Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 235 (2009) 35–53
www.elsevier.com/locate/entcs

Event-oriented Web-based E-trading

Steve Barker[1]

Dept. of ComputerScience
King's College London
London, United Kingdom, WC2R 2LS

Gill Lowen[2]

Information Systems Dept.,
Royal Hampshire County Hospital,
Romsey Road, Winchester SO22 5DG, United Kingdom

Abstract

We address the problem of defining policies that may be used in the evaluation of requests made by client actors, in the course of web-based e-trading, to perform actions on the resources maintained by the server agents of an e-cooperative. An e-cooperative is a group of agents in cyberspace that may act individually or in conjunction with other agents to satisfy a client's request to act. Our principal contribution to this key problem is to define formally an event-oriented model in terms of which policies may be specified for helping to ensure that only legitimate forms of client actions are performed in the course of engaging in e-trading via the web. We call this model the *Event-oriented Web-based E-trading (EWE) model*. Policies defined in terms of the EWE model are used to specify a set of actions that client actors can perform as a consequence of the client having a particular status. We define the EWE model using a logic programming language and we give examples of web-based e-trading policy representation, validation and evaluation.

Keywords: Web applications, e-trading, policies, logic specification.

1 Introduction

One of the principal Web-based applications is e-trading. For such applications, client interaction with Web services requires that complex policy specifications be formally specified to define legitimate forms of interactions. For that, conceptual models are required in terms of which web-based e-trading policies can be specified. In this paper, we describe one such model and we show how web-based e-trading policies may be formally specified in terms of this model. These policies may be used in practice for managing web-based e-trading where (pre-authenticated) client agents make requests for an action to be performed by a type of virtual organization

[1] Email: steve.barker@kcl.ac.uk
[2] Email: gillian.lowen@hotmail.co.uk

doi:10.1016/j.entcs.2009.03.004

that we call an *e-cooperative*. Informally, an e-cooperative is a collection of server agents that exist in cyberspace and that may act individually or collaboratively in order to satisfy a client's request that some action be performed.

E-cooperatives implement business rules. For organizations to engage effectively in e-trading via the Web, business rules need to be formally specified in a high-level language, which: permits properties to be proven of policies for verification purposes, allows changes to e-trading policies to be effected autonomously, and allows for the efficient processing of client requests. To meet these requirements, we use a logic language for the definition of a model and policies for web-based e-trading. The logic language allows policy properties to be specified and proof-theoretic principles enable these properties to be verified; operational semantics exist for efficient request evaluation with respect to policy specifications that are expressed in terms of the logic language.

In the model that we define, server agent intentions and empowerments are represented. Moreover, server agent decisions on whether to act to satisfy a client's request will depend on the client's status. To capture these requirements, we propose a new type of model of e-trading, which we call the *event-oriented web-based e-trading (EWE) model*; EWE policies can be specified in terms of the EWE model for the control of client agent actions in the context of e-trading via the Web.

In the EWE model, a coordinator agent for the cooperative determines whether a client's requested action is permitted by considering the client's ascribed status (e.g., the assignment of a client to a role) and a description of events relating to the client. Permitted actions depend on the mode in which a server agent acts, individually or cooperatively. When a server agent of an e-cooperative acts individually to satisfy a client's request then we say that the server agent acts in $\mathcal{I}$-mode; when a server agent must collaborate with other server agents in the e-cooperative to satisfy a client's request (with each server agent contributing to the satisfaction of the request) then we say that the server agent acts in $\mathcal{C}$-mode (cf. [17]). The distinction between $\mathcal{I}$-mode and $\mathcal{C}$-mode acting by server agents motivates the need for a shared meta-policy that determines how server agents may act individually or cooperatively to service client agent requests and to what extent. For that, the notion of a *group ethos* is used. In the $\mathcal{C}$-mode case, a request by a client agent for an action to be performed may be satisfied by the server agents of the e-cooperative subdividing the request and satisfying part of the original request. We call this *request slicing*. In order for a client's action to be performed in $\mathcal{I}$-mode or $\mathcal{C}$-mode the action must be consistent with the group ethos. To the best of our knowledge, the EWE model is the first formal model of e-trading that combines individual and collective actions that are specified in terms of a meta-theory, a group ethos.

Despite being thus far relatively neglected, e-cooperatives and the policies that they employ will become routinely used in the context of Internet and Semantic Web technologies for e-trading. As an e-cooperative, a group of server agents may be able to pool their resources, they may choose to subcontract a client's request for business reasons, and collections of server agents may decide to collaborate in order to combine their individual strengths.

The remainder of the paper is organized in the following way. In Section 2, we describe some formal preliminaries. In Section 3, we describe e-cooperatives in more detail and we informally describe our EWE model. In Section 4, we give the logical axioms that define the EWE model. In Section 5, we describe the application-specific, non-logical axioms and we discuss their use in EWE policy formulation. In Section 6, we describe an implementation of our approach. In Section 7, we discuss the related literature. In Section 8, conclusions are drawn and further work is suggested.

2 Formal Preliminaries

In this section, we describe *Identification-based Logic Programs* (IBLPs). IBLPs are an annotated form of logic programs that allow distributed information sources to be specified in formulations of web-based e-trading policies. IBLPs are based on a syntactic variant of normal logic programs.

Definition 2.1 A normal clause is a formula of the form ($m \geq 0, n \geq 0$):

$$A \leftarrow A_1,\ \ldots,\ A_m,\ not\ A_{m+1}, \ldots,\ not\ A_{m+n}.$$

A normal logic program is a set of normal clauses.

The head A of the clause in Definition 2.1, and each A_i ($1 \leq i \leq m+n$), are atoms. In the *body* of the clause, $A_1,\ \ldots,\ A_m, not\ A_{m+1},\ \ldots,\ not\ A_{m+n}$, the atoms $A_1, \ldots, A_m$ are called 'positive literals', and $not\ A_{m+1}, \ldots, not\ A_{m+n}$ are 'negative literals'; *not* denotes negation-as-failure. A clause with an empty body (i.e., of the form $A \leftarrow$) is a *fact*. In what follows, we usually omit the arrow when writing facts, writing $A \leftarrow$ as A. The term *rule* refers to a clause with a non-empty body. The literals in the body of the clause are also referred to as 'conditions'. We denote variables in clauses by using symbols that start with a letter in the upper case; we denote constants by using symbols that start with a letter in the lower case.

In an IBLP, each atom in the body of a clause is annotated with the name of a uniquely identifiable module, which may be stored on any file server. *Uniform resource identifiers (URIs)* provide a unique global identity for referencing IBLPs

Definition 2.2 An identification-based clause is a formula of the following form:

$$A \leftarrow A_1 \Leftarrow \upsilon_1,\ \ldots,\ A_m \Leftarrow \upsilon_m,\ not\ A_{m+1} \Leftarrow \upsilon_{m+1},$$
$$\ldots,\ not\ A_{m+n} \Leftarrow \upsilon_{m+n} \quad (m \geq 0, n \geq 0).$$

where the head A and each A_i ($1 \leq i \leq m+n$) are atoms, and each υ_i ($1 \leq i \leq m+n$) is a URI in $\mathcal{R}$. An IBLP is a finite set of identification-based clauses.

Informally, the semantics of $\Leftarrow$ can be described thus. Let δ be a mapping $\delta : \mathcal{R} \longrightarrow \{P_1, \ldots, P_n\}$ where $\mathcal{R}$ is a set of URIs, and where each P_i ($1 \leq i \leq n$) is an IBLP. The condition $A_i \Leftarrow \upsilon_i$ is true (provable) iff A_i is true (provable) with

respect to the IBLP $\delta(\upsilon_i)$, and *not* $A_i \Leftarrow \upsilon_i$ is true (provable) iff A_i is not true (not provable in finite time) with respect to $\delta(\upsilon_i)$. We will make this informal semantics more precise below, by means of a translation from IBLPs to normal logic programs. Note however that the annotation $A_i \Leftarrow \upsilon_i$ has an *operational* meaning as well as a logical meaning: whenever a query evaluation attempts execution of $A_i \Leftarrow \upsilon_i$, execution is passed to the (usually remote) server whose URI is υ_i.

We say that the atom A is *defined locally* in υ when A is the head of a clause in the IBLP $\delta(\upsilon)$. If a literal of the form $A_i \Leftarrow \upsilon$ or *not* $A_i \Leftarrow \upsilon$ appears in the body of a clause in $\delta(\upsilon)$, then the module annotation can be omitted: A_i then stands for $A_i \Leftarrow \upsilon$ (and *not* A_i for *not* $A_i \Leftarrow \upsilon$).

Notice that a clause of the form $A \leftarrow A \Leftarrow \upsilon_j$ in the IBLP $\delta(\upsilon_i)$ defines A in terms of its local definition in $\delta(\upsilon_j)$. Operationally, execution of A in υ_i invokes the execution of A in υ_j.

Standard comparison operators on numbers $\{=, \neq, <, >, \leq, \geq\}$ and arithmetic operators $\{+, -, \times, \div\}$ are assumed to be defined locally for all IBLPs. We also allow conditions of the form $A \Leftarrow X$ and *not* $A \Leftarrow X$ in clauses where X is a variable that ranges over a (finite) domain of URIs. The meaning of a condition $A \Leftarrow X$ (*not* $A \Leftarrow X$) in the body of an IBLP clause is that $A \Leftarrow X$ (*not* $A \Leftarrow X$) is true if and only if A is provable (not provable) from $\delta(\upsilon_i) \cup \cdots \cup \delta(\upsilon_j)$, where each υ_k $(i \leq k \leq j)$ identifies an IBLP and υ_k can be substituted for X. We require that an *allowedness condition* is satisfied by every clause c in an IBLP, to wit: every variable in c must occur in a positive literal in the body of c. What is more, a substitution for X must exist whenever an $A \Leftarrow X$ or *not* $A \Leftarrow X$ condition is evaluated.

Definition 2.3 Let $\mathcal{R}$ be a finite set of URIs and $P_1, \ldots, P_n$ be IBLPs. An IBLP configuration is a pair $(\mathcal{R}, \delta)$ where δ is a mapping $\delta : \mathcal{R} \longrightarrow \{P_1, \ldots, P_n\}$.

In order to simplify notation, in the remainder of the paper we will write υ for the IBLP $\delta(\upsilon)$ where context permits, and we will write $\upsilon_i := P_i$ to denote that the IBLP configuration assigns URI υ_i to the IBLP P_i, i.e, $\upsilon_i := P_i$ is shorthand for $\delta(\upsilon_i) =_{\text{def}} P_i$. In our examples, the set of URIs is usually obvious from context.

Next, we define the declarative semantics of IBLPs by means of a *reduction* of IBLPs to normal logic programs. This reduction translates an IBLP P to a normal logic program P' by replacing every occurrence of an atom $p(t_1, \ldots, t_k) \Leftarrow \upsilon$ by the atom $p{:}\upsilon(t_1, \ldots, t_k)$ where $p{:}\upsilon$ is a new predicate symbol constructed by concatenating p and υ. When P is an IBLP identified by υ_i, i.e., when $\delta(\upsilon_i) = P$, we write $\Delta\upsilon_i$ for the normal logic program P' obtained by the reduction just described.

Definition 2.4 Let $(\mathcal{R}, \delta)$ be an IBLP configuration, and let υ_i be a URI in $\mathcal{R}$. $\Delta\upsilon_i$ denotes the normal logic program obtained by replacing every occurrence of an atom of the form $p(t_1, \ldots, t_k) \Leftarrow \upsilon$ in the IBLP $\delta(\upsilon_i)$ by the atom $p{:}\upsilon(t_1, \ldots, t_k)$.

Let $\mathcal{R} = \{\upsilon_1, \ldots, \upsilon_n\}$. The reduction of $(\mathcal{R}, \delta)$, written $\Delta(\mathcal{R}, \delta)$, is the normal logic program $\Delta\upsilon_1 \cup \cdots \cup \Delta\upsilon_n$.

This transformation reduces an IBLP to a normal logic program; hence, we can adopt any of the standard semantics for normal logic programs as the declarative

semantics for IBLPs. In this paper, we will employ the *stable model semantics* [3] (also known as 'answer sets'). The stable model of an IBLP will be defined in terms of the stable model of the normal logic program to which it translates.

Definition 2.5 Let $(\mathcal{R}, \delta)$ be an IBLP configuration. $\mathcal{M}$ is an identification-based (IB) stable model of $(\mathcal{R}, \delta)$ iff $\mathcal{M}$ is a stable model of the normal logic program $\Delta(\mathcal{R}, \delta)$.

Let $\mathcal{R} = \{\upsilon_1, \ldots, \upsilon_n\}$ and $\upsilon_i \in \mathcal{R}$. The set of ground atoms

$$\mathcal{M}_{\upsilon_i} =_{\text{def}} \{p(c_1, \ldots, c_k) \mid p{:}\upsilon_i(c_1, \ldots, c_k) \in \mathcal{M}\}$$

is an identification-based (IB) stable model of υ_i in the configuration $(\mathcal{R}, \delta)$.

We write $\mathcal{M} \models_{\upsilon_i} p(c_1, \ldots, c_k)$ when $p{:}\upsilon_i(c_1, \ldots, c_k) \in \mathcal{M}$. The satisfaction relation $\models_{\upsilon_i}$ is extended from atoms to formulas in the usual way. We sometimes omit the subscript υ_i when it is obvious from context.

In this paper, we restrict attention to *stratified* configurations of IBLPs.

Definition 2.6 Let $(\mathcal{R}, \delta)$ be an IBLP configuration, and $\mathcal{R} = \{\upsilon_1, \ldots, \upsilon_n\}$. $(\mathcal{R}, \delta)$ is stratified iff the normal logic program $\Delta(\mathcal{R}, \delta) = \Delta\upsilon_1 \cup \cdots \cup \Delta\upsilon_n$ is stratified, i.e., iff the dependency graph [3] for $\Delta\upsilon_1 \cup \cdots \cup \Delta\upsilon_n$ is acyclic.

The syntactic restriction to a stratified configuration of IBLPs guarantees that the stable model [3] is unique. Moreover, the soundness of an operational semantics for IBLPs will be the same as that for stratified logic programs and completeness results are preserved in the absence of system-related failures.

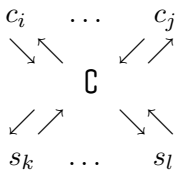
3 Informal Preliminaries

In this section, we informally discuss the key features of the EWE model.

3.1 E-cooperatives

We first describe an architecture for e-cooperatives. For the type of e-cooperative that we assume, all requests are sent by a pre-authenticated requester (a client agent) to a single coordinator agent for the e-cooperative. The e-cooperative typically includes multiple server agents. All clients and server agents are assumed to be uniquely identified. The sets of client agents and server agents are disjoint. The coordinator is neither a server agent of the e-cooperative nor a client of the e-cooperative. The principal role of the coordinator is to manage indirectly all interaction between the clients and the servers of the e-cooperative (e-cooperative server agents do not directly communicate with each other or with clients). The coordinator also manages matters like server agent membership of the e-cooperative and the imposition of sanctions on server agents that act non-normatively in the group context.

In overview, the simple architecture that we will henceforth assume for e-cooperatives may be illustrated thus:



Here, $c_i, \ldots, c_j$ are clients, C is the coordinator and $s_k, \ldots, s_l$ are the server agents of the e-cooperative. The double arrows denote that communications between client agents and the coordinator and between server agents and the coordinator are bidirectional. (Other web topologies are clearly possible.)

A typical interaction will involve a client c making a request to the coordinator for an action to be performed; the coordinator then broadcasts the request to the subset $\mathcal{S}$ of server agents that permit c's requested action to be performed. Each server agent in $\mathcal{S}$ sends to C the optional, intended, empowered, authorized actions that it will perform for c (in $\mathcal{I}$-mode and $\mathcal{C}$-mode). The coordinator C, in turn, communicates these options to c. If c wishes to commit to a proposed option ω then c will request C to commit (atomically) a transaction to effect ω. It is important to note that a server agent of the e-cooperative may also be the coordinator of its own e-cooperative. Hence, multiple e-cooperatives can be linked without restriction and thus complex webs of e-cooperatives can be naturally represented

3.2 Permissions

In order to act on behalf of a client, a server agent s must have an intention to act either individualistically (in $\mathcal{I}$-mode) or cooperatively (in $\mathcal{C}$-mode). The intentions that an agent s of the e-cooperative has to act often depends on whether s is acting in $\mathcal{I}$-mode or $\mathcal{C}$-mode (e.g., for an act of *buying*, the minimum value of a purchase that is allowed by an agent s in $\mathcal{I}$-mode may be less than s allows in $\mathcal{C}$-mode).

In addition to declaring its intentions, each e-cooperative server agent s will specify its own empowerments to act. That is, in addition to being willing to perform a required action, a server agent s must also express its (current) capability of acting in order for s to engage in satisfying c's request that an action be performed. When acting in $\mathcal{I}$-mode, s must be able to satisfy a client's request completely by acting alone. In contrast, when acting in $\mathcal{C}$-mode the agent s acts with other server agents of the e-cooperative to satisfy a client's request to perform action a. Of course, s may be empowered to satisfy a client's request in $\mathcal{C}$-mode when it cannot satisfy the client's request in $\mathcal{I}$-mode, but not vice-versa i.e., $\mathcal{I}$-mode satisfaction implies $\mathcal{C}$-mode satisfaction, but not conversely.

An e-cooperative server agent s that has the intention of performing an action a and that is empowered to do a for a client c must specify whether c should be permitted to perform a when s is acting in $\mathcal{I}$-mode and when s is acting in $\mathcal{C}$-mode. For that, the client's status level is computed. In the case of $\mathcal{I}$-mode authorization, a single e-cooperative server agent s will determine whether c has the status to perform a requested action or not. In contrast, in $\mathcal{C}$-mode all of the server agents $s_1, \ldots, s_n$ that act cooperatively to satisfy c's request must agree that c has the

status that s_i $(1 \leq i \leq n)$ requires c to have in order to have performed whatever action c has requested.

The combination of specifications of an intention policy and an empowerment policy together with the status of the client c is used to define a set of triples (c, a, r), which specify that the server agent s allows a client agent c the option of performing the action a on resource r.

3.3 Group Ethos

In the EWE model, a permission must be consistent with the group ethos of the e-cooperative. The group ethos determines what powers the server agents can exercise in the context of the e-cooperative. We describe a simple form of group ethos such that an e-cooperative server agent s acts to satisfy a request from a client c in full by s operating alone whenever that is consistent with s's specifications of intentions, empowerments and authorizations. If s has an intention to act and authorizes c to perform the requested act but s is not empowered to act then s will request other server agents of the e-cooperative to help in satisfying c's request. Hence, each server agent of the e-cooperative will act in $\mathcal{I}$-mode if possible and in $\mathcal{C}$-mode otherwise. Despite its simplicity, this group ethos is consistent with the concept of *prospective rationality* that is defined within the framework of *Rational Choice* [15]. Henceforth, we will call this ethos the *Principle of Maximal Individualistic Satisfaction*, to wit:

> *If an e-cooperative server agent s intends and can fully satisfy a request of a client c for an action to be performed that s authorizes for c then s will act in an $\mathcal{I}$-mode capacity (i.e., privately, selfishly) subject to any constraints on $\mathcal{I}$-mode action (conversely, s has the minimal commitment to act cooperatively). Otherwise, s determines the maximal extent to which it can satisfy the request by c, and will request help from other server agents of the e-cooperative on how to satisfy the remainder of c's request. In this case, the $\mathcal{C}$-mode case, s invites server agents of the e-cooperative to engage in joint activity to satisfy c's request.*

All server agents must respect the Principle of Maximal Individualistic Satisfaction, the shared group ethos. Other forms of group ethos are, of course, possible (e.g., to allow for group competitiveness); these alternatives are the basis for different ethoses that can, nevertheless, be naturally represented in our framework.

3.4 Events

The notion of an event is of primary importance in our approach. The importance of the concept of events is emphasized by their widespread use in linguistics and knowledge representation (e.g., in verb nominalization). Events provide a categorically homogeneous basis for representing *change*, a feature that is an essential aspect of our EWE model.

We interpret events as happenings at an instance of time. We adopt a one-dimensional, linear, discrete view of time, with a beginning and no end point. That is, the model of time that we choose is a total ordering of time points that is isomorphic to the natural numbers. In the ensuing discussion, we represent times

as natural numbers in $YYYYMMDD$ format. We assume that time is bidirectional so that proactive and postactive changes may be made to represent EWE policy requirements, and past, present and future times can be used to make decisions on e-trading requests made by client agents.

EWE policy conditions will often need to be specified with respect to the current time extracted from the system clock. For that, we utilize the auxiliary predicate $current\_time/1$, which gives system clock times and has a fixed interpretation such that $current\_time(T) = true$ if $T = now$ and $false$ otherwise. The semantics of now is defined in [9].

4 EWE Model: Logical Axioms

In this section, we formally define the EWE model. We begin by describing some of the key sets of constants in the alphabet that we use in the formulation of the EWE model and EWE policies. Specifically, a single e-cooperative Γ will be defined in terms of an alphabet that includes:

- A countable set $\mathcal{C}$ of client identifiers for Γ, c, c_0, c_1, ...
- A countable set $\mathcal{L}$ of client status levels, l, l_0, l_1, ...
- A singleton set $\{\gamma\}$ where γ is the unique identifier of the coordinator for Γ.
- A countable set $\mathcal{S}$ of identifiers s, $s_0, s_1, \ldots$ of server agents of Γ.
- A countable set $\mathcal{A}$ of named atomic *actions*. Actions are strings that denote actions of arbitrary complexity in a world of interest e.g., *buy*, *sell*, *supply*, *hire*, ...
- A countable set $\mathcal{R}$ of *resource identifiers*, r, r_0, r_1, ...
- A countable set $\mathcal{T}$ of *time points*, t, t_0, t_1, ...
- A countable set $\mathcal{E}$ of *event identifiers*, e, e_0, e_1, ...

More formally, an e-cooperative Γ is a tree with the coordinator γ as the root element and with a finite set of server agents $s_1, \ldots, s_n$ that are the descendants of γ and that are either leaf nodes or the root of a subtree of Γ. In the latter case, s_i $(1 \leq i \leq n)$ is both a server agent of Γ and the coordinator of an e-cooperative Γ' $(\Gamma \neq \Gamma')$. Notice too that, from the previous discussion, we have that $\mathcal{C} \cap \mathcal{S} = \emptyset$, $\gamma \notin \mathcal{C}$ and $\gamma \notin \mathcal{S}$ for Γ.

Next, we define the EWE model as a logical theory; henceforth, we use the term *EWE theory* for any instance of a logical theory that is defined in terms of the EWE model. A EWE theory is expressed in terms of a finite set of IBLP rules. From the previous discussion, it should be clear that a EWE theory consists of four subtheories: a $\mathcal{PER}$ subtheory is the set of rules that is used by the e-cooperative coordinator to define the set of actions permitted for client agents, $\mathcal{INT}$ is a theory of intentions that server agents have to act (in $\mathcal{I}$-mode and $\mathcal{C}$-mode), $\mathcal{EMP}$ is a logical theory that specifies a server agent's empowerment to act (in $\mathcal{I}$-mode or $\mathcal{C}$-mode), and $\mathcal{AUTH}$ is a theory that defines authorized actions for clients (in $\mathcal{I}$-mode or $\mathcal{C}$-mode).

4.1 $\mathcal{PER}$ Theory

The $\mathcal{PER}$ sub-theory, that is maintained at the coordinator site for the e-cooperative, consists of the following rules:

$$permission(C, A, R) \leftarrow agent(S), sla(C, L), i\_permission(L, A, R, S).$$

$$permission(C, A, R) \leftarrow agent(S), sla(C, L), c\_permission(L, A, R, S).$$

$$i\_permission(L, A, R, S) \leftarrow i\_empowered(L, A, R) \Leftarrow E_i,$$
$$i\_intent(L, A, R) \Leftarrow I_i, i\_authorized(L, A, R) \Leftarrow A_i.$$

$$c\_permission(L, A, R, S) \leftarrow c\_empowered(L, A, R) \Leftarrow E_c,$$
$$c\_intent(L, A, R) \Leftarrow I_c, c\_authorized(L, A, R) \Leftarrow A_c.$$

The meaning of $permission/3$ is that a client c has the option of performing the action a on resource r if c is assigned the status l (as defined by $sla/2$ where sla is short for status level assignment) and some server agent s of the e-cooperative (acting in $\mathcal{I}$-mode) or some (finite) subset $\{s_1, \ldots, s_n\}$ of the set of server agents of the e-cooperative (acting in $\mathcal{C}$-mode) specify that clients with l status have the option of performing the action a on r.

The *i_permission* clause is used to specify that client c's request to perform the a action on resource r is satisfied by the e-cooperative server agent $s \in \mathcal{S}$ acting in $\mathcal{I}$-mode if, for l level clients of which c is one, s has the intention of performing the action a on the resource r, in $\mathcal{I}$-mode, s is empowered, in $\mathcal{I}$-mode, to perform the action a on r, and s authorizes c to perform the action a in $\mathcal{I}$-mode. The *c_permission* clause specifies that client c's request to perform the a action on resource r is satisfied by the e-cooperative server agent s acting in $\mathcal{C}$-mode if, for c's status level assignment l, s has the intention of performing the action a on the resource r, in $\mathcal{C}$-mode, s is empowered, in $\mathcal{C}$-mode, to perform the action a on r, and s authorizes the performance of the requested action.

4.2 Event Representation

In the EWE model, the actions a client can perform will depend, in part, upon the transactions the requester agent has engaged in with the e-cooperative. Transactions are expressed via a set of application-specific *security event descriptions (SEDs).* [3]

Definition 4.1 A security event description is a finite set of ground 2-place assertions that describe an event, identified by $e_i, i \in \mathbb{N}$, and which includes three *necessary* facts, and n non-necessary facts ($n \geq 0$). A necessary fact in a security

[3] We restrict attention to a simple form of security event description in this paper; however, more complex forms of SEDs are possible e.g., events with duration.

event description ϵ is a fact that must appear in ϵ; all other facts in an SED are non-necessary.

The three necessary facts in a SED together with their intended meanings are as follows (where $MOD(EWE)$ denotes the IB stable model of a EWE theory):

- $MOD(EWE) \models happens(e_i, t_j)$, *the event* e_i *happens at time* t_j.
- $MOD(EWE) \models act(e_i, a_l)$, *the event* e_i *involves an action* a_l.
- $MOD(EWE) \models agent(e_i, c_m)$, *the event* e_i *involves the client agent* c_m.

It follows from the discussion above that the $happens(e_i, t_j)$, $act(e_i, a_l)$ and $agent(e_i, c_m)$ facts in an SED are used to represent a happening e_i at a time t_j of *act* a_l performed by a client agent c_m.

Example 4.2 *Consider the SED,* $\epsilon = \{happens(e_1, 20080612); agent(e_1, bob); act(e_1, buying); object(e_1, widget); colour(e_1, green); amount(e_1, 1000)\}$. *The facts in* ϵ *describe an event* e_1 *that happens on 12th June, 2008, and involves the agent Bob buying an amount of* 1000 *green widgets. The* $amount(e_1, 1000)$, $object(e_1, widget)$ *and* $colour(e_1, green)$ *facts are non-necessary facts in* ϵ.

It follows from the discussion above that we view events as structured and described via the set of predicates that form a SED. We do not envisage SEDs including adjectival or adverbial modifiers, and we assume that all distinctly named acts, agents, objects, etc. denote distinct acts, agents, objects, respectively. Our binary representation of the elements of an SED is motivated by Davidson's work on event semantics [11]. Henceforth, we call a finite set of SEDs $\{\epsilon_1, \ldots, \epsilon_n\}$ a *history*.

Recall that, in the EWE model, a client agent's status is used to determine the actions the client may perform. On that, a set Λ of axioms is used to specify the assignment of a client agent to a status level; Λ is intended to capture the following form of reasoning:

> *A client agent* C *is currently assigned the status level* L *if an event* $E1$ *happened at a time* $T1$*, that is earlier than the current time* T*, and resulted in the initiation of* C*'s assignment to* L*, and this assignment has not been ended before* T *as a consequence of an event* $E2$ *happening at a time* $T2$ *in the interval* $[T1, T]$ *that causes* C*'s assignment of the status level* L *to be terminated.*

To capture the required form of reasoning, Λ includes the following axioms, Λ_1 and Λ_2, that define the predicates `sla/2` and `ended_sla/4`:

$$\langle\Lambda_1\rangle \quad sla(C, L) \leftarrow current\_time(T), agent(E1, C), happens(E1, T1), act(E1, A), T1 \leq T, sla\_init(E1, C, A, L, T1, T), not\ ended\_sla(C, L, T1, T).$$

$$\langle\Lambda_2\rangle \quad ended\_sla(C, L, T1, T) \leftarrow agent(E2, C), happens(E2, T2), act(E2, A), sla\_term(E2, C, A, L, T1, T), T1 \leq T2, T2 \leq T.$$

The axioms Λ_1 and Λ_2 are part of the core set of axioms of the EWE model. Notice too that our use of $act(E1, A)$, $agent(E1, C)$, and $happens(E1, T1)$ in the definition of Λ_1 is based on our conception of an event, see Definition 4.1, and emphasizes that we consider the act A a client agent C performs at a time $T1$ to be necessary facts the description of an event $E1$.

The predicates $sla\_init/6$ and $sla\_term/6$ in Λ_1 and Λ_2, specify how the actions performed by a client agent C affect the agent's status level assignments. The description of an agent's history of actions is part of the application-specific component of a formulation of a EWE policy. Each $sla\_init(E1, C, A, L, T1, T)$ definition expresses that, according to the information held in the system at time T, an event E representing an action A by C at time $T1$ initiates a period of time, starting at $T1$, during which C is assigned status level L. Each definition of $sla\_term(E2, C, A, L, T1, T)$ expresses that an event $E2$, representing an action A by a client C at time T, terminates a period of time during which C was assigned status level L since $T1$.

From a history (of SEDs) and rules defining status initiation and termination, a client's status may be determined by the coordinator of an e-collective. This status is used to decide what actions a client agent is able to perform according to a EWE specification that defines a EWE policy for an e-collective.

5 EWE Model: Non-logical Axioms

In this section, we describe the non-logical axioms that are maintained by individual web servers in an e-cooperative. We describe the different sets of rules in terms of a running example. The example is based on a simple form of e-cooperative with four Web server agents that are identified by s_1, s_2, s_3 and s_4. We make the simplifying assumptions that the only resource of interest to client agents is a *part* relation and the only action of interest is an act of *buying*. Each server agent of an e-cooperative, with an intention of satisfying a client's request to buy some part, will maintain its own version of *part*. In our example, the *part* relation has the structure $part(X_1, X_2, X_3)$, where X_1 ranges over a domain of part names, X_2 ranges over a domain of colors (of parts) and X_3 is the quantity of colored parts. Of course, relations may be far more complex than $part/3$ and many more relations and actions will typically be involved in practice, but these extra complexities do not raise any special issues. As a further simplification, to reduce the number of predicates that we use in our example policies, we ignore the issue of satisfying allowedness and range-restriction conditions. The simple example of a EWE policy that we describe in this section is presented with exposition as the main motivation.

Given a EWE policy specification, a client c_0 may make a request of the form $permission(c_0, buy, part(p, c, q))$. In this case, c_0's request is to know whether it is permitted to buy q units worth of the p part colored c. On receiving c_0's request, the coordinator for an e-cooperative will request the server agents of the e-cooperative to evaluate c_0's request with respect to their declarations of intentions, empowerments, and authorizations.

5.1 $\mathcal{INT}$ Theory

An intention theory, $\mathcal{INT}$, for a server agent s defines s's intentions (if any) in terms of an action a that s is willing to engage in on resource r in $\mathcal{C}$-mode on behalf of L-level clients (the *c_intent* rules), to wit:

$$i\_intent(L, A, R) \leftarrow C_{1_i} \Leftarrow \upsilon_{1_i}, \ldots, C_{m_i} \Leftarrow \upsilon_{n_i}.$$
$$c\_intent(L, A, R) \leftarrow C_{1_c} \Leftarrow \upsilon_{1_c}, \ldots, C_{m_c} \Leftarrow \upsilon_{n_c}.$$

Each *i_intent* (*c_intent*) clause specifies that the e-cooperative server agent s has an intention to perform an action a on resource r in $\mathcal{I}$-mode ($\mathcal{C}$-mode) on behalf of client c with status L if the conjunction of conditions, $C_{1_i} \Leftarrow \upsilon_{1_i}, \ldots, C_{m_i} \Leftarrow \upsilon_{n_i}$ ($C_{1_c} \Leftarrow \upsilon_{1_c}, \ldots, C_{m_c} \Leftarrow \upsilon_{n_c}$) is satisfied.

Example 5.1 *Consider the following set of intention theories for the e-cooperative server agents s_1, s_2, s_3 and s_4 such that these intention theories are identified as $\mathcal{INT}_{s_1}$, $\mathcal{INT}_{s_2}$, $\mathcal{INT}_{s_3}$, and $\mathcal{INT}_{s_4}$, respectively:*

$$\mathcal{INT}_{s_1} := \begin{cases} i\_intent(L, buy, part(X, Y, Z)). \\ \\ c\_intent(L, buy, part(X, Y, Z)) \leftarrow Z > 1000. \end{cases}$$

$$\mathcal{INT}_{s_2} := \begin{cases} i\_intent(L, buy, part(widget, green, Z)) \leftarrow Z \leq 1000, L > l_2. \\ \\ c\_intent(L, buy, part(widget, red, Z)) \leftarrow Z \geq 50. \end{cases}$$

$$\mathcal{INT}_{s_3} := \Big\{ i\_intent(l_0, A, R).$$

$$\mathcal{INT}_{s_4} := \begin{cases} i\_intent(L, A, R) \leftarrow current\_time(T), T \geq 20080601. \\ c\_intent(L, A, R) \leftarrow current\_time(T), T \geq 20081001. \end{cases}$$

Here, $\mathcal{INT}_{s_1}$ is a specification of the intentions the e-cooperative server agent s_1 has of satisfying any client's request to perform an act of buying in $\mathcal{I}$-mode or in $\mathcal{C}$-mode provided that the total size of the order (in the latter case) is greater than 1000 units. $\mathcal{INT}_{s_2}$ expresses s_2's intention to act individually to satisfy a request from any client with greater than l_2 status to buy if the request is for less than 1000 green widgets; c_2 is also willing to act in $\mathcal{C}$-mode to satisfy requests for sales of more than 50 units worth of red widgets from clients with any status L. $\mathcal{INT}_{s_3}$ expresses that s_3 is willing to satisfy any action on any resource without restriction, but only in $\mathcal{I}$-mode and only for the clients that have l_0 status. Finally, $\mathcal{INT}_{s_4}$ specifies that s_4 is willing to satisfy any action requested by clients with status L on any resource in $\mathcal{I}$-mode as soon as s_4 starts trading on 1st June 2008 and will act in $\mathcal{C}$-mode four months after starting to trade.

5.1.1 $\mathcal{EMP}$ Theory

In addition to an $\mathcal{INT}$ theory, each server agent s of the e-cooperative will maintain its own $\mathcal{EMP}$ theory. An $\mathcal{EMP}$ theory is a set of rules of the form:

$$i\_empowered(L, A, R) \leftarrow C_{1_i} \Leftarrow \upsilon_{1_i}, \ldots, C_{m_i} \Leftarrow \upsilon_{n_i}.$$
$$c\_empowered(L, A, R) \leftarrow C_{1_c} \Leftarrow \upsilon_{1_c}, \ldots, C_{m_c} \Leftarrow \upsilon_{n_c}.$$

Each *i_empowered* (*c_empowered*) clause specifies that s is empowered to perform the a action on resource r for client c in $\mathcal{I}$-mode ($\mathcal{C}$-mode) if the conjunction of conditions $C_{1_i} \Leftarrow \upsilon_{1_i}, \ldots, C_{m_i} \Leftarrow \upsilon_{n_i}$ ($C_{1_c} \Leftarrow \upsilon_{1_c}, \ldots, C_{m_c} \Leftarrow \upsilon_{n_c}$) is satisfied. In the absence of a definition of *i_empowered* (*c_empowered*), s is not empowered to perform the action a on r for any client in $\mathcal{I}$-mode ($\mathcal{C}$-mode).

For an $\mathcal{EMP}$ theory, it is important to recognize that, in the case of $\mathcal{C}$-mode evaluation, at least one of the C_{j_c} ($1 \leq j \leq m$) conditions is expressed in terms of *c_permission* (or *permission*). The reason for this should be noted: in the case of $\mathcal{C}$-mode evaluation, the evaluation of *c_empowered* involves a server agent s of the e-cooperative requesting the collaboration of other agents in the e-cooperative (or another e-cooperative if the server agent of the e-collective γ is also a coordinator for an e-collective γ', $\gamma \neq \gamma'$) in satisfying the request that an a action be performed in relation to resource r on behalf of the client c. Hence, a *c_permission* condition in the body of a *c_empowered* clause is a recursive call that generates a set of server agents of the e-cooperative (or another e-cooperative if a *permission* condition is evaluated) that will collaborate with s to satisfy the request by c to perform the action a on r. Hence, when s acts in $\mathcal{C}$-mode then c requests to act jointly with other e-cooperative server agents that also have an intention and are empowered and authorized to perform the requested action. By the Principle of Maximal Individualistic Satisfaction, a server agent s of the e-cooperative, which acts collaboratively with other e-cooperative server agents, will offer to satisfy a client's request to the maximal extent that is consistent with the permissions that s defines.

Next, consider the $\mathcal{EMP}$ theories to be used with the $\mathcal{INT}$ theories, from Example 5.1, and the application-specific information on the stock levels of colored widgets held by the server agents s_1, s_2, s_3 and s_4 and stored in the IBLPs identified by υ_1, υ_2, υ_3 and υ_4, respectively, viz: $\upsilon_1 := \{stock(widget, green, 200)\}$, $\upsilon_2 := \{stock(widget, green, 100)\}$, $\upsilon_3 := \{stock(widget, green, 1000)\}$, and $\upsilon_4 := \{stock(widget, green, 150)\}$. Here, $stock(r, c, q) \in \upsilon_i$ iff the e-cooperative server agent s_i ($1 \leq i \leq 4$) has the quantity q of the resource of type r and of color c available in stock. (We assume that the stock relation used by different e-cooperative server agents has a common format, but this need not, of course, be the case.)

Example 5.2 *Consider the following theories $\mathcal{EMP}_{s_1}$, $\mathcal{EMP}_{s_2}$, $\mathcal{EMP}_{s_3}$, and $\mathcal{EMP}_{s_4}$, for the e-cooperative server agents s_1, s_2, s_3 and s_4, respectively. These empowerment theories assume that the Principle of Maximal Satisfaction is to be implemented. Two of the definitions involve e-cooperative server agents querying*

the coordinator (assumed to be referenced by κ) to generate $\mathcal{C}$-mode responses:

$$\mathcal{EMP}_{s_1} := \begin{cases} i\_empowered(L, buy, part(X,Y,Z)) \leftarrow stock(X,Y,V) \Leftarrow \upsilon_1, Z \leq V. \\ c\_empowered(L, buy, part(X,Y,Z)) \leftarrow stock(X,Y,V) \Leftarrow \upsilon_1, Z > V, \\ \qquad A = Z - V, c\_permission(L, buy, part(X,Y,A), S) \Leftarrow \kappa, \\ \qquad S \neq s_1. \end{cases}$$

$$\mathcal{EMP}_{s_2} := \begin{cases} i\_empowered(L, buy, part(X,Y,Z)) \leftarrow current\_time(T), \\ \qquad month(T,M), 6 \leq M, M \leq 9, stock(X,Y,V) \Leftarrow \upsilon_2, \\ \qquad Z \leq V, Z \leq 500. \\ c\_empowered(L, buy, part(X,Y,Z)) \leftarrow Z \geq 500, \\ \qquad permission(s_2, buy, part(X,Y,Z)) \Leftarrow \pi_{s_2}. \end{cases}$$

$$\mathcal{EMP}_{s_3} := \Big\{ i\_empowered(l_0, buy, part(X,Y,Z)) \leftarrow stock(X,Y,V) \Leftarrow \upsilon_3, Z \leq V.$$

$$\mathcal{EMP}_{s_4} := \begin{cases} i\_empowered(L, buy, part(X,Y,Z)) \leftarrow stock(X,Y,V) \Leftarrow \upsilon_4, \\ \qquad Z \leq V, L \geq l_3. \\ c\_empowered(L, buy, part(X,Y,Z)) \leftarrow stock(X,Y,V) \Leftarrow \upsilon_4, \\ \qquad Z > V, A = Z - V, \\ \qquad c\_permission(L, buy, part(X,Y,A), S) \Leftarrow \kappa, S \neq s_4. \end{cases}$$

Here, the agent s_1 will act in $\mathcal{I}$-mode for any client with status L if s_1 is empowered to so act by having enough stock, as recorded in υ_1, to satisfy the request to perform an act of buying. Alternatively, if s_1 cannot satisfy the request itself then s_1 will attempt to act in $\mathcal{C}$-mode, with any other server agent of the e-cooperative, to service c's request. That is, s_1 will adhere to the Principle of Maximal Individualistic Satisfaction and satisfy whatever it can of the request and will then ask for assistance with the remainder. In contrast, s_3 is only prepared to act in $\mathcal{I}$-mode for l_0 level clients and if it has enough stock as recorded in υ_3 to satisfy a buying request. In $\mathcal{I}$-mode, s_2 is empowered to allow an act of buying subject to certain temporal constraints being satisfied and for orders of less than 500 units. In the case where a request is to buy more than 500 units, s_2 will seek assistance from outside of the e-cooperative by checking its permissions defined at π_{s_2} and requesting empowered agents, to which s_2 subcontracts "large" orders, to contribute to s_2's satisfaction of a buying request. The empowerement theory for s_4 is the same as s_1's empowerment theory except that s_4 queries the e-cooperative server agents other than itself in attempting to satisfy a client's request, for clients with status L in the $\mathcal{C}$-mode case, and its empowerments in the $\mathcal{I}$-mode case apply only to clients with status greater than l_3.

5.1.2 $\mathcal{AUTH}$ Theory

An $\mathcal{AUTH}$ theory is a set of rules for specifying whether s authorizes clients with status l to perform an action a on an resource r when s is acting in $\mathcal{I}$-mode or

$\mathcal{C}$-mode. These rules are of the following general form:

$$i\_authorized(L, A, R) \leftarrow C_{1_i} \Leftarrow v_{1_i}, \ldots, C_{m_i} \Leftarrow v_{n_i}.$$
$$c\_authorized(L, A, R) \leftarrow C_{1_c} \Leftarrow v_{1_c}, \ldots, C_{m_c} \Leftarrow v_{n_c}.$$

Each *i_authorized* (*c_authorized*) clause specifies that the server s authorizes clients with status l to perform an action a on resource r in $\mathcal{I}$-mode ($\mathcal{C}$-mode) if the conjunction of conditions $C_{1_i} \Leftarrow v_{1_i}, \ldots, C_{m_i} \Leftarrow v_{n_i}$ ($C_{1_c} \Leftarrow v_{1_c}, \ldots, C_{m_c} \Leftarrow v_{n_c}$) is satisfied. In the absence of an *i_authorized* (*c_authorized*) definition, clients with L status are not authorized by s to perform the action a on r when s is acting in $\mathcal{I}$-mode ($\mathcal{C}$-mode).

Example 5.3 *For the authorization subtheory that is used in our running example, we make the simplifying assumption that all server agents of the e-collective use the following definitions:*

$$i\_authorized(L, A, R) \leftarrow pla(L, A, R).$$
$$c\_authorized(L, A, R) \leftarrow not\ dla(L, A, R).$$

That is, a client agent with status L is authorized by an e-collective server agent s to perform the A action on resource R in $\mathcal{I}$-mode if L-level clients are assigned the A privilege on R. In contrast, a client agent with status L is authorized by an e-collective server agent s to perform the A action on resource R in $\mathcal{C}$-mode if L-level agents are not denied the A privilege on R.

Any number of authorization theories of the type that we describe can, of course, be flexibly defined.

6 Implementation Issues

In this section, we briefly describe *one* candidate implementation (in Ciao Prolog [8]) of a EWE theory and some performance measures for the implementation.

For testing our implementation, we use the example EWE policies from Section 5 (these policy specifications are directly executable in Ciao). The $\mathcal{AUTH}$ subtheory that we use (cf. Example 5.3) includes 1000 *pla* definitions and 1000 *dla* definitions that relate to the client c_0; c_0 is assigned to one status level to which each *pla* and *dla* applies, none of which constrains access to data. Moreover, a 2000 event history of SEDs is used in testing. For distributed processing, each $p(t_1, \ldots, t_n) \Leftarrow c$ condition in an action control policy specification is translated to a `:-use_active_module(c,[p/n,...])` directive where the active module `c` is executed at the s_i sites ($1 \leq i \leq 4$), which are remote from the coordinator site. Each $p(t_1, \ldots, t_n) \Leftarrow X$ condition is mapped to a conjunction of conditions (where module X is stored locally): $consult(X), p(t_1, \ldots, t_n)$.

The set of refutations that are generated in a successful derivation for a client's request gives the possible ways of satisfying the request. Specifically, from each refutation, the set of *i_permission* and *c_permission* atoms gives a candidate option for satisfying the client's request (cf. [4]).

Example 6.1 We consider two example requests, r_1 and r_2, issued on 1st July 2008 by the client c_0 with status l_0 to the e-cooperative with the agents identified as s_1, s_2, s_3 and s_4:

$$r_1 : permission(c_0, buy, part(widget, green, 50)).$$
$$r_2 : permission(c_0, buy, part(widget, green, 250)).$$

The $\mathcal{INT}$ and $\mathcal{EMP}$ theories that are used in testing are from Example 5.1 and Example 5.2, respectively. From the $\mathcal{AUTH}$ theory that we use, c_0 is authorized to buy any quantity of green widgets by s_1, s_2, s_3 and s_4 acting in $\mathcal{I}$-mode or $\mathcal{C}$-mode. It follows that the candidate *i_permission* and *c_permission* atoms that are generated by the e-cooperative coordinator, in response to the requests r_1 and r_2, can be expressed, in disjunctive normal form, for l_0 status level users of which c_0 is one, thus:

$$i\_permission(l_0, buy, part(widget, green, 50), s_1) \vee$$
$$i\_permission(l_0, buy, part(widget, green, 50), s_2) \vee$$
$$i\_permission(l_0, buy, part(widget, green, 50), s_3) \vee$$
$$i\_permission(l_0, buy, part(widget, green, 50), s_4).$$

$$i\_permission(l_0, buy, part(widget, green, 250), s_3) \vee$$
$$(i\_permission(l_0, buy, part(widget, green, 200), s_1) \wedge$$
$$c\_permission(l_0, buy, part(widget, green, 50), s_4)) \vee$$
$$(i\_permission(l_0, buy, part(widget, green, 150), s_4) \wedge$$
$$c\_permission(l_0, buy, part(widget, green, 100), s_1)).$$

The permissions are communicated to c_0; each disjunct in one of the DNF formulas above expresses a candidate option for satisfying c_0's request. From these options, c_0 will select the option (a transaction) it wishes to have performed (if any) and informs the coordinator of its decision. The transaction that c_0 wishes to have performed is rechecked on receipt by the coordinator. If the permission holds for c_0 at the time of the check then the coordinator commits the transaction atomically.

For testing, we initially evaluate r_1 and r_2 with respect to our example EWE policy stored on a single machine. This enables CPU times for request evaluation to be generated. We then repeat the evaluation when the intention, empowerment

and authorization theories for s_1, s_2, s_3 and s_4 are remotely located (with respect to the coordinator).

Our tests have been performed using a Sun Ray machine (a terminal to a Sun Fire v-490 cluster with 8 CPUs and 10 Gbtye RAM) running Solaris 10 together with a Toshiba Satellite Pro 2100 machine with an Intel Pentium 4, 1.9GHz processor and 1Gbyte RAM. The two machines are connected over a 100Mbits per second Ethernet LAN. The active modules are executables on the Satellite.

For the first set of tests, using measures extracted from the built-in `statistics` predicate, the evaluations of r_1 and r_2 take a few milleseconds of CPU time to perform (averaged over 10 runs). For the second set of tests, we use *walltime*. For these tests, walltime costing are of the order of 2-5 seconds (again averaged over 10 runs). These processing costs are within acceptable bounds given the "expensive case" choices in testing (i.e., in practice, we do not envisage 2000 rules and a 2000 event history being accessed to evaluate the requests of a single client). In practice, client "think time" [1] will dominate processing costs.

7 Related Work

The issue of developing policy specification languages for Web and Semantic Web applications has received considerable attention in recent years. For example, Berners-Lee [6] has proposed N3 and the related policy languages Rei and Rein. Moreover, XACML [16], KAoS and RuleML [7] have been developed as languages for policy specification for Web-based applications. Rather than proposing another form of policy language, the principal contribution of our approach is to define a general *model* in terms of which a variety of web-based e-trading policies can be formally defined. Although, languages like XACML can be used to define business policies (and access control policies, in particular), XACML is a language for policy specification rather than being a model in terms of which policy information is defined. Similarly, the work by Antoniou and Arief [2] is related to ours in the sense that a declarative language for business rules for e-commerce is defined, but again the emphasis in [2] is on the language for policy specification rather than the model that defines the conceptual framework in terms of which the language is to be used. It should also be noted that our model and the policy languages that it admits enables both business rules and authorization policies to be defined in a general framework. In effect, the authorization rules enable an access control policy to be defined whereas the intention and empowerment rules are of fundamental importance in expressing business requirements.

The Ponder language [10] has been developed to enable a range of management and security policies to be represented and incorporates a notion of events for determining what actions are permissible (as we do). However, unlike our approach, Ponder is a language-based approach for policy representation rather than being model-based and the formal semantics of Ponder are not well defined.

From the perspective of the access control community, we note that rule-based, certificate-based languages have been proposed for distributed security policy speci-

fication (e.g., SD3 [14] and Binder [12]). Nevertheless, these languages are concerned with defining authorization policies and (again) do not assume the existence of a well-defined model in terms of which authorization policies are defined. In the access control literature, there has been much more of an emphasis on the models for centralized systems in terms of which languages for policy specification are defined. For example, Barker and Stuckey [5] adopt a rule-based approach for formally specifying a family of policy languages from the definitions of conceptual models that are related to the family of ANSI RBAC models [13]. Methodologically, our approach is of this same type as that used in [5]. However, we combine intentions and empowerments with specification of authorizations. In our proposal, specifications of intentions and empowerments are not simply to be included as "side-conditions" in terms of which authorizations are defined. To the best of our knowledge, no e-trading model has thus far been proposed that combines the concepts of server agent intentions and empowerments and client event-based statuses to develop a well-founded model of web-based e-trading that is grounded in a well-established theory of cooperation.

8 Conclusions and Further Work

We have defined formally the EWE model in terms of which policies for event-oriented web-based e-trading can be defined. The EWE model incorporates intentional, empowered, authorized actions that apply to clients with different statuses, which are determined from the history of client engagement in e-trading. Server agents of an e-cooperative may act individually or jointly and in a manner that is consistent with a group ethos. Intentions, empowerments and authorizations are important in their own right for determining legitimate forms of access. However, they become especially important to consider when server agents may act collaboratively as well as individually to satisfy requests for action. In addition to being formally well-defined, the EWE model: enables changes to policies to be effected dynamically and autonomously; it makes it possible to treat individualistic and joint actions and the combining of multiple actions and e-cooperatives in a completely uniform manner (in both cases, by exploiting recursive definitions of empowerments); and it permits properties of policies to be defined and proven for assurance purposes. EWE policies can also be efficiently implemented. Moreover, despite the complexities involved in policy expression, EWE policies can be straightforwardly specified using a declarative language (the EWE model is defined by a small number of axioms that can be specialized for application-specific requirements).

To simplify the discussion in this paper, we have described a simple EWE model in terms of a basic form of group ethos and atomic events. It must be noted, however, that many forms of ethos and more complex forms of events can be naturally accommodated in more powerful EWE models. In future work, we want to consider other forms of group ethos (e.g., to allow for server agent competitiveness in request servicing); we also intend to consider an enhanced form of the EWE model on which constraints may be expressed to capture higher-level policy requirements.

Other matters that demand attention include to consider the use of an XML-based representation of EWE policies (cf. RuleML) and preference specification on EWE policy rules with rule conflict resolution strategies.

References

[1] Rakesh Agrawal, Michael Carey, and Miron Livny. Concurrency control performance modeling: Alternatives and implications. *ACM TODS*, 12(4):609–654, 1987.

[2] Grigoris Antoniou and M. Arief. Executable declarative business rules and their use in electronic commerce. In *SAC*, pages 6–10, 2002.

[3] C. Baral and M. Gelfond. Logic programming and knowledge representation. *JLP*, 19/20:73–148, 1994.

[4] S. Barker. Protecting deductive databases from unauthorized retrieval and update requests. *Journal of Data and Knowledge Engineering*, 23(3):231–285, 2002.

[5] S. Barker and P. Stuckey. Flexible access control policy specification with constraint logic programming. *ACM Trans. on Information and System Security*, 6(4):501–546, 2003.

[6] Tim Berners-Lee, Dan Connolly, Lalana Kagal, Yosi Scharf, and Jim Hendler. N3logic: A logical framework for the world wide web. *CoRR*, abs/0711.1533, 2007.

[7] H. Boley, S. Tabet, and G. Wagner. Design rationale of ruleml: A markup language for semantic web rules. In *SWWS 2001*, pages 381–401, 2001.

[8] *The Ciao Prolog System*, 2004.

[9] J. Clifford, C. Dyreson, T. Isakowitz, C. Jensen, and R. Snodgrass. On the semantics of "now" in databases. *ACM TODS*, 22(2):171–214, 1997.

[10] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The Ponder Policy Specification Language. In *Proc. Int. Workshop on Policies for Distributed Systems and Networks*, volume 1995 of *LNCS*, pages 18–38. Springer, 2001.

[11] Donald Davidson. *Essays on Actions and Events*. 2001. Oxford University Press.

[12] J. DeTreville. Binder, a logic-based security language. In *Proc. IEEE Symposium on Security and Privacy*, pages 105–113, 2002.

[13] David F. Ferraiolo, Ravi S. Sandhu, Serban I. Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed nist standard for role-based access control. *ACM Trans. Inf. Syst. Secur.*, 4(3):224–274, 2001.

[14] T. Jim. SD3: A trust management system with certified evaluation. In *IEEE Symp. Security and Privacy*, pages 106–115, 2001.

[15] Tadao Miyakawa. *The Science of Public Policy*. Routledge, 1999.

[16] OASIS. eXtensible Access Control Markup Language (XACML), 2003. http://www.oasis-open.org/xacml/docs/.

[17] R. Tuomela. *Cooperation*. Kluwer, 1999.