Deontic logic for modelling data flow and use compliance

David Evans
Computer Laboratory
University of Cambridge
Cambridge CB3 0FD, UK
{firstname.lastname}@cl.cam.ac.uk

David M. Eyers
Computer Laboratory
University of Cambridge
Cambridge CB3 0FD, UK
{firstname.lastname}@cl.cam.ac.uk

ABSTRACT
We advocate using deontic logic and its representation in the Event Calculus to control access to information in a distributed ubiquitous system. Contracts between information owners are encoded in terms of classes of organisations, data, and interactions. Fluids, events, and application-specific rules that link the two are then extracted from the contracts and mapped to the components, endpoints, and messages used to implement the system. The expression of organisations’ responsibilities is natural and leads to a simple mechanism of data flow monitoring. Some parts of the system can make forward progress while others are in conflict, meaning that resolution does not impede other processing. Furthermore, specification in terms of entities’ behaviour rather than explicit modelling of service level agreements (SLAs) means that it is straightforward to make decisions based on observations that are not specified in the SLA but that are noticed by a human as being abnormal.

Categories and Subject Descriptors
H.3.4 [Information Storage and Retrieval]: Systems and Software—Distributed systems; I.2.4 [Artificial Intelligence]: Knowledge Representation Formalisms and Methods—Relation systems; H.1.1 [Models and Principles]: Systems and Information Theory—Value of information

General Terms
Management, Security

Keywords
Middleware, contract, deontic logic, event calculus

1. INTRODUCTION
Pervasive applications work by providing services to users based on data in the surrounding world. These data may come from a model of a particular user’s behaviour, be collected and aggregated by the user’s social contacts, be based on knowledge from the application itself (such as a lighting system using the output from proximity sensors), or may be provided by other sources outside the application.

A given organisation often is best placed to measure a particular thing. For example, the local bus company may periodically collect the location of each of its buses to assist with route changes and capacity planning, taxi companies know the location of each of their vehicles at all times to aid in dispatch, and pollution levels may be recorded by the appropriate part of local government. In each of these cases the data have an owner, the entity deciding who may receive the data and how they may be used.

Clearly ubiquitous applications can be far more sophisticated if they use the widest range of data possible and for this they will require data from multiple owners. For owners to be comfortable with making data assets available, they must be confident that there will be no misuse. We believe that monitoring compliance with owners’ wishes should be a function of the middleware used to build distributed ubiquitous applications; making applications responsible for compliance checking will lead to a range of incompatible approaches, and a flexible middleware service allows applications and data providers freedom to use functionality that is as strong or as weak as they like.

We advocate deontic logic and its expression in the Event Calculus as the basis for data use rules and the monitoring of subsequent compliance. The result is a natural way to describe the restrictions specified in a data usage contract and methods to detect suspected violations and reason about state change (e.g., to answer questions like “may I send data Y to X?”) are straightforward. It is also easy to incorporate manual triggers, such as when a human suspects contractual violation. There is a tight coupling between the operation of the system and the legal documents that detail compliance. This leads to simple behaviour. Managing contractual compliance explicitly within a ubiquitous system aims to restrict to those parties involved the impact of situations that are perceived to be contractual failures. Given that the system infrastructure may be shared by many parties and applications, it is important that the whole system is not adversely affected by a conflict that arises between some subset of those parties.

Our work in this area is focused on a project to construct a Transport Information Monitoring Environment (TIME) [5]. Data regarding movement of pedestrians, buses, cyclists, vehicles, trains, and so on (as well as their effects, such as pollution) are collected by many different organisations having disparate responsibilities and loyalties. Our goal is to de-
sign and build the middleware which can be used to develop useful ubiquitous applications that make use of these data. This means that organisations and individual users must be able to contribute information with the confidence that it won’t be misused by applications.

This paper is organised as follows. Section 2 provides an overview of deontic logic and its representation in the Event Calculus. Section 3 presents the architecture of the TIME system, which is our testbed. Section 4 describes how contracts are encoded and the software components needed for compliance monitoring. Section 5 describes an example contract and Section 6 concludes and outlines future work.

2. DEONTIC LOGIC, EVENT CALCULUS, AND CONTRACT REPRESENTATION

This section provides some background to deontic logic, knowledge representation using Kowalski’s Event Calculus, and contract encoding. We attempt to set each area in the context of related research work.

2.1 Deontic Logic

Typical forms of classical predicate logic are unable to represent modalities—notions of possibility and necessity—in an elegant manner. Modal logics extend classical logic with modal operators. For example, it is common to provide the unary modal operators $\Box$ for necessarily and $\Diamond$ for possibly. Depending on the specific logic, axioms that relate the two, such as $\Box A \iff \neg \Diamond \neg A$ (“possibly $A$ is equivalent to not necessarily not $A$”) may hold.

Deontic logic is a modal logic that focuses on representations of obligation and prohibition, including ideas such as social norms and legal interpretation. The logic has a rich publication record, an overview of which is available in Meyer and Wieringa’s book [12]. We use deontic logic to represent the clauses of the contracts that dictate data management; non-modal logics are not appropriate given that contracts by their nature discuss possibility and necessity. Furthermore, logically inconsistent states of affairs may represent contract violations but do not preclude rational forward progress between participants that are not party to the contract in dispute.

The key deontic concepts are obligation, prohibition, violation, and annulment: They have the following meanings:

**Obligation** Some set of actions needs to be performed in the future to progress the state of affairs.

**Prohibition** This means that there is an obligation not to perform some set of actions.

**Violation** Violations occur when an obligation to not do something is broken, or an obligation to do something is not done within the required time-frame.

**Annulment** An annulment cancels out the effect of some other predicate (possibly another annulment). Annulments cover both the notion of exception (a document clause may override the effect of another clause) and the notion of satisfaction (where completion of an obligatory task annuls the obligation to do that task).

We also speak of permissions, which are essentially named annulments. The instantiation of a permission within a document is likely to be coupled with the explicit annulment of clauses that would otherwise be in conflict with it. For example, a permission might annul the obligations to do or not do actions specified elsewhere in that document. Moreover, the naming of permissions will allow them to be used as external reference points from other documents’ clauses.

2.2 Event Calculus

The Event Calculus is a straightforward mechanism for reasoning about changes in states of affairs [10]. A subset, known as the Simplified Event Calculus, provides the functionality that we need here; we shall use the term Event Calculus when technically we mean Simplified Event Calculus. The Event Calculus is particularly natural to implement in the Prolog language and a basic implementation is shown in Figure 1.

To model a particular system with the Event Calculus, the core Event Calculus rules are augmented with application-specific event and fluent definitions and application-specific rules that link the two. Events are named, instantaneous effects that are accompanied with a representation of their time of occurrence. It is usually useful to augment events with parameters per instance; Prolog encodings facilitate this trivially. Fluents are named, half-open intervals in the same time domain as events. A fluent describes a state of affairs that holds over that time period. Using intervals that are closed at the beginning and open at the end (so that an interval from $t_1$ to $t_2$ would be written $[t_1, t_2)$) ensures sensible semantics for fluent composition and for boundary cases such as simultaneous initiation and termination of a fluent (in such cases the fluent is not considered to hold at all). As for events, it will frequently be convenient for applications to parametrise fluents. Finally, application-specific rules define the relationship between events and the initiation and termination of fluents.

In Figure 1, initially(U) means that fluent $U$ initially holds. holds_at(U, T) means that fluent $U$ holds at time $T$, while holds_for(U, T1, T2) means that fluent $U$ holds for the time interval $[T_1, T_2)$. clipped(U, T1, T2) means that at some time between times $T_1$ and $T_2$, fluent $U$ ceases to hold. happens(E, T) means that event $E$ takes place at time $T$. Instances of the initiates/3 and terminates/3 clauses (where /3 indicates, in standard Prolog style, the number of ordered parameters in a functional term) effect the application-specific rules that link events and fluents. In the simplest cases, fluent initiation and termination will be based on Prolog facts; particular events will have a direct impact on particular fluents.

Predictably, the core semantics of the Event Calculus do not depend on absolute notions of time, merely on the order of related events (which will be a partial order over all events in the system). For simplicity and stability, integers are often used to represent time. This avoids potential confusion over comparison of values given computers’ lack of floating-point precision. Tuning appropriate behaviour for simultaneous events will be an application-specific concern.

2.3 Event Calculus Extensions

As clauses of legal documents are linked with particular fluents, further logic must combine the states of those fluents to determine the overall deontic state. A similar problem was tackled in Grosol, Labrou, and Chan’s representation of contracts using Situated Courteous Logic Programs [9]. However, our logic programming representation is formalised
holds_at(U,T) :- 0 =< T, initially(U), \+ clipped(U,0,T).
holds_at(U,T) :- happens(E,Ts), Ts =< T, initiates(E,U,Ts), \+ clipped(U,Ts,T).
holds_for(U,T1,T2) :- holds_at(U,T1), \+ clipped(U,T1,T2).
clipped(U,T1,T2) :- happens(E,T), T1 =< T, T < T2, terminates(E,U,T).

Figure 1: A basic Prolog implementation of the Simplified Event Calculus.

d_holds_at(U,T) :-
    holds_at(U,T),
    \+ d_annulled_at(U,T).

d_annulled_at(U1,T) :-
    annulled_by(U1,U2),
    d_holds_at(U2,T).

Figure 2: An implementation of deontic fluent inference.

using the Event Calculus and represents deontic states explicitly.

The core Event Calculus predicates allow deductive reasoning as to the states of affairs that hold at a particular point in time. However, the states described in Section 2.1 alone are not directly useful because complementary states relating to a particular clause (such as an obligation and its satisfaction) are maintained independently on the basis that their evidence is independent. For this reason, we apply an extra level of deontic inference to the core deontic fluents that hold at any point in time. Informally, a state is considered to deontically hold if it holds and it is not annulled by an annulment that is not itself annulled (and so on recursively). The Prolog encoding of this is shown in Figure 2, in which the T variable represents time and the U variables are fluents. The annulled_by/2 predicate is an application-specific rule and indicates whether one fluent would annul another.

3. THE TIME MODEL

The TIME project exemplifies the type of pervasive application described in the introduction. Data are collected from multiple sources, are processed in ways that may or may not be envisaged by the data owners, and are presented to users in useful ways.

The TIME world is shown schematically in Figure 3. The basic entity is the component. A component is responsible for performing a set of functions that should be related. Components communicate by exchanging messages and this is how all data exchange is effected. Messages emanate from and are received by endpoints; each endpoint is plugged into one or more others. An endpoint specifies the schema of the messages that it will emit and accept. The framework enforces matching of sender and receiver schemas, ensuring that only compatible endpoints are connected. (A system for polymorphic endpoints facilitates writing components that don’t know ahead of time the schemas of the messages that they will produce or consume.)

Each endpoint comes with a list of roles required to connect to it. Role certificates are issued by components via a special endpoint in response to the presentation of credentials, or through administrator action. Connections between endpoints use link-level encryption that is similar to TLS. Further details are described by Bacon et al. [5].

Multiple components may be under the control of a single entity; for example, a city may have one component that manages traffic signals while another monitors levels of pollution. Such an entity is named an organisation and is our unit of data ownership. Data in a message does not have to be owned by the organisation encompassing the component that emitted the message. Since components communicate only by exchanging messages, that is how organisations interact. This means that monitoring the message flow between components is sufficient to keep track of organisations’ compliance with policy.

4. CONTRACT LABELLING AND REQUISITE COMPONENTS

Many systems have to track compliance against service level agreements (SLAs). Natural language legal clauses are usually first rendered into some sort of technical-level concern, such as an access control list, a particular workflow, or bounds on network latency, bandwidth, and so on. There is of course a vast body of research and implementation work on managing processes within organisations. Computational approaches are often related to workflow systems (for an overview of workflow standards see Schmidt [13]). In terms of policy representation, research has been done on the representation of deontic state using Petri Nets [6, 7, 11] and
Finite State Machines [7, 8]. Both of these technologies impose limitations on the action sequences by which policy compliance can be achieved.

We want to preserve as direct a correlation as possible between the source legal documents and the operation of the system. Therefore we avoid explicit policy encodings that lead to particular workflow specifications, instead focusing on representations that track the constraints implied by policy. The document annotation that we perform leads to named deontic fluents and links particular clauses to them; the approach has been previously described by Abrahams and Eyers [2]

The authors have been involved with previous work done on user-focused frameworks that are designed to assist users to comply with policy, in particular the two contract representation frameworks, EDEE [1, 3] and CamPACE [4]. Both of these systems take a bottom-up approach, trying to determine overall contract status from very fine grained annotation of policy clauses. Our approach here is instead based on a top-down decomposition of the contractual clauses by document region into particular sections of logic programming. The internal semantics of each such region will not necessarily be completely specified.

Recall that our goal is to monitor the exchange of messages between components and flag (or prevent) those that are inconsistent with the contracts between the organisations that own the components. In order for this to work, contracts must be encoded so that they are meaningful to the deontic logic engine. This begins with labelling the relevant parts of the contracts, after which deontic fluents may be constructed that correspond to the clauses in the contract.

Labelling involves construction of classes of organisations, data, and interactions. This is done as follows:

1. Organisation classes are defined according to behaviour that is expected and useful. Examples of such classes include “us”, “our partners who have signed an NDA”, and “our customers.”

2. The types of data that might be exchanged are classified, such as “proprietary data”, “data that we are willing to give to anyone”, and so on.

3. Classes of interactions, and thus data exchange, are constructed.

These classes are then used to construct events and fluents corresponding to the actions and states of compliance specified in the contract.

We monitor the flow of messages passing between organisations as follows. A special component—the deontic manager—is placed in each organisation. This component is responsible for representing the organisation to others; an organisation is not required to exchange messages with another with which it has no business relationship. Organisations and their deontic managers become known to each other through some external mechanism that is beyond the scope of this work.

The deontic manager is responsible for issuing role certificates for components within its organisation and those outside. This is done based on the types of message the endpoint wishes to exchange (which is encoded in the endpoint’s schema), how those data are classified in the encoded contract between the organisations, and the class of the component requesting the certificate. Once in operation, the deontic manager is interposed between all inter-organisation endpoint connections. (Standard techniques for achieving scalability may be used to ensure that the deontic manager is not a performance bottleneck.) For the moment we expect the deontic manager to use a Prolog-like implementation of the Event Calculus to monitor message transmissions and, based on classifications and application-specific rules, transmit a notification message of its own when violation is suspected. In the future, the deontic manager might take pro-active action such as discarding messages or effecting automatic conflict resolution.

In some cases, it may be appropriate to deploy software that can trigger activation of deontic fluents automatically. For example, the fluents corresponding to the sub-clause “information has been disclosed to a third party” might be able to be monitored by observation of the interactions between classes of endpoints and their encompassing organisations. But this is by no means a necessity: fluents such as the ones linked with “affiliates were not legally bound to uphold confidentiality to the maximum practical degree” may be activated by events that are caused by human operators and the framework allows these manual triggers to sit comfortably beside automatic monitoring. An appropriate user interface would map real-world, observable concerns to the events that are sent to appropriate deontic managers. Regardless of how the deontic fluents are activated, we suggest that in many cases some further operator attention will be required before extreme actions such as suspension or termination of a data sharing agreement are taken. Any process, manual or automatic, that activates deontic fluents will need to have evidence to hand to support the initiation of any such fluents.

Compared to contract monitoring systems based on workflow, our approach facilitates expressive inference over combinations (e.g., recursive annulment) of active fluents. This allows for management of exceptions—as raised by either party—at any time for which the contract holds, without leading to the combinatorial explosions of states that would result in FSM and Petri Net approaches (or incorporation of ad hoc extensions in either to support exceptions).

In terms of conflict resolution, even if contractual clauses are not encoded at a great level of detail, it may still be possible for an automatic process to eliminate many routes to annulment of a particular fluent. For example, the attention of an operator for a party accused of breaching confidentiality can be focused on areas of the contract that might potentially annul the claimed contractual violation.

5. EXAMPLE CONTRACT

The University of Cambridge entered into a contract with a local company regarding sharing of data describing, in real time, bus locations. More information is available on request, but the specific contractual details are omitted here since the contract is not a public document. In this case there were only two parties involved: the University of Cambridge (the research project involved has no incorporated organisation to allow specification at a finer level of granularity) and the bus company.

The contract contains clauses such as the following.

5.1 Undertakings (Clause 2)
This section of the contract indicates ideal behaviour; the inverse is a violation trigger that either party may invoke. In the nomenclature of the Event Calculus, we use such triggers to define events that may then affect fluents describing the current state of compliance. Such events are usually defined in terms of classes of interactions, organisations, and data, but they may also be caused manually through an appropriate user interface.

There are three particular violation triggers in this section:

- Use of information was made that is not deemed to be in the spirit of the agreement.
- Confidential information was not actually kept confidential. This trigger has a specific sub-clause: that information has been disclosed to a third party. Note that clause 4, described in Section 5.3, clarifies notions of confidentiality.
- Any of the obligations mentioned in clause 5 (Section 5.4) are not met.

This clause also defines a class of data (confidential information) and a class of organisation (third party) and they may be used to parameterise events and fluents.

5.2 Exceptions (Clause 3)

The notions discussed in the “exceptions” section lead to the definition of fluents that have an effect on whether other fluents deontically hold. In other words, the exceptions behave as overrides and are described as annulling fluents related to clause 2.

- There was a legal necessity or other proper purpose that led to a requirement for information disclosure.
- The party to which information appears to have leaked had the information already.
- The data transmitted was received free of any obligation from some third party (who did not acquire it directly from the accused party).
- The leaked information had ended up being made public through no fault of the accused party.

Here we see the definition of a class of interactions (legal necessity).

5.3 Confidentiality Measures (Clause 4)

This section elaborates on the notion of “violation of confidentiality” as used in this contract. Trigger events could initiate fluents related to the following problems:

- The confidential material was not kept separate from other materials.
- The confidential material was not protected to the same extent as other confidential information owned by the accused organisation.
- Confidential information is externally available from the accused organisation.
- Confidential information was exposed to an organisation’s affiliate.
- Affiliate organisations did not have reasonable need to see the confidential information or were not informed of the confidential nature of information they were issued with.
- Affiliates were not legally bound to uphold confidentiality to the maximum practical degree.
- Discovery of the unauthorised use of confidential information did not lead to the other party being notified as quickly as possible.

Here, fluents representing violations are explicitly defined and imply definitions of organisation classes such as external and affiliate and data classes such as owned-by-us.

5.4 Purpose Being Ended (Clause 5)

Clause five discusses how to wrap up the data interchange agreement between the two parties. Predictably, events that indicate termination of the contract activate two obligations:

- The University of Cambridge must destroy all confidential material owned by the bus company (relevant to the contract).
- The bus company must destroy all of the University of Cambridge’s confidential materials (relevant to the contract).

6. CONCLUSIONS

We have advocated using deontic logic, expressed in the Event Calculus, to represent data use contracts and we have outlined how this encoding might be done with reference to a sample agreement between a local bus company and the University of Cambridge. Furthermore, we have described the software components that are necessary to incorporate such encodings of policy into distributed pervasive applications and have a test system—the TIME framework—within which we can build them. The natural expression of a contract’s clauses and what results leads to a tight coupling between the requirements of the contracts and the flows of data that the system permits. This direct relationship increases observers’ confidence in the system’s ability to monitor and enforce data use policy, thereby encouraging owners to make their data available. The resulting cooperation between data owners and application developers allows for more complex and rich pervasive applications, ultimately increasing the value to users.

Our plans for the future include the following. We will implement the deontic manager component and complete the encoding of the contract that has been described. These will be done in parallel as we expect the details of the application-specific rules that the contract requires to inform the structure and function of the deontic manager. The types of human intervention that the contract suggests (such as signalling a conflict regarding use of data outside the system) determine the best user interface elements for operators to influence the system’s behaviour.

At the moment, our design relies on examining every message, passing its classification through the rule system to determine whether delivery should be allowed. We intend to conduct a study of the impact of this on performance. We expect that on modern hardware the negative effects may be small considering the improvement in compliance.
monitoring that results. However, for data streams that involve a rapid succession of messages, the overhead of checking each one may be problematic. Another approach is to statistically sample the messages coming from each endpoint with the goal of checking a representative subset for contractual compliance. Taken to the extreme, compliance is only checked when endpoints are connected. How well this works, and how to design the sampling strategy so as to form an accurate picture of the interactions taking place, is an area for future work.

Finally, we can extend our scheme to maintain data provenance by recording classes of data, endpoints, and organizations at organisation boundaries. Stored and queried properly, this can form the basis for not only data pedigree but for reasoning about a datum’s history of compliance (derived by placing the datum within the context of the contracts that were in place at the time of its transmission). We intend to determine how this may best be done.

7. REFERENCES


