, \texttt{renewlicence}, d\rangle)
\end{array}
$$

4.4 Modelling Permission and Prohibition Clauses.

We proceed now to represent agreement clauses in Event-B. As a starting point, we assume that principal actions are modelled in the system as Event-B events. For instance, if $\langle p, a, d\rangle$ is an arbitrary principal action, we assume then there is an event called $\overline{a}$ that models the effect of principal p performing action a on data d as a guarded substitution.

$$
\begin{array}{l}
\texttt{EVENT}\ \overline{a1}\ \texttt{ANY}\ action\ \texttt{WHERE}\ action = \langle p1, a1, d1\rangle\ \texttt{THEN} \\
\quad \cdots\ \ \texttt{/* Execution of action } a1\ \texttt{*/}\ \ \texttt{END} \\
\texttt{EVENT}\ \overline{a2}\ \texttt{ANY}\ action\ \texttt{WHERE}\ action = \langle p2, a2, d2\rangle\ \texttt{THEN} \\
\quad \cdots\ \ \texttt{/* Execution of action } a2\ \texttt{*/}\ \ \texttt{END} \\
\ldots
\end{array}
$$

We represent agreement clauses as assertions that the system must validate. Let $\langle p, a, d\rangle$ be an arbitrary principal action and $\overline{a}$ be its associated Event-B event. Let $\texttt{IF}\ G\ \texttt{THEN}\ \mathcal{P}(\langle p, a, d\rangle)$ be an arbitrary permitted clause. The following assertion must be valid in the system, indicating that the event associated to the clause may be executed when condition G holds:

$$
(\ (action = \langle p, a, d\rangle\ \wedge\ G)\ \Rightarrow\ \texttt{enabled}(\overline{a})\ )
$$

where `enabled` is the event-B enable proposition, indicating that an event is enabled in the current state.

Let $\texttt{IF}\ G\ \texttt{THEN}\ \mathcal{F}(\langle p, a, d\rangle)$ be an arbitrary prohibition clause. The following assertion must be valid in the system, indicating that the event associated to the clause must not be executed when condition G holds:

$$((action = \langle p, a, d\rangle \ \wedge \ G) \ \Rightarrow \ \neg \, \texttt{enabled}(\overline{a}))$$

In order to validate agreement clauses, some changes are needed in the structure of the events associated to actions. First, the permitted clauses indicate when an action may be executed, then the guard of the associated actions should be strengthened with the disjunction of the clauses conditions. Second, the prohibition clauses indicate when an action must not be executed, then the guard of the associated actions is strengthened with the conjunction of the negation of the clauses. Formally, let $\langle p, a, d\rangle$ be an arbitrary principal action and $\overline{a}$ be its associated event. Let $\texttt{IF}\ G_i\ \texttt{THEN}\ \mathcal{P}(\langle p_i, a, d_i\rangle)$, for $i = 1, \cdots, k$ be all the permitted clauses associated to action a, and let $\texttt{IF}\ G'_j\ \texttt{THEN}\ \mathcal{F}(\langle p_j, a, d_j\rangle)$, for $j = 1, \cdots, l$ be all the prohibition clauses associated to action a. The event is transformed as follows to meet the agreement clauses.

$$\begin{array}{l}\texttt{EVENT}\ \overline{a}\ \texttt{ANY}\ action\ \texttt{WHERE}\ action = \langle p, a, d\rangle \ \wedge \ \bigvee_{i=1}^{k} G_i \ \wedge \bigwedge_{j=1}^{l} \neg G'_j\ \texttt{THEN} \\ \quad \cdots \ \| \ actionLog := actionLog \cup \{currentTime \mapsto (p \mapsto a \mapsto d)\} \\ \texttt{END}\end{array}$$

4.5 Modelling Obligations on the System.

Without loss of generality, we will consider only bounded-obligation clauses. Any unbounded obligation in a DSA can be transformed into a bounded one by limiting it by the duration of the contract. Let $\langle b, c, d\rangle$ be an arbitrary system action; and $\mathcal{C} : \ \texttt{IF}\ G\ \texttt{THEN}\ \mathcal{O}_k(\langle b, c, d\rangle)$ be an arbitrary obligation clause. This imposes an obligation on the system expressed by the following temporal logic formula.

$$\begin{array}{l}(G \ \wedge \ currentTime < clock_{\mathcal{C}} + k) \ \Rightarrow \\ \quad \Diamond(\ (\neg G \ \vee \ (currentTime \mapsto (b \mapsto c \mapsto d)) \ \in \ systemLog) \ \wedge \\ \quad\quad currentTime \leq clock_{\mathcal{C}} + k \ )\end{array}$$

In above LTL formula, variable $clock_{\mathcal{C}}$ indicates the time when trigger condition G becomes true. The formula expresses that condition G holds for at most k time units, until a new action $\langle b, c, d\rangle$ is registered in the $systemLog$. This is indeed the intuitive meaning of the obligation clause expressed using deontic operators. The above obligation could be enforced by adding a new event performing the associated change in the system state, as described in subsection 5.4.

4.6 Modelling Obligations on Users.

Let $\langle p, a, d\rangle$ be a principal action and $\mathcal{C} : \ \texttt{IF}\ G\ \texttt{THEN}\ \mathcal{O}_k(\langle p, a, d\rangle)$ be an arbitrary obliged clause. In general, the system cannot enforce users' obligations, but it can detect when a user's obligation has not been fulfilled, as illustrated by the formula below.

$$\begin{array}{l}(G \ \wedge \ currentTime < clock_{\mathcal{C}} + k) \ \Rightarrow \\ \quad \Diamond(\ (\neg G \ \vee \ (currentTime \mapsto (p \mapsto a \mapsto d)) \ \in \ actionLog) \ \wedge \\ \quad\quad currentTime \leq clock_{\mathcal{C}} + k \ )\end{array}$$

5 An Example of a DSA in Event-B

This section applies the method presented previously to our DSA example for scientific collaborations. The first two steps, defining system domain and variables and expressing the agreement clauses using deontic operators, were presented in section 4. We continue with the modelling of the clauses as logic formulae.

5.1 The Agreement Clauses as Logic Formulae.

Below we model the clauses introduced in subsection 4.3 as logic formulae, following the patterns presented in the previous section. Permission and prohibition clauses are modelled in predicate logic, and can be represented as invariants of the system. Obligations are modelled in LTL, and can be represented as theorems that the system can verify using model-checking techniques or validate via simulators. We assume that all clauses are universally quantified with the variable $p \in PRINCIPALS$ and $d \in DATA$.

$$
\begin{array}{l}
\mathcal{C}_1 : (action = \langle p, \texttt{access}, d\rangle \ \wedge \ \ d = ExpData \ \wedge \\
\qquad currentTime \leq embargoTime(d) \ \wedge \ roleAssig(p,d) \in \{PI, CoI\}\ ) \ \Rightarrow \\
\qquad\quad \texttt{enabled}(\overline{\texttt{access}}) \\
\mathcal{C}_2 : (action = \langle p, \texttt{access}, d\rangle \ \wedge \ \ d = ExpData \ \wedge \ currentTime > embargoTime(d)) \ \Rightarrow \\
\qquad\quad \texttt{enabled}(\overline{\texttt{access}}) \\
\mathcal{C}_3 : (action = \langle p, \texttt{access}, d\rangle \ \wedge \ \neg\, safeLocation(location(p))\ ) \ \Rightarrow \\
\qquad\quad \neg\texttt{enabled}(\overline{\texttt{access}}) \\
\mathcal{C}_4 : (\ (clock_{\mathcal{C}_4} \mapsto (p \mapsto \texttt{access} \mapsto d)) \in actionLong\ ) \wedge \\
\qquad roleAssig(p,d) = PublicUser \ \wedge \ currentTime < clock_{\mathcal{C}_4} + 2) \ \ \Rightarrow \\
\qquad \Diamond(\ (currentTime \mapsto (\texttt{notify} \mapsto getPI(d) \mapsto (p \mapsto d))) \in systemLog \ \ \wedge \\
\qquad\quad currentTime \leq clock_{\mathcal{C}_4} + 2) \\
\mathcal{C}_5 : (currentTime > licenceExpTime(p,d) \ \wedge \ currentTime < clock_{\mathcal{C}_5} + 30) \ \ \Rightarrow \\
\qquad \Diamond(\ (currentTime \mapsto (p \mapsto \texttt{renewlicence} \mapsto d)) \in actionLog\ ) \ \ \wedge \\
\qquad\quad currentTime \leq clock_{\mathcal{C}_5} + 30
\end{array}
$$

5.2 Initialising the System

For completeness sake, we include here the initialisation of the system. The DSA is assumed to have a duration of 100 time units.

INITIALISATION

$$
\begin{array}{l}
actionLog, systemLog := \varnothing, \varnothing \\
fromTime, endTime, currentTime := 1, 100, 1 \\
location := \{\ Alice \mapsto room1,\ Bob \mapsto room2,\ Charlie \mapsto cafeteria\ \} \\
safeLocation := \{\ room1 \mapsto TRUE,\ room2 \mapsto TRUE,\ cafeteria \mapsto FALSE\ \} \\
embargoTime := \{\ ExpData \mapsto 60\ \} \\
getPI := \{\ ExpData \mapsto Alice\ \} \\
roleAssig := \{\ (Alice, ExpData) \mapsto PI,\ (Bob, ExpData) \mapsto CoI, \\
\qquad\qquad (Charlie, ExpData) \mapsto PublicUser\ \} \\
licenceExpTime := \{\ (Alice, ExpData) \mapsto 50,\ (Bob, ExpData) \mapsto 30, \\
\qquad\qquad\qquad (Charlie, ExpData) \mapsto 30\ \}
\end{array}
$$

5.3 User Actions as Event-B Events

The system includes two events: $\overline{\texttt{access}}$, indicating that a principal will access a data, and $\overline{\texttt{renewlicence}}$, indicating that a principal has renewed the licence to access a data.

In the case of $\overline{\texttt{access}}$, the event guard is strengthened with the disjunction of the guards of its associated permitted clauses (clauses $\mathcal{C}_1$ and $\mathcal{C}_2$ in subsection 4.3) and the conjunction of the negation of the forbidden clauses (clause $\mathcal{C}_3$).

EVENT $\overline{\texttt{access}}$ ANY $action$ WHERE
$action = \langle p, \texttt{access}, d \rangle \ \wedge \ d = ExpData \wedge$
$(\ (currentTime \leq embargoTime(d) \ \wedge \ (roleAssig(p, d) \in \{PI, CoI\})) \vee$
$currentTime > embargoTime(d) \ ) \wedge$
$safeLocation(location(p))$ THEN
$actionLog := actionLog \cup \{currentTime \mapsto (p \mapsto \texttt{access} \mapsto d)\}$
END

For $\overline{\texttt{renewlicence}}$, we introduce a new constant $RENEWTIME$ that indicates the constant time a licence is increased once the event occurs.

EVENT $\overline{\texttt{renewlicence}}$ ANY $action$ WHERE
$action = \langle p, \texttt{renewlicence}, d \rangle$ THEN
$licenceExpTime(p, d) := currentTime + RENEWTIME \ \ \|$
$actionLog := actionLog \cup \{currentTime \mapsto (p \mapsto \texttt{renewlicence} \mapsto d)\}$
END

5.4 System Obligations as Events

The system is obliged to notify the principal investigator when a data item is accessed by a public user. This is modelled as the obligated Event-B event $\overline{\texttt{notify}}$, which must be executed within 2 time units after the guard holds.

EVENT $\overline{\texttt{notify}}$ WHEN
$((p \mapsto \texttt{download} \mapsto d) \in \texttt{ran}\,(actionLog) \ \wedge \ roleAssig(p, d) = PublicUser)$
WITHIN 2 NEXT
$systemLog := systemLog \cup \{currentTime \mapsto (\texttt{notify} \mapsto getPI(d) \mapsto (p, d))\}$
END

5.5 User Obligations as Events

As mentioned before, user obligations cannot be enforced by the system, although they can be detected. Below, we show the $\overline{\texttt{checklicence}}$ event, which checks if a principal has renewed a licence for accessing a data within 30 time units after expiring the license. If the licence is not renewed, the principal is *blacklisted* for using such data. To model such action, we assume the existence of a predicate $blacklist : PRINCIPALS \times DATA \rightarrow \mathbb{B}$, which is initialisated as false for any principal and

any data. We also assume that the $RENEWTIME$ constant is greater that 30 time units. The system designer could, for instance, use the blacklist information to restrict access to the associated data.

```
EVENT checklicence WHEN
  (currentTime > licenceExpTime(p, d)) WITHIN 30 NEXT
        blacklist(p, d) := (currentTime > licenceExpTime(p, d))
END
```

5.6 Dealing with the Environment

In our model of the DSAs in Event-B, we assume that the environment is modelled through time and location. Our notion of time can based on some standard represetation (e.g. ISO 8601), which could include a combined date and time representation. We assume, for simplicity, that *currentTime* is a natural number and that every event in the machine representing an action will perform its own time incrementation. Hence, the event representing action $\overline{a}$ will become as follows, assuming that the event lasts for only one time unit,

```
EVENT ā1 WHEN ... THEN
    ··· ‖ currentTime := currentTime + 1 END
```

The other environment variable is location, where we have assumed the second solution, i.e. a separate event for changing locations of users, which has overrides the current value of the *location* function with a new location for some principal.

6 Formal Verification and Validation of DSA Properties

6.1 Verifying DSA Properties

The minimum global property that must hold for any DSA is that all permissions, prohibitions and system obligations stated in the DSA clauses must hold true.

One example of common conflicts is when an event corresponding to a principal action is both enabled (permitted) and disabled (prohibited) in the same state of the machine. Verifying that the model is free from this sort of conflicts corresponds to model-checking the following LTL formulae:

$$\Box\neg(\texttt{enabled}(\overline{a}) \wedge \neg(\texttt{enabled}(\overline{a})))$$

Another example of conflicts is when obligations are not permitted. Hence, for example, a user obligation of the sort $\texttt{IF}\ G\ \texttt{THEN}\ \mathcal{O}_n(\langle p, a, d\rangle)$ must be permitted,

$$\forall t \in \mathbb{N} \cdot (G \wedge currentTime = t) \Rightarrow \neg((\neg\texttt{enabled}(\overline{a}))\ \mathcal{U}\ (currentTime > t + n))$$

which means that the obliged event $\overline{a}$ must have been enabled at some stage prior to the current time passing the deadline $t + n$.

Healthiness properties are desirable aspects of the DSA that are not expressed by any of the previous properties we mentioned above. For example, healthiness property of our DSA for scientific collaborations state that all accesses to experimental data are performed by principals with a valid licence. This would correspond to the following formula, which was verified in our system.

$$\forall p \in \text{PRINCIPALS}, d \in \text{DATA}, n \in \mathbb{N} : \\ (n, (p, \texttt{access}, d)) \in actionLog \;\Rightarrow\; n \leq \textit{licenceExpTime}(p, d)$$

Other possible healthiness properties would be: to check that no accesses occur at unsafe locations; to prove that system-notification events are idempotent; and to verify that any penalties associated with obligation violation (both for systems and users) are properly enforced. For example, in the case of violating user obligations, that the associated capabilities (certain events) are disabled if the user has not fulfilled their obligation.

6.2 Validating DSA Properties

The validation of the example DSA was carried out using the animation capabilities of the Pro-B plug-in for Rodin. This method of validation has a number of advantages for revealing problems with the specification, mainly:

- It helps monitor every value of the state variables as the machine executes each event. This may reveal certain under-specifications of the types of variables. For example, in one instance, it was discovered that sets are not sufficient to model logs (both action and system logs), since sets do not distinguish different instances of the same actions. Hence, timestamps were added to achieve that effect.
- It helps to view which of the events of the machine are currently active. In the case of modelling user obligations, this is quite helpful since it can reveal whether the obligation conditions are strong enough to disable other events in the case where the obliged event has not yet been executed. More generally, this gives an idea as to whether the activation of events is as expected or not.
- Simulating the machine allows understanding better traces that violate invariants, for those invariants that cannot be verified. For example, in the case of the obligations on users to renew the licence when it expires, these cannot be enforced. Hence, there are runs of the machine in which the user will not renew the licence. Therefore, simulation is beneficial since it will demonstrate the effects of not fulfilling such obligations.

7 Related Work

There have been other attempts to model and analyse contracts using event/state-based modelling approaches. [10] presents how standard conventional contracts can be described by means of Finite State Machines (FSMs). Rights and obligations extracted

from the clauses of the contract are mapped into the states, transition and output functions, and input and output symbols of a FSM. The FSM representation is then used to guarantee that the clauses stipulated in the contract are observed when the contract is executed. The approach is more directed to contract monitoring than analysis, since no formal reasoning is included.

In [6], Daskalopulu discusses the use of Petri Nets for contract state tracking, and assessing contract performance. Her approach is best suited for contracts which can naturally be expressed as protocols, or workflows. She model-checks a contract to verify desired properties. Our work has been inspired by hers, but it differs in that we separate the modelling of the contract – declarative approach by representing clauses as temporal-logic predicates – from the modelling of the system. Following similar objectives, [7] uses the Event Calculus to represent a contract in terms of how its state evolves according to a narrative of (contract-related) events.

An initial model of DSAs is proposed in [12]. The model is based on dataflow graphs whose nodes are principals with local stores, and whose edges are channels along which data flows. Agreement clauses are modelled as obligation constraints expressed as distributed temporal logic predicates over data stores and data flows. Although the model is formal, they do not exploit formal reasoning about agreements.

In [9], the authors propose an event language with deontic concepts like permissions, rights and obligations, which are relevant to the modelling of DSAs. However, their main focus is the access control framework, whereas we focus on a more general framework related to clauses that may occur in any DSA.

Our work has been influenced by the work by Matteucci *et al* [8], which presents an authoring tool for expressing DSAs in a controlled natural language. One of our aims is to provide formal reasoning support for such a tool. The underlying semantic model for DSAs in [8] is an operational semantics expressing how a system evolves when executed under the restriction of an agreement. Within this view, agreement clauses are encoded within the system. We consider that for formal reasoning, it is important to separate the agreement clauses from the system functionality, hence we have proposed to represent clauses as LTL assertions that the system must respect.

8 Conclusion and Future Work

This paper presents an approach for modelling contracts using the Event-B specification language. This involves incorporating normative deontic notions (obligations, permissions and prohibitions) into the Event-B modelling process. We have focused on one particular type of contract, the so-called data sharing agreements (DSAs).

The starting point of the proposed method is an informal DSA. In order to formalise it, the following steps are proposed. First, the system domain and variables are defined, and actions are identified. Second, agreements clauses are modelled using deontic logic. Third, each deontic-logic clause is represented in linear temporal logic (LTL), and the Event-B event dealing with the action is transformed so that the LTL predicate is valid in the model. The relation between the LTL clause and the Event-B model is established by applying verification (model-checking) and validation (animation) techniques.

DSAs typically follow a lifecycle comprising the following stages: (1) *contract drafting*, which includes the drafting of contracts with the aid of authoring tools; (2) *contract analysis*, which includes the formalisation and analysis of contracts in order to detect potential conflicts among contract clauses; (3) *policy derivation* from the contract clauses; and (4) finally, *monitoring and enforcement* of contract policies. This paper has concentrated on the second stage, contract analysis. As future work, we plan to study formally the policy derivation process. In addition, we will investigate the problem of agreement evolution (changes in an agreement), and whether those changes can be verified/maintained using refinement techniques.

Acknowledgment

This work was partly supported by the EU FP7 project Consequence (Context-Aware Data-Centric Information Sharing), project grant 214859.

References

1. J-R. Abrial. *The B Book*. Cambridge University Press, 1996.
2. J-R. Abrial and S. Hallerstede. Refinement, Decomposition, and Instantiation of Discrete Models: Application to Event-B. *Fundamenta Informaticae*, 77(1-2):1–28, 2007.
3. J. Bicarregui, A. E. Arenas, B. Aziz, P. Massonet, and C. Ponsard. Toward Modelling Obligations in Event-B. In *International Conference of ASM, B and Z Users*, volume 5238 of *Lecture Notes in Computer Science*, pages 181–194. Springer, 2008.
4. S. Crompton, B. Aziz, and M. D. Wilson. Sharing Scientific Data: Scenarios and Challenges. In *W3C Workshop on Access Control Application Scenarios*, 2009.
5. N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The Ponder Policy Specification Language. In *Policies for Distributed Systems and Networks*, volume 1995 of *Lecture Notes in Computer Science*, pages 18–38. Springer, 2001.
6. A. Daskalopulu. Model Checking Contractual Protocols. In *Legal Knowledge and Information Systems*, Frontiers in Artificial Intelligence and Applications Series, 2001.
7. A. D. H. Farrell, M. J. Sergot, M. Sallé, and C. Bartolini. Using the Event Calculus for Tracking the Normative State of Contracts. *International Journal of Cooperative Information Systems*, 14(2-3):99–129, 2005.
8. I. Matteucci, M. Petrocchi, and M. L. Sbodio. CNL4DSA a Controlled Natural Language for Data Sharing Agreements. In *25th Symposium on Applied Computing, Privacy on the Web Track*. ACM, 2010.
9. D. Méry and S. Merz. Event Systems and Access Control. In D. Gollmann and J. Jürjens, editors, *6th Intl. Workshop Issues in the Theory of Security*, pages 40–54, Vienna, Austria, 2006. IFIP WG 1.7, Vienna University of Technology.
10. C. Molina-Jimenez, S. Shrivastava, E. Solaiman, and J. Warne. Run-Time Monitoring and Enforcement of Electronic Contracts. *Electronic Commerce Research and Applications*, 3(2):108–125, 2004.
11. J. E. Sieber. Data Sharing: Defining Problems and Seeking Solutions. *Law and Human Behaviour*, 12(2):199–206, 1988.
12. V. Swarup, L. Seligman, and A. Rosenthal. A Data Sharing Agreement Framework. In *ICISS 2006, Second International Conference on Information Systems Security*, volume 4332 of *Lecture Notes in Computer Science*, pages 22–36. Springer, 2006.

Appendix: Event-B Model of the Example DSA

CONTEXT Context

The context of the contract example in the paper

SETS

`PRINCIPALS` The set of principals

`PACTIONNAMES` The set of user action names

`DATA` The set of data

`SACTIONNAMES` The set of system action names

`LOCATIONS` The set of locations

`ROLES` The set of roles

CONSTANTS

`Access`
`Renewlicence`
`Alice`
`Charlie`
`ExpData`
`notify`
`Room1`
`Room2`
`Cafeteria`
`PI`
`COI`
`PublicUser`

AXIOMS

`axm1` : $partition(PACTIONNAMES, \{Access\}, \{Renewlicence\})$

`axm2` : $partition(PRINCIPALS, \{Alice\}, \{Charlie\})$

`axm3` : $partition(DATA, \{ExpData\})$

`axm4` : $partition(SACTIONNAMES, \{notify\})$

`axm5` : $partition(LOCATIONS, \{Room1\}, \{Room2\}, \{Cafeteria\})$

`axm6` : $partition(ROLES, \{PI\}, \{COI\}, \{PublicUser\})$

END

MACHINE Contract

A machine modelling the contract example in the paper

SEES Context

VARIABLES

`actionLog` The log registering all the user actions

`systemLog` The log registering all the system actions

`currentTime` The current time of the machine

`fromTime` The time of the start of the contract

`endTime` The time of the end of the contract

`embargoTime` The embargo time on a dataset

`roleAssig` a function mapping principals to their role with respect to some dataset

`location` a function giving the location of a principal

`getPI` a function giving the PI of a dataset

`safeLocation` a function indicating whether a location is safe or not

`licenceExpTime` a function giving the time at which the licence for a user expires with respect to some dataset

`notifycounter` a counter for the notify event

`clocknotify` a clock for the notify event to register the first time its condition becomes true

`blackList` a function denoting whether a user is blacklisted with respect to a dataset

`checklicencecounter` a counter for the checklicence event

INVARIANTS

`inv4` : $actionLog \in \mathbb{P}(\mathbb{N} \times PRINCIPALS \times PACTIONNAMES \times DATA)$

`inv5` : $systemLog \in \mathbb{P}((\mathbb{N} \times SACTIONNAMES \times PRINCIPALS) \times (\mathbb{N} \times PRINCIPALS \times PACTIONNAMES \times DATA))$

`inv6` : $currentTime \in \mathbb{N}$

`inv12` : $fromTime \in \mathbb{N}$

`inv13` : $endTime \in \mathbb{N}$

`inv7` : $embargoTime \in DATA \rightarrow \mathbb{N}$

`inv8` : $roleAssig \in (PRINCIPALS \times DATA) \rightarrow ROLES$

`inv9` : $location \in PRINCIPALS \rightarrow LOCATIONS$

`inv11` : $getPI \in DATA \rightarrow PRINCIPALS$

`inv14` : $fromTime \leq currentTime \land currentTime \leq endTime$

`inv15` : $safeLocation \in LOCATIONS \rightarrow BOOL$

`inv16` : $licenceExpTime \in (PRINCIPALS \times DATA) \rightarrow \mathbb{N}$

`inv17` : $\forall t \cdot t \in ran(licenceExpTime) \Rightarrow t \leq endTime$

`inv18` : $\forall d \cdot d \in DATA \land (\exists p \cdot ((d \mapsto p) \in getPI) \Rightarrow ((p \mapsto d) \mapsto PI) \in roleAssig)$

`inv21` : $notifycounter \in \mathbb{N}$

`inv23` : $0 \leq notifycounter \land notifycounter \leq 3$

`inv24` : $clocknotify \in \mathbb{N}$

`inv27` : $blackList \in (PRINCIPALS \times DATA) \rightarrow BOOL$

`inv28` : $checklicencecounter \in \mathbb{N}$

`inv29` : $0 \leq checklicencecounter \land checklicencecounter \leq 5$

EVENTS

Initialisation

The initialisation event

begin

act4 : $actionLog := \varnothing$

act5 : $systemLog := \varnothing$

act6 : $currentTime := 1$

act11 : $fromTime := 1$

act12 : $endTime := 100$

act7 : $embargoTime := \{ExpData \mapsto 3\}$

act8 : $roleAssig := \{(Alice \mapsto ExpData) \mapsto PI, (Charlie \mapsto ExpData) \mapsto PublicUser\}$

act9 : $location := \{Alice \mapsto Room1, Charlie \mapsto Cafeteria\}$

act10 : $getPI := \{ExpData \mapsto Alice\}$

act13 : $safeLocation := \{Room1 \mapsto TRUE, Room2 \mapsto TRUE, Cafeteria \mapsto FALSE\}$

act14 : $licenceExpTime := \{(Alice \mapsto ExpData) \mapsto 15, (Charlie \mapsto ExpData) \mapsto 10\}$

act17 : $notifycounter := 3$

act18 : $clocknotify := 0$

act19 : $blackList := \{(Alice \mapsto ExpData \mapsto FALSE), (Charlie \mapsto ExpData \mapsto FALSE)\}$

act20 : $checklicencecounter := 5$

end

Event $access \mathrel{\widehat{=}}$

An event for allowing users to access datasets

any

$action$

p

d

where

grd1 : $action = (p \mapsto Access \mapsto d) \land (blackList(p \mapsto d) = FALSE) \land (safeLocation(location(p)) = TRUE) \land ((currentTime \leq embargoTime(d) \land roleAssig(p \mapsto d) \in \{PI, COI\}) \lor (currentTime > embargoTime(d)))$

grd7 : $(\neg(\exists t1, q, e \cdot ((t1 \mapsto q \mapsto Access \mapsto e) \in actionLog) \land (roleAssig(q \mapsto e) = PublicUser) \land (\neg(t1 \mapsto q \mapsto Access \mapsto e) \in ran(systemLog)))) \lor (notifycounter > 1)$

(not T1 or counter1¿ 0)

grd8 : $(\neg(currentTime > licenceExpTime(p \mapsto d))) \lor (checklicencecounter > 1)$

(not T2 or counter2¿ 1)

then

act1 : $actionLog := actionLog \cup \{currentTime \mapsto p \mapsto Access \mapsto d\}$

act2 : $notifycounter : |\exists q, e, t1 \cdot (((t1 \mapsto q \mapsto Access \mapsto e) \in actionLog) \land (roleAssig(q \mapsto e) = PublicUser) \land (\neg(t1 \mapsto q \mapsto Access \mapsto e) \in ran(systemLog)) \land (notifycounter' = notifycounter - 1)) \lor (\neg(\exists p, d, t2 \cdot ((t2 \mapsto p \mapsto Access \mapsto d) \in actionLog) \land (roleAssig(p \mapsto d) = PublicUser) \land$

$(\neg(t2 \mapsto p \mapsto Access \mapsto d) \in ran(systemLog))) \land (notifycounter' = 3))$

act3 : $clocknotify : |((((roleAssig(p \mapsto d) = PublicUser)) \land (clocknotify = 0)) \land (clocknotify' = currentTime)) \lor (\neg(((roleAssig(p \mapsto d) = PublicUser)) \land (clocknotify = 0)) \land (clocknotify' = clocknotify))$

act4 : $currentTime := currentTime + 1$

act5 : $checklicencecounter : |\exists q, e \cdot ((currentTime > licenceExpTime(q \mapsto e)) \land (checklicencecounter' = checklicencecounter - 1)) \lor ((\neg(\exists r, f \cdot (currentTime > licenceExpTime(r \mapsto f)))) \land (checklicencecounter' = 5))$

end

Event *notify* $\widehat{=}$

An event for notifying PIs when public users access datasets

any

p

d

$t1$

z

where

grd1 : $((t1 \mapsto p \mapsto Access \mapsto d) \in actionLog) \land (roleAssig(p \mapsto d) = PublicUser) \land (\neg(t1 \mapsto p \mapsto Access \mapsto d) \in ran(systemLog))$

grd2 : $(\neg(currentTime > licenceExpTime(p \mapsto d))) \lor (checklicencecounter > 1)$

not T2 or counter2¿ 1

grd3 : $(currentTime > licenceExpTime(p \mapsto d)) \Rightarrow (z = (checklicencecounter - 1))$

grd4 : $(\neg(currentTime > licenceExpTime(p \mapsto d))) \Rightarrow (z = 5)$

then

act5 : $systemLog := systemLog \cup \{((currentTime \mapsto notify \mapsto getPI(d)) \mapsto (t1 \mapsto p \mapsto Access \mapsto d))\}$

act2 : $notifycounter := 3$

act3 : $clocknotify := 0$

act4 : $currentTime := currentTime + 1$

act6 : $checklicencecounter := z$

end

Event *locationchange* $\widehat{=}$

An environment event to change the locations of users

any

p

l

where

grd2 : $p \in PRINCIPALS$

grd1 : $location(p) \neq l$

then

act1 : $location(p) := l$

act2 : $currentTime := currentTime + 1$

end

Event *renewlicence* $\widehat{=}$

An event to renew the data licences

any
$action$
p
$renewtime$
d
where
grd1 : $p \in PRINCIPALS$
grd2 : $action \in PRINCIPALS \times PACTIONNAMES \times DATA$
grd3 : $renewtime \in \mathbb{N}$
grd4 : $action = (p \mapsto Renewlicence \mapsto d)$
grd5 : $(\neg(\exists t1, q, e \cdot ((t1 \mapsto q \mapsto Access \mapsto e) \in actionLog) \wedge (roleAssig(q \mapsto e) = PublicUser) \wedge (\neg(t1 \mapsto q \mapsto Access \mapsto e) \in ran(systemLog)))) \vee (notifycounter > 1)$
(not T1 or counter1¿ 1)
grd6 : $(\neg(currentTime > licenceExpTime(p \mapsto d))) \vee (checklicencecounter > 1)$
(not T2 or counter2¿ 1)
then
act1 : $licenceExpTime(p \mapsto ExpData) := currentTime + renewtime$
act2 : $actionLog := actionLog \cup \{currentTime \mapsto p \mapsto Renewlicence \mapsto ExpData\}$
act3 : $notifycounter : |\exists q, e \cdot ((((currentTime - 1) \mapsto q \mapsto Access \mapsto e) \in actionLog \wedge (roleAssig(q \mapsto e) = PublicUser)) \wedge (notifycounter' = notifycounter - 1)) \vee (\neg(((currentTime - 1) \mapsto q \mapsto Access \mapsto e) \in actionLog \wedge (roleAssig(q \mapsto e) = PublicUser)) \wedge (notifycounter' = 3))$
act4 : $clocknotify : |\exists w, f \cdot (((((currentTime - 1) \mapsto w \mapsto Access \mapsto f) \in actionLog \wedge (roleAssig(w \mapsto f) = PublicUser)) \wedge (clocknotify = 0)) \wedge (clocknotify' = currentTime)) \vee (\neg((((currentTime - 1) \mapsto w \mapsto Access \mapsto f) \in actionLog \wedge (roleAssig(w \mapsto f) = PublicUser)) \wedge (clocknotify = 0)) \wedge (clocknotify' = clocknotify))$
act5 : $currentTime := currentTime + 1$
end
Event *checklicence* $\widehat{=}$
An event to blacklist users with expired licences
any
p
d
z
where
grd1 : $currentTime > licenceExpTime(p \mapsto d)$
grd2 : $(\neg(\exists t1, q, e \cdot ((t1 \mapsto q \mapsto Access \mapsto e) \in actionLog) \wedge (roleAssig(q \mapsto e) = PublicUser) \wedge (\neg(t1 \mapsto q \mapsto Access \mapsto e) \in ran(systemLog)))) \vee (notifycounter > 1)$
(not T1 or counter1¿ 0)
grd3 : $((\exists t \cdot (t \mapsto p \mapsto Access \mapsto d) \in actionLog \wedge (roleAssig(p \mapsto d) = PublicUser))) \Rightarrow z = (notifycounter - 1)$

grd4 : $(\neg(\exists y \cdot (y \mapsto p \mapsto Access \mapsto d) \in actionLog \wedge (roleAssig(p \mapsto d) = PublicUser))) \Rightarrow z = 2$

then

act1 : $blackList(p \mapsto d) := TRUE$

act2 : $checklicencecounter := 5$

act3 : $notifycounter := z$

act4 : $currentTime := currentTime + 1$

end

END