State needed to infer data use compliance in distributed transport applications

David Evans and David M. Eyers

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

Abstract. Pervasive transport applications will benefit from sharing underlying communication infrastructure and using data that are owned by different parties. In the past we have suggested using deontic logic and the Event Calculus to control access to information in distributed transport applications. Here we argue that monitoring compliance with policy entails separating two types of state, namely, a subset of the application state and the inferred explicit assertions of the apparent state of compliance. The tight coupling of such assertions to the clauses of the natural language policy documents that form the agreements between organisations mean that the organisations agree on the assertions' meanings. This approach (i) reduces what must be maintained, avoiding state explosion; (ii) provides unambiguous points of interaction between organisations—the assertions of apparent compliance; and (iii) allows organisations to use whatever mechanisms they prefer to infer compliance but hides these internals behind the unambiguous assertions. We have a prototype implementation of these ideas as part of middleware designed to share transport information.

1 Introduction

Transport applications provide services to users based on data from a variety of sources. For example, the local bus company may note periodically the locations of its buses to assist with route changes and capacity planning, taxi companies use the location of each of their vehicles to aid in dispatch, and pollution levels may be recorded by the appropriate part of local government to influence planning decisions. Data typically have owners and non-trivial applications will use the widest range of data possible, meaning that they must process information from multiple owners. Such applications are impossible to build if organisations manage data entirely within isolated, vertical silos.

To encourage contribution of transport data, owners must be confident that the data will not be misused. We believe that software can help and that monitoring compliance with data use agreements should be a function of the middleware used to build applications. Making applications responsible will lead to a range of incompatible approaches. Our work in this area is focused on a project to

construct middleware allowing transport data to be collected by many different organisations, having disparate responsibilities and loyalties, and using the data to develop useful applications [1]. We have advocated deontic logic—a modal logic that focuses on representations of obligation and prohibition—and its expression in the Event Calculus as the basis for data use rules and the monitoring of subsequent compliance [2].

The key contributions of this paper include the observation that one must maintain information about the current state of affairs in terms of application semantics and, as a form of indirection, use this to infer the current degree of compliance. These are separate types of state and lead to two types of fluents in the Event Calculus sense. We illustrate that this division is effective using an example scenario.

This paper is organised as follows. Section 2 provides background, including an overview of our middleware's architecture. Section 3 describes our approach, including the responsibilities of the deontic manager component and the types of state required to monitor compliance. Section 4 illustrates an example deployment and section 5 provides concluding remarks and outlines future work.

2 Background

Many systems have to track compliance against service level agreements, and frequently this has been in the context of workflow systems (for an overview of workflow standards see Schmidt [3]). In terms of policy specification, research has been done on the representation of deontic state using Petri Nets [4–6] and Finite State Machines [5]. In these approaches, application state *is* the state of compliance. This leads inevitably to a combinatorial state explosion.

We use the Event Calculus (more specifically the Simplified Event Calculus) to reason about changes in the states of affairs [7]. A summary is included in our previous work [2], along with a comparison between our approach and those mentioned above. Contract representation with the Event Calculus is itself not a new idea. It has been used, for example, to reason about the kinds of contract that a composite software service may accept [8], but it is unclear that these schemes are effective at reducing the size of the state space.

It is worth defining some terminology. 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* that emanate from and are received by *endpoints*; each endpoint is plugged into one or more other endpoints. Each endpoint specifies the schema of the messages that it will emit and accept. The middleware enforces matching of sender and receiver schemas, ensuring that only compatible endpoints are connected. Multiple components may be under the control of a single entity. Such an entity is named an *organisation* and is our unit of data ownership. Monitoring the message flow between components is sufficient to keep track of organisations' compliance with policy.

3 Monitoring Compliance: State and the Deontic Manager

A contract defines states of compliance, specifies happenings that can affect these, and describes actions that should be taken as a consequence. We represent compliance by fluents in an Event Calculus rendition of the contract and monitor it using a special proxy component called the *deontic manager*. One is placed in each organisation and is interposed between all inter-organisation endpoint connections.

The deontic manager examines for each message it receives (i) the type of message in question (which is given by the message's schema) and how such messages are classified in the encoded contract between the organisations, (ii) the classes of the sending and receiving endpoints, and (iii) the classes of the sending and receiving components' enclosing organisations. Classes are expressed as part of the encoded contract following the pattern described in our previous work [2]. The deontic manager then activates and terminates fluents that represent the application-specific state of the system. This state in turn is used to estimate the current degree of compliance, again by activating and terminating fluents—"deontic" fluents—derived from the contract. Once this is done, the deontic manager forwards the message to its intended recipient or discards it. Notification messages corresponding to deontic fluents may be transmitted by the deontic manager itself.

This straightforward split between application-specific and deontic fluents allows the latter to be unambiguous "hooks" for monitoring compliance. The owners of the data necessarily reach consensus on their rights and responsibilities as codified in the agreements that they sign. Since deontic fluents are coupled tightly with these agreements, the meanings of such fluents are consistent between the organisations involved. Once these have been defined, each party is free to trigger them using any means that suits the local environment. The application-specific fluents are defined so that they do just that.

4 Example

In this section we describe an example, which we have implemented, where this scheme is used. The main organisations involved are the local bus operator, a local taxi company, and ourselves. The bus operator has agreed to provide us, using the shared middleware, with streams of data that indicate the locations of their buses. They have given permission for us to interpret these data in order to provide a public service that helps people know when they need to reach their local bus stops in order to catch the next bus. Although this service is available to the public, the bus position data that it uses are not. In fact, these data are confidential: they provide a large amount of information considered to be commercially sensitive to bus operations and so our example centres on the bus operator determining whether or not they consider a breach to have occurred.

The middleware also carries data about the locations of taxis. The taxi company provides a service, via the middleware, allowing one to determine the distance from a given spot to the nearest taxi. This is useful to their potential customers as it allows estimation of waiting times. It is important to realise that the bus company also has access to this service.

In our hypothetical scenario, we suppose that the bus operator decides to evaluate the efficacy of adding more bus routes. They estimate the coverage of the city of buses and taxis in part by finding the proximity of taxis to their buses. To their surprise, it appears that taxis are almost invariably trailing a short distance behind each of their buses. Staff presume that there is a commercial interest in taxis doing this: potential bus passengers who realise that they have just missed a bus will be more likely to jump into a taxi than wait for the next bus. The bus company suspects that the taxi operator has been able to discover, using the shared middleware, information about the positions of their buses. The spotlight falls on us: regardless of whether it is intentional, information may be leaking beyond that produced by our bus stop arrival application.

Here the fluents needed to represent application-specific state include BUS LOCATION and TAXI LOCATION, parametrised with an identifier and the location of the bus or taxi, respectively. These may be combined to yield the application-specific fluent TAXI NEAR BUS. While this fluent does not have any inherent deontic interpretation, in the context of this scenario it can be used to infer the suspicion of a violation. This state of compliance can be reflected by the deontic fluent VIOLATION SUSPECTED. This fluent might elicit a severe automated response, *e.g.*, the bus company could terminate deontic fluents that need to hold for data interactions, thereby halting our feed of bus position data and breaking our bus-stop notification application.

The indirection from application-specific to deontic fluents allows the bus company to monitor, using private rules internal to them, the correlation between positions of taxis and their own buses. They must agree with us on the meaning of the VIOLATION SUSPECTED fluent but can keep the details of the others, including the evidence that is used to trigger them, private. Disclosure will be necessary only to the extent that conflict resolution requires it. Throughout the period of conflict between the bus company and us, however, the underlying infrastructure continues to function, allowing data to travel between the bus company and other, independent subscribers.

5 Conclusions and Future Work

We have described, as part of using deontic logic expressed in the Event Calculus to represent data use contracts, how application-specific state must be monitored and used to infer the degree of compliance—the deontic state. This means that while two types of state must be maintained, combinatorial state explosion can be avoided because the scope of relevant application-specific state is limited. We have illustrated this using an example of automatic monitoring that binds the two to yield suspicion of contract violation.

Our plans for the future include the following. At the moment, our design relies on examining every message, using its classification to effect state changes and determine whether delivery should be allowed. We intend to conduct a study of the impact of this process on performance, using, for example, the methodology applied to Houdini as a guide [9]. While we expect that on modern hardware the negative effects may be small considering the improvement in compliance monitoring that results, for data streams that involve a rapid succession of messages the overhead of checking each one may be problematic. We also intend to examine sampling messages with the goal of checking a representative subset for contractual compliance. Taken to the extreme, compliance is only checked when endpoints are connected. At the moment it is unclear how to design the sampling strategy so as to form an accurate picture of the interactions taking place and how to reason about resulting probabilistic compliance.

We can extend our scheme to maintain data provenance if deontic managers record the information that they use to guide state changes. 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.

References

1. Bacon, J., Beresford, A.R., Evans, D., Ingram, D., Trigoni, N., Guitton, A., Skordylis, A.: TIME: An open platform for capturing, processing and delivering transport-related data. In: Proceedings of the IEEE consumer communications and networking conference. (2008) 687–691
2. Evans, D., Eyers, D.M.: Deontic logic for modelling data flow and use compliance. In: MPAC '08: Proceedings of the 6th international workshop on middleware for pervasive and ad-hoc computing, New York, NY, USA, ACM (2008) 19–24
3. Schmidt, M.T.: The evolution of workflow standards. IEEE Concurrency (1999)
4. Bons, R.W., Lee, R.M., Wagenaar, R.W., Wrigley, C.D.: Modelling inter-organizational trade procedures using documentary Petri nets. In: Proceedings of the Hawaii international conference on system sciences. (1995)
5. Daskalopulu, A.: Logic-based tools for the analysis and representation of legal contracts. PhD thesis, Department of Computing, Imperial College, University of London (1999)
6. Lee, R.M.: Bureaucracies as deontic systems. ACM transactions on office information systems **6**(2) (April 1988) 87–108
7. Kowalski, R., Sergot, M.: A logic-based calculus of events. New Generation Computing **4** (1986) 67–95
8. Ishikawa, F., Yoshioka, N., Honiden, S.: Developing consistent contractual policies in service composition. Proceedings of the 2nd IEEE Asia-Pacific conference on service computing (Dec. 2007) 527–534
9. Hull, R., Kumar, B., Lieuwen, D., Patel-Schneider, P., Sahuguet, A., Varadarajan, S., Vyas, A.: Enabling context-aware and privacy-conscious user data sharing. Proceedings of the IEEE international conference on mobile data management (2004) 187–198