Representing and Enforcing E-Commerce Contracts Using Occurrences

Alan S. Abrahams and Jean M. Bacon
University of Cambridge Computer Laboratory
William Gates Building, J J Thomson Avenue
Cambridge CB3 0FD, UK
{asa28, jmb}@cam.ac.uk

Abstract

This paper describes an approach for representing electronic commerce contracts using occurrences. We first define what is meant by an occurrence and demonstrate how an occurrence may be used to store a workflow event, such as a purchase. Next we demonstrate how the occurrence abstraction can be used to store names. We continue with an explanation of various types of query and demonstrate how queries can be represented and stored using occurrences. We show that the storage of queries is necessary in order to determine which stored descriptions describe a given occurrence, and consequently which policies apply to that occurrence. A discussion of contracts and their representational requirements follows, and we show how occurrences and queries may be used to represent defined terminology, regular and conditional duties, permissions, and powers in contracts. We provide an overview of our implementation and describe related work.

1 Introduction

Business policies, originating from internal and external contractual agreements, are currently buried in application logic in various opaque languages, such as Java, C, and SQL. These languages do not provide businesses with a direct means of representing, storing, querying, enforcing, disseminating, and collating the business policies associated with contracts.

We present a novel approach for representing electronic commerce contracts using occurrences. We begin with an overview of the context of our work. We then describe how occurrences are represented and stored, and explain how the participants in occurrences may be explicitly specified using identifiers or may be described using queries. We illustrate how queries can be stored and demonstrate how the ability to determine which queries cover a particular item stored in the database may be used to determine which contractual provisions (policies) apply to a given item. We discuss how contracts may be represented, and consequently automatically performed and enforced, in terms of defined terminology, regular and conditional duties, permissions, and powers. A brief overview of our implementation architecture is provided. We conclude with a comparison to related work and a summary of our results.

2 Overview

The concepts described in this paper have been implemented in the E-commerce Application Development and Execution Environment (EDEE), written in Java. The implementation context for the environment is as shown in *Figure 1 – Contextual Overview* below. Business analysts feed the business policies defined in the business's contracts and the system's user requirements specification into the system using EdeeLang, a language which provides support for occurrences, queries, and policies as described in this paper.

The business policies (contracts between the legal entities in the human activity system) and the user requirements specification for the system (contracts between the computer system and its users), are stored in an occurrence store. For instance, a user requirements specification for a system may contain the policy 'the system is obliged to email the travel itinerary to the user 60 days before travel' which encodes the contract between the system and its users and is stored in the occurrence store. Workflow occurrences (fulfillment occurrences) and further contracting occurrences are added to the occurrence store and checked against existing contracts. Occurrences are triggered automatically by the system in accordance with the policies defined in the contracts (specifications) in the occurrence store.

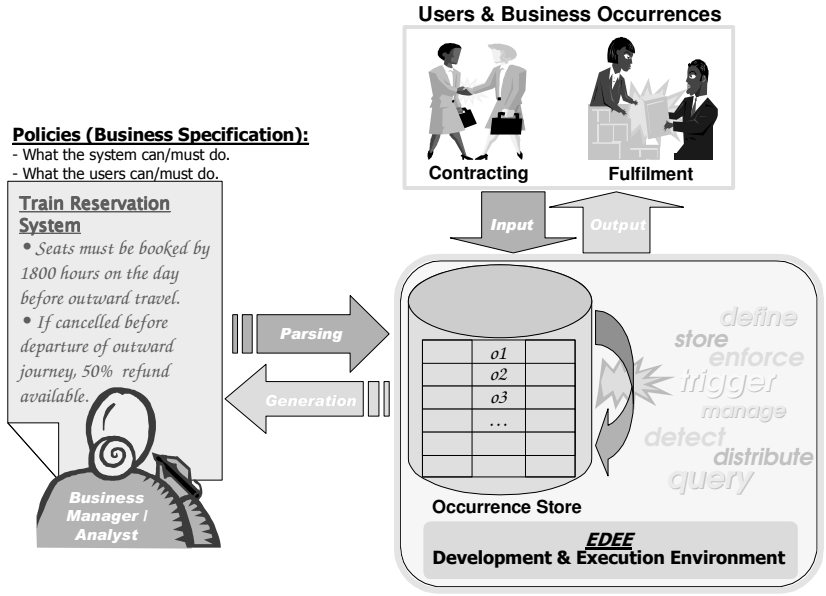

Figure 1 – Contextual Overview

This paper illustrates the detailed underlying representation of occurrences, which are used to store policies and workflow events, at a storage level. It should be remembered though that, for reasons of usability, the interface to the system which stores the contracts and policies is more likely to be a constraining menu-driven interface or a simple English-like language which allows the user to work with the policies in a convenient and readable manner.

3 Representing and Storing Occurrences

An occurrence is a happening that occurs at a particular point in time or that occupies a continuous or discontinuous period of time. Occurrences have participants acting in various roles. For example, a purchasing occurrence has participants acting in the roles `purchaser`[1], `purchased`, and `seller`. The occurrence "Brian's selling of his book" can be considered as "an entity (`c1`) named 'Brian' participates as `seller` in an occurrence (`o1`), classified by some authority as a sale/purchase, with an entity (`c2`), classified by some individual or institution as a book, participating as the `sold` item". The notion of individuated occurrences was arguably pioneered by the philosopher Donald Davidson [9, 22] who suggested that events (similar to what we call 'occurrences') such as 'the killing of Caesar by Brutus' are identified particulars[2]. We have implemented a formal system inspired in part by Davidsonian event semantics to represent and reason about commercial occurrences.

Four simple occurrences are illustrated diagrammatically in *Figure 2* below: an occurrence (`o1`) of Brian purchasing a particular book, an occurrence (`o2`) of Brian owning that book, an occurrence (`o3`) of Walt purchasing the book from Brian, and an occurrence (`o4`) of Walt owning the book.

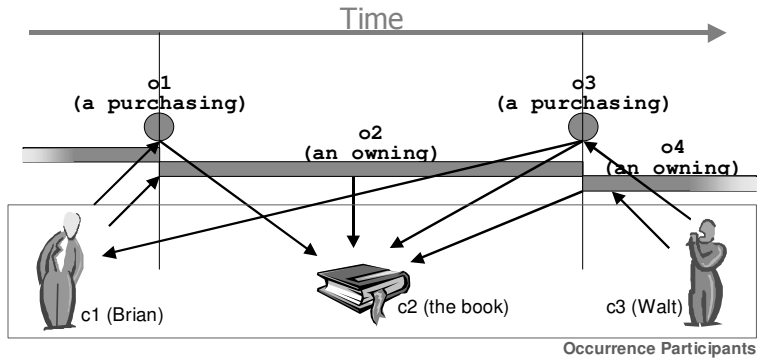

Figure 2 – Typical Commercial Occurrences

[1] To highlight role names in occurrences, we will frequently (though not always) show role names in `Courier` font.

[2] The use of the definite article 'the' in the phrase '*the* killing of Caesar by Brutus' was, amongst other factors, taken by Davidson to indicate that a specific identifiable entity was being referred to by the phrase 'the killing of Caesar by Brutus'.

Occurrences may be represented in a tabular form which defines the occurrence identifier, the participants in the occurrence, and their respective roles. Our motivation for choosing this particular participant-occurrence-role implementation schema is that it provides a simple, convenient, and uniform means of storing and retrieving *occurrences* - and consequently, as we shall see later, it provides a convenient and uniform means of storing and retrieving *queries* and *policies (contractual provisions)* – in traditional relational databases or tabular data structures. *Table 1* below shows the tabular representation of the occurrence `o3` which is a purchasing of a particular book by Walt, from Brian.

Participant	Occurrence	Role
c1 (named 'Brian')	o3 (a purchasing)	seller
c2 (classified as a book)	o3	bought
c3 (named 'Walt')	o3	buyer

Table 1: Representing a purchasing occurrence "Walt buying the book from Brian"

We choose to use domain- or application-specific roles[3], rather than merely generic thematic roles, in the representation of occurrences. While generic thematic roles – such as 'agent', 'patient', 'instrument', 'source', and 'destination' – are often helpful and are commonly used in grammatical description and knowledge representation in artificial intelligence [1, 12, 23, 26], they have been criticized. Davis, for instance, argues that the thematic role of a participant in an event occurrence may be difficult to determine or ambiguous [10]. Reliance solely on thematic roles is therefore problematic as thematic roles do not always uniquely identify participants in a given role in a commercial occurrence. For example, in 'Walt buys the book from Brian'[4], both Walt and Brian can be construed to be in the 'agent' thematic role of the purchase occurrence, as Walt is doing the buying and Brian is doing the selling. Using the domain-specific roles 'buyer' and 'seller' allows us to avoid this vagueness.

Davidson [9] comments that occurrences (what he terms 'events') may fall under multiple descriptions. In our example, the phrases 'the *buying* of the book by Walt from Brian', 'the *purchasing* of the book by Walt from Brian', and 'the *selling* of the book by Brian to Walt' could all be used to refer to the occurrence `o3`; the use of different descriptions such as 'buying', 'purchasing' or 'selling' should not distract us from noting that in each case we are referring to the same underlying occurrence. For this reason, we prefer to name occurrences using the pure identifier `o3` because a naming such as `buying3` or `selling4` would undesirably tie the description to the occurrence identifier. As we shall see later, occurrences can be multiply classified[5], and the classification of the occurrence may occur retrospectively based on the attributes of the occurrence.

It is worthwhile to point out at this stage that, unlike traditional frame-based representations, in the occurrence-based representation proposed, the semantics arise from the classification of the occurrence (via a classifying event) rather than from the role-names. The role-names are primarily to distinguish between different roles. System decisions generally depend on the classification of the occurrence, rather than on the names of the various roles involved in the occurrence. We shall see later, in the section *Representing and Storing Classifications*, how occurrences are classified.

4 Representing and Storing Names

The special treatment of symbols (names) is useful as it provides a strong conceptual separation between the symbols and the items they represent. For instance, the symbol 'Brian' (identified as symbol `s4` and stored in the Symbol table) is conceptually distinct from the concept named Brian (identified as `c1`). The conceptual separation between the symbol and the concept to which it refers is implemented through naming occurrences, which are occurrences of the form '[individual or institution, a `namer`] naming [concept, a `named` item] [symbol,

[3] Jurafsky and Martin [17] refer to domain- or application-specific roles as 'deep roles'. The names of the deep roles are derived from the verb – e.g. the verb 'buying', as in 'Walt buying the book from Brian' yields the roles 'buyer', 'bought', and 'seller'.

[4] In the reading we intend for 'Walt buys the book from Brian', consider 'buys' to be referring to a single, particular occurrence of purchasing, which we can denote here by occurrence identifier `o3`. Other readings are possible.

[5] Indeed occurrences may be given different classifications according to the same or different institutions over time.

a name]'. The Church of England's naming of c1 as 'Brian' is depicted in *Table 2* below.

Participant	Occurrence	Role
c1 (Brian)	o4	named
s4 (the symbol 'Brian')[6]	o4	name
c4 (the Church of England)	o4	namer

Table 2: Representing a naming occurrence "The Church of England names c1 'Brian'"

This representation permits different individuals or institutions to assign different names to the same concept. In the case of two or more individuals assigning the same name to a concept, we can resolve the ambiguity by qualifying the name with the name of the namer, or perhaps with the time or context of the naming.

5 Representing and Storing Queries

A query is a request for items fitting certain criteria. A query can be viewed as an occurrence of querying, where the criteria specified play various roles in the querying occurrence. It is important to distinguish between a *query* and its *results*. The query (i.e. query *definition*) may be referred to using its occurrence identifier – e.g. o5 – whereas the *results* of the query are referred to using the corresponding query resolution identifier for the query, which we will indicate using the occurrence identifier for the query preceded with a question mark – e.g. ?o5. To illustrate the importance of this distinction, consider an occurrence of "rich people buying caviar". If we consider o5 as being the query 'rich people' it would clearly be incorrect to state the occurrence as "o5 buying caviar" since it is not the query that is buying caviar but the items returned by the query; that is, we are *not* predicating something of the *query*, but rather of the *results returned* by the query. The correct rendition of the occurrence is therefore "?o5 buying caviar" which reads "[the results of the query o5] buying caviar".

We currently define algebraic, alphabetic, set-theoretic, occurrence-related, conditional, and nested queries. The following sections explain each of these query types and their structure in terms of roles (criteria).

5.1 Algebraic Queries

Three basic types of algebraic query are catered for: equality, strictly less than, and strictly greater than. The three basic types of algebraic query are represented as shown in *Table 3* below, which depicts the algebraic queries 'x = 6.2', 'x < 7', and 'x > 9' respectively (where x is the solution of the query). o1, o2, and o3 in the table below are occurrences of querying.

Participant		Occurrence	Role
6.2	(n1)[7]	o1	algebraic-equals-criterion
7	(n2)	o2	algebraic-less-than-criterion
9	(n3)	o3	algebraic-greater-than-criterion

Table 3- Representation of Basic Algebraic Queries: Equality, Strictly Less Than, and Strictly Greater Than

Queries of the form 'greater than or equal', 'less than or equal', 'between' (with inclusive or exclusive upper and lower limits), and discontinuous ranges are represented using combinations of basic algebraic queries and set-theoretic queries described below. For example 'greater than or equal to' is represented as a union of a strictly greater than query and an equal to query, and a 'between' query with exclusive upper and lower limits is represented as the intersection of a strictly less than query involving the upper limit and a greater than query involving the lower limit.

[6] Data-type restrictions suggest that it is undesirable to store symbols and numbers in the table which stores occurrences in participant-occurrence-role form. Symbols and numbers are therefore stored in separate Symbol and Number tables. The former simply contains symbol-identifier and symbol-text columns, and the latter number-identifier and number-value columns.

[7] As mentioned earlier, due to data-type restrictions, numbers are stored in the separate Number table. For readability we show the numbers directly; however, what appears in this column is really the number identifier: e.g. n1, n2, n3 where n1=6.2, n2=7, n3=9.

5.2 Alphabetic Queries

Alphabetic queries are catered for in a similar manner to algebraic queries, though alphabetic queries of course cater for alphabetic comparison rather than numeric comparison. For the sake of brevity, the detailed structure of alphabetic queries is shown in the *Appendix*.

5.3 Set-Theoretic Queries

Set-theoretic queries for union, intersection, difference, identification, universal set, empty set, identification, negation, and cardinality (counting) are supported. For the sake of brevity, only union and negation are discussed in detail here; other forms of set-theoretic query are explained in the *Appendix*.

Union

`?o1` ∪ `?o2` ∪ … returns the items in *any* of the sets `?o1`, `?o2`, etc. where `?o1`, `?o2`, etc. are the sets resulting from the resolution of the queries `o1`, `o2`, etc. In all of the examples to follow, the question mark '`?`' preceding the querying occurrence identifier is used to refer to the *resolution* of the query, rather than the query *definition* itself. The criteria (arguments) to the union query can be referred to as 'uniands'. The representation of a general union query is as shown in *Table 4* below:

Participant	Occurrence	Role
?o1 (results of query o1)	o4	uniand
?o2 (results of query o2)	o4	uniand
…	…	…

Table 4- Representation of a Union Query

Negation

A negation ('not') query can be represented using the universal set as the differor and the negated set as the differand. For example, the query 'not rich pensioners' can be represented as shown in *Table 5* below:

Participant	Occurrence	Role
Universal Set	o4	differor
'rich pensioners'[8]	o4	differand

Table 5- Representation of the Query 'not rich pensioners'.

5.4 Occurrence Related Queries

Occurrence-related queries are queries that are solved by specifying any two of the participant, occurrence, and role columns and solving for the third. Occurrence-related queries look for identifiers in matching participant-occurrence-role triples; the values (or set or range of values) of two columns are specified in the criteria and the results are the contents of the third column in all rows where the two specified columns match. There are naturally three types of occurrence-related query:

- An *occurrence query* specifies the participant and role and returns all occurrence-identifiers with this participant in that role.
- A *participant query* specifies the occurrence and role and returns all participant-identifiers in this occurrence with this role.
- A *role query* specifies the occurrence and participant and returns all role-identifiers for the participant in the occurrence.

The representation and resolution of occurrence-related queries is described in depth in the *Appendix*.

5.5 Conditional Queries (Side-Conditions)

It is frequently the case that a query should only return results when certain side-conditions are met. These side-conditions evaluate to true or false; an item is only returned by (and, conversely, covered by), a conditional query if the item satisfies the main criteria of the query and the side-condition also evaluates to true. The distinction between main criteria, which relate directly to the attributes of an item, and side-conditions, which relate to the situation in which

[8] What is stored here is the identifier of the results of the query 'rich pensioners' (e.g. '`?o5`') rather than the text 'rich pensioners'. The text 'rich pensioners' which represents the underlying query 'concepts classified as rich, intersection concepts classified as pensioners' is used merely for readability.

the query is performed, is important. For instance, let us assume that the occurrence-store stores the following occurrences: 'John pays $2 at 3pm', 'John pays $4 at 7pm', and 'John pays $5 at 8pm'. The query 'payments by John *after 6pm*' could have two interpretations, each returning different results:

- Interpretation as a pure occurrence-related query, where 'after 6pm' is a main criterion of the occurrence rather than a side-condition: Effectively, this interpretation of the query could be paraphrased as 'payments by John where the payment occurs after 6pm'. This query returns 'John pays $4 at 7pm' and 'John pays $5 at 8pm', irrespective of the time at which the query is evaluated.
- Interpretation as a conditional query, where 'after 6pm' is a side-condition: Effectively, this interpretation of the query could be paraphrased as 'payments by John when the current time is after 6pm'. This query returns nothing when the query is evaluated before 6pm, and returns 'John pays $2 at 3pm', 'John pays $4 at 7pm' and 'John pays $5 at 8pm' when the query is evaluated after 6pm.

A conditional query can be represented by combining a regular query with a side-condition that evaluates to true or false. As shown in *Table 6* below, the side-condition is merely a query-resolution-identifier (in this case `?o2`); the mechanism for determining whether a side-condition is true or false is explained in the *Appendix*.[9]

Participant	Occurrence	Role
?o1	o6 (conditional-query)	main-query criterion
?o2	o6	side-condition criterion

Table 6- Representation of a Conditional Query

[9] To cater for side-conditions which involve checking the system-clock (e.g. 'after 6pm') we would treat this as a query which returns the system time where the system time is greater than 6pm; the query would return nothing if the system time was less than 6pm. Clearly such a query involves invoking a method which interrogates the system clock. Queries which involve method invocations are not dealt with in this paper, but nevertheless may often be necessary for certain types of side-condition.

As we shall shortly see in Section 7.4 on *Conditional Duties*, which discusses conditional queries in more depth, conditional queries are needed to represent conditional obligations.

5.6 Nested Queries

Queries can be nested within each other. Nesting queries allows us to combine queries in order to specify complex criteria that are not otherwise expressible. Set-theoretic queries, occurrence-related, and conditional- queries can nest queries of any other type within them. As the participants in the criteria used for set-theoretic, occurrence-related, and conditional queries are queries themselves (or more specifically, query resolution identifiers associated with specific queries), the representations shown above and in the *Appendix* can be used to store nested queries. The next section gives an example of storing a nested query.

5.7 Storing Queries

For the purposes of representing policies and contracts and determining which policies (contractual provisions) apply, queries need to be stored rather than merely answered and discarded as is typically the case with conventional relational databases. Consider the business policy "The department head requires that the librarian must be notified of purchases of books". This business policy can be viewed as a contractual requirement and can be considered as an occurrence of obliging (say, `o20`) where:

- the participant in the role `obliger` is the department head,
- the condition under which the obligation holds is "a *purchasing of a book* has occurred", and
- what is `obliged` is occurrences fitting the description "*a notifying to the librarian of the purchase*".

Two queries are important here, each of which needs to be stored in order to store the policy as a whole:

1. The query "purchasings of books": this can be read as "occurrences of purchasing where the item in the role `purchased` is classified as a book".

2. The query "notifyings to the librarian of the purchase": this can be read as "occurrences of notifying where the person in the role `notified` is the librarian".

These queries, and the occurrence of obliging (`o20`) which links them, need to be stored for a number of reasons:

1. The query "purchasings of books" needs to be stored so that, upon occurrence of an event, we can determine whether the event occurrence is covered by the query "purchasings of books" – that is, whether the occurrence can be described as a "purchasing of a book" – and we can consequently determine which policies cover the occurrence. We can determine which policies cover the occurrence because we can determine which policies are linked to the stored query "purchasings of books" which we have determined, via database look-up, covers the occurrence. In this case, we determine that there is an obligation policy (which is an occurrence of 'obliging') that is linked to the stored query "purchasings of books", because something must be done when an occurrence fitting the description "purchasings of books" occurs.
2. The query "notifyings to the librarian of the purchase" needs to be stored so that, firstly, we can determine what courses of action can be taken in order to satisfy the obligation, and, secondly, we can determine whether a given course of action satisfies the description of the occurrences obliged under the obligation. For instance, an occurrence `o23` that is an emailing of the details of the purchase to the librarian, can clearly also be described as a "notifying to the librarian of the purchase", and would consequently satisfy the obligation, as would an occurrence `o25` that is a faxing of the details of the purchase to the librarian. We say that the occurrence `o23` (or `o25` depending on which implementation mechanism is chosen) can be *described as* a "notifying to the librarian of the purchase" because the occurrence is covered by the stored query "notifyings to the librarian of the purchase".

Now that we have motivated why it is that queries must be stored, let us turn to the means by which queries are stored. It is clear from the descriptions above of the various types of query that storing a query in an occurrence-centric database is simply a matter of storing the query-occurrences with their relevant criterion-types (that is, roles in the query occurrence) and actual criteria (that is, participants in the query occurrence).

Following is an example of query storage for a simple nested query. The query "*select concepts named 'Brian'*" is a nested query consisting of an alphabetic-equals query (to identify the symbol 'Brian'), nested inside an occurrence query (to select naming occurrences in which 'Brian' is the name given), which in turn is nested inside a participant query (to select concepts being named in the afore-mentioned naming occurrences). Specifically, the query "*select concepts named 'Brian'*" could be regarded as: "select participants in the role '`named`' in occurrences in the set (select occurrences where (select symbols equal to 'Brian') are in the role '`name`')"; the brackets indicate the various levels of nesting. The query would be represented using occurrences as shown in *Table 7* below, which captures the full structure of the query.

Participant	Occurrence	Role
'Brian' (s4)	o10 (an alphabetic equals query)	alphabetic-equals-criterion
?o10 (results of query o10)	o11 (an occurrence query)	participant-criterion
the role 'named'	o11	role-criterion
?o11 (results of query o11)	o12 (a participant query)	occurrence-criterion
the role 'name'	o12	role-criterion

Table 7: Storage of the nested-query "select concepts named 'Brian'"

The ability to store queries also means that we can use queries to look-up other (stored) queries. The ability to look-up queries that return results which fit certain criteria is useful for finding covering-queries – that is, the ability to analytically find which queries cover a certain item. This in turn is useful for determining which policies cover a particular item, and for interrogating a contract to determine what items and eventualities the contract covers. We describe the mechanism for finding covering-queries, and its use in finding the policies applicable to an item, in more detail in Section 5.9 on *Determination of Covering-Queries*.

5.8 Query Resolution

We may wish to locate, reconstruct, and resolve queries that are stored in policies, or we may wish to resolve non-stored queries such as "tell me what you've got for sale" or "tell me what happened from the audit" against the rows stored in the occurrence store. A stored query can be resolved against occurrences stored in a relational database by locating and reconstructing the query and executing it against the database using SQL. Non-stored queries can be parsed and resolved directly. For in-memory data stores, which are also supported by EDEE, EDEE executes the query by locating-and-reconstructing the stored query or by directly-parsing the non-stored query and then resolving it using appropriate Java set operations on tabular, indexed, in-memory data structures. The mechanism for query resolution in all cases is predominantly a tabular lookup combined with operations on sets of identifiers, and is described in more detail in the following paragraphs.

As an example of query resolution, the query "*select concepts named 'Brian'*" (which is a nested query as described in *Storing Queries* above), would be resolved as follows:

1. Alphabetic-equals query: select the symbol identifier for the symbol alphabetically equal to the symbol 'Brian'. Here we specify our alphabetic-equals-criterion as 'Brian' and, given *Table 9* below, we find that symbol `s4` satisfies this criterion.

Symbol Identifier	Symbol Text
s4	'Brian'
s5	'Niki'
s6	'Peter'

Table 8: Mechanism used to find the symbol alphabetically equal to 'Brian'
(See footnote 6)

2. Occurrence query: select the occurrences (of naming an individual with a name) in which the name given was the symbol 'Brian' (`s4`). Here we specify our participant-criterion as `s4` (the symbol identifier for 'Brian', returned by the alphabetic-equals query in step 1 above) and we specify our role-criterion as the role 'name'. Given *Table 9* below, we find that occurrence o2 satisfies this query.

Participant	Occurrence	Role
c5 (the Church of England)	o2	namer
c1	o2	named
(the symbol) 'Brian' (s4)	o2	name

Table 9: Mechanism used by occurrence query for finding occurrences in which the name given was 'Brian' (`s4`)

3. Participant query: select the participants who are being named in the previously found occurrence(s). Here we specify our role-criterion as the role 'named' and our occurrence-criterion as the occurrence `o2` (which has the occurrence-identifier found by the query in step 2 above). Given *Table 10* below, we find that concept `c1` satisfies this criterion.

Participant	Occurrence	Role
c5 (the Church of England)	o2	namer
c1	o2	named
(the symbol) 'Brian' (s4)	o2	name

Table 10: Mechanism used by participant-query for finding participants in occurrence `o2` (a naming 'Brian')

5.9 Determination of Covering-Queries

Finding covering-queries is the reverse of query resolution. In query resolution, we begin with a defined query and return all the results that match the specified criteria. When finding covering-queries we begin with an item, and determining which queries cover the item. The item may be a regular concept or it may itself be a query, in which case we can still determine analytically which other queries completely cover the results of that query.

Figure 3 below illustrates the complementary nature of query resolution and determination of covering-queries. Assume we have the occurrences 'Walt paying Ken', 'Walt paying Brian', and 'Peter paying Brian' stored in the occurrence store. Further, assume we have the queries 'Walt paying' and 'Brian

paid' stored in the occurrence store. Using the schema defined earlier the queries 'Walt paying' (i.e. 'select paying occurrences where the payer is Walt') and the query 'Brian paid' (i.e. 'select paying occurrences where the paid person is Brian') would each be stored as combinations of nested occurrence-queries. Turning now to **query resolution**, the query 'Walt paying' would return the occurrences 'Walt paying Ken' and 'Walt paying Brian'. Similarly, the query 'Brian paid' would return the occurrences 'Walt paying Brian' and the occurrence 'Peter paying Brian'. If we consider the reverse, **determination of covering-queries**, we see that the occurrence 'Walt paying Brian' is covered by both the queries 'Walt paying' (i.e. 'payings where Walt is the `payer`') and 'Brian paid' (i.e. 'payings where Brian is the person `paid`'). We say that an occurrence (or indeed any item) *fits a description* if it is *covered by a stored query*.

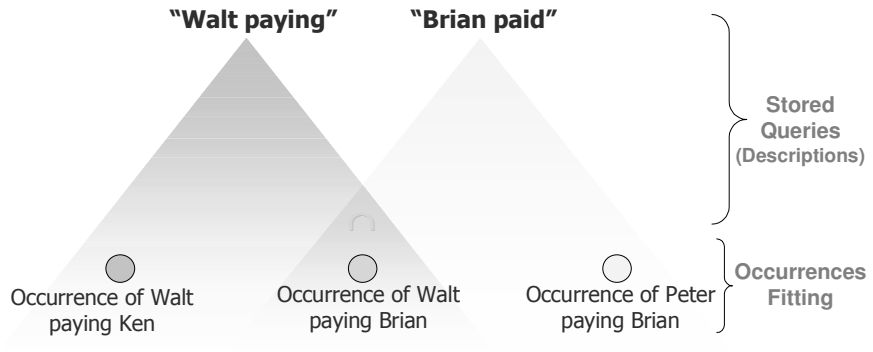

Figure 3 – Occurrences Fitting a Description (Covered by a Stored Query)

To give an example of determining covering-queries, assume we had the following queries stored:

- Query occurrence `o1`, representing 'x < 6'
- Query occurrence `o2`, representing 'x < 7'
- Query occurrence `o3`, representing 'x > 9'

Given the number 'x=6.2' it is straightforward to determine that this number is covered by the query `o2` ('x < 7') only. To determine this we simply use the rule "a number is covered by numeric-less-than queries where the 'less-than' criterion is greater than the number, and by numeric-greater-than queries, where the 'greater-than' criterion is less than the number". This rule is easily formulated as a query which returns the covering-queries for that number.

Given the query 'x<5' it is easy to determine that this query is covered by the queries `o1` ('x < 6') and `o2` ('x < 7'). To determine this we simply use the rule "a numeric-less-than query A is covered by numeric-less-than queries where the 'less-than' criterion is greater than the 'less-than' criterion of A". Again, this rule is easily formulated as a query which returns the covering-queries for that numeric-less-than query.

Given that there are a broad range of query types (algebraic, alphabetic, set-theoretic, occurrence-centric, and conditional queries) there is a complex set of queries which can be used to determine which stored queries cover a given item or query. For instance, to determine which union queries cover a given query we use the rule "a union query A covers a query B where any 'uniand' criterion of A covers B". To determine which conditional queries cover a given query A we find those conditional queries which have the query A, or a query which covers A, as the main-query criterion, and whose side-condition criterion evaluates to true. The complete set of rules relating to determination of covering-queries is quite lengthy, and has been omitted from this paper for the sake of brevity.

As mentioned previously, the ability to determine which queries cover a given item or query is essential for determining which policies apply to a given item or occurrence. For example, let us take the policy 'AuctionHouse PLC prohibits bids over $100', which is represented as shown in *Table 11* below:

Participant	Occurrence	Role
c1 (named 'Auction House PLC')	o4 (a prohibiting)	prohibitor
?o2 (results of the query/description 'biddings over $100')[10]	o4	prohibited (occurrences)

Table 11- Representation of 'AuctionHouse PLC prohibiting bids over $100'

We show in the representation above that what is prohibited is any occurrence fitting the description (i.e. query) 'biddings over $100'. Attempting to add the bidding occurrence 'Steve bidding $150' to the database is clearly a violation of this policy as the occurrence is covered by the query 'a bidding over $100' and is consequently covered by the prohibition.

6 Representing and Storing Classifications

Classifications can be represented using queries and classifying occurrences. In the initial purchasing example demonstrated in *Table 1* above we saw that a participant in an occurrence is often specifically identified by an identifier. However, it is also possible that the *participants in an occurrence may be identified by a query which returns a set of identifiers*. This mechanism of using queries to specify participants in an occurrence is useful for specifying the participants in an occurrence of classifying. Classifying occurrences take the form '[individual or institution, the `classifier`] classifying [concept or set of concepts, the `classified` item(s)] as [concept, the `class`].' The class[11] is then usually named using a naming occurrence. For example we could represent 'accounts with balance > $100 for longer than 60 days are classified as overdue' using the classifying and naming occurrences shown in *Table 12* below.

[10] For brevity, the storage of this query is not shown here. The query would involve selecting the bidding occurrences where the price bid (i.e. amount in the 'bid' role) is greater than $100.

[11] More accurately, the participant in the `class` role in the classifying occurrence.

Participant	Occurrence	Role
?o1 (the results of the query/description 'accounts with balance > $100 for longer than 60 days'[12])	o4 (a classifying)	classified concepts
c2 (the concept *overdue*)	o4	class (classification)
c4 (the company making the classification)	o4	classifier (institution)
c2 (the concept *overdue*)	o5 (a naming)	named
s8 (the symbol 'overdue')	o5	name
c4 (the company that calls this 'overdue')	o5	namer

Table 12: Representing a classifying occurrence "Accounts with balance > $100 for longer than 60 days are classified as overdue"

[12] This is a complex query that could be formulated with a combination of intersection, occurrence, and algebraic queries.

7 Representing and Storing Contracts

In order to represent contracts, formalisms must be provided to represent contractual provisions – that is, the descriptive and prescriptive policies specified in the contract. The provisions of a contract include policies that define terminology and its interpretation, regular and conditional rights and duties of the parties, and authority (permissions and powers). We must be able to specify what constitutes a breach of contract, and how contracts are interpreted in terms of various governing institutions and systems of law. Furthermore the act of stating the terms of an agreement must be separated from an act that make those terms binding. The formalisms needed to represent these requirements in an occurrence centric manner are explained in the following sections.

7.1 Terminology and the Interpretation of Terms

Contracts commonly define terms[13] which are intended to be construed in a particular sense. For example, a contract may define the terms: 'you', 'yours', 'the Company', and others. Terminology is defined so as to explain how the parties name and classify items and occurrences being referred to by the contract. The ability to define terminology reduces ambiguity, improves consistency, and makes the contract more concise [27]. Defined terminology can be easily implemented using a combination of classifying and naming occurrences. Earlier (*Table 12* above) we defined the term 'overdue' according to a definition specified by a company; the term 'overdue' could equally well be defined according to a definition specified by a contract.

[13] Here we mean terms in the specific sense of 'words and phrases', not in the sense of 'provisions of the agreement'.

The interpretation of a term is all the results returned by the query associated with the defined term (i.e. the query associated with the classification – the term is simply the name of the class associated with the classifying occurrence). In the case of the term 'overdue' defined in *Table 12*, 'overdue' is the `name` of the `class c2` associated with the classifying occurrence `o4`. The interpretation of 'overdue' would be `?o1` – that is the results of the query `o1` – which holds the role `classified` in the classifying occurrence `o4`. This means that an account with a balance of $125 for 72 days would be interpreted as 'overdue', whereas an account with a balance of $20 for 30 days would not.

The use of classifying occurrences allows us to construct ontologies in a far more flexible manner than is possible with object-oriented languages. Object-oriented languages expect the static class hierarchy to be created first – objects are instantiated according to fixed class definitions; this inhibits evolution and precludes multiple interpretations. Our approach allows alternative classifications and retrospective reclassifications, through the simple mechanism of adding classifying occurrences. Furthermore, unlike object-oriented class structures, classifying occurrences are subjective, allowing for multiple classifications by different parties or by the same party over time. Subjectivism is common in the interpretation of contracts and essential in the application of different systems of law to contracts since each system of law applies its own subjective classifications. The subjective classification imposed by the highest court of the land will be the ultimate classification applied during disputes about the contract, but other classifications may be used by the parties during performance of the contract. We explain more about this in Section 7.7 on *Institutional Interpretations and Systems of Law*. We turn now to the representation of rights.

7.2 Rights of Parties

Hohfeld [15] explains that the word 'right' has multiple senses, inter alia:

- One sense of 'right' is the correlative of 'duty' or 'obligation'. For example, if X has a duty (obligation) to deliver specific goods to Y, then Y has the corresponding right (entitlement) to receive delivery of the goods from X.
- Another sense of 'right' is synonymous with 'power'. A party X has the right (i.e. power) to sell an item if they have the ability to bring

about certain occurrences which constitute a valid sale; that is, if they have the ability to bring about certain occurrences that are classified as valid sales by a governing institution. The notion of power is defined in the section on *Authority* below.

7.3 Duties of Parties

The first sense of 'right' as an 'obligation' or 'entitlement' can be represented in the form: [obliger] obliges [occurrence(s) fitting a description]. *Table 13* below represents the duty of Brian to deliver a particular book to Walt within 5 days. This duty is equivalent to the entitlement of Walt to have the particular book delivered to him. Notice that it is only the *first* delivering occurrence fitting the description that is obliged[14]. This means that once a suitable delivery occurs, the obligation is fulfilled and nothing else is obliged. When the query describing the obliged occurrences (?o1 in this case) can return no more results irrespective of what is added to the database, the query is said to be 'full', and the corresponding obligation is said to be 'fulfilled'.

Participant	Occurrence	Role
c3 (named 'Walt')	o4 (an obliging)	obliger
?o1 (the results of the query/description 'the first occurrence of delivering of the particular book by Brian to Walt within 5 days')	o4	obliged occurrences

Table 13: Representing the duty (obligation o4) of Brian to deliver the book to Walt

Violations of obligations are formalized in the section on *Breaches* below.

7.4 Conditional Duties

It is common for duties in contracts to be **conditional obligations** that come into force upon occurrence of a certain event. For instance, the duty of Brian to deliver is contingent upon Walt's paying for the book. The conditional nature of the obligation can be achieved by classifying the occurrence o4 as an 'obliging' (i.e. a valid and in-force obligation according to a certain party or institution) **only if** the conditions have been met. This is implemented using a classification of the results of a conditional query, as illustrated in *Table 14* below; the conditional query returns the occurrence o4 only if the conditions have been met, and consequently o4 is classified as an in-force obligation only if the conditions have been met. In the absence of being classified as an 'obliging' ('in-force obligation'), o4 is not an enforceable obligation. As mentioned earlier, the role names 'obliger' and 'obliged occurrence' (in *Table 13* above) have no semantics in the absence of an explicit classification of the occurrence with which these roles are associated (in *Table 14* below). So the occurrence o4 becomes an in-force obligation only when it is classified as such by a particular institution, and does *not* become an obligation solely by virtue of having the roles 'obliger' and 'obliged occurrences'.

Participant	Occurrence	Role
?o5 – this is a conditional query which: • when 'Walt has paid Brian' exists, returns the occurrence o4 – in *Table 13* above - which has 'Brian' as obliger and 'the first delivering of the book to Walt within 5 days' as the obliged occurrence • when 'Walt has paid Brian' does not exist, returns nothing	o6 (a classifying)	classified
c6 (an obliging / an in-force obligation)	o6	class
the parties to the contract[15]	o6	classifiers

Table 14: Representing a conditional obligation of Brian to deliver the book to Walt if Walt has paid for the book, using a conditional query in a classifying occurrence

A further example of a conditional obligation is: "You must pay a penalty of $100 if you pay late", or alternatively "You must, if you pay late, pay a penalty of $100". Here "paying a penalty of $100" is a description of an occurrence; the first occurrence fitting the description is what is obliged, but the obligation is only recognized if "you pay late" is true (i.e. occurs). The obligation is not

[14] Contrast this to permitted and prohibited occurrences, to be discussed in Section 7.5, where it is usually *all* occurrences fitting the description (rather than just the first *n* occurrences fitting the description) that are permitted or prohibited.

[15] Again, what is stored here is not the text 'the parties to the contract', but rather the query-resolution identifier ?o7, where o7 is the query that returns the parties to the contract.

recognized if you pay on time. *Table 15* below represents this conditional obligation.

Participant	Occurrence	Role
?o1 – the results of the query 'the first paying of a fine of $100'	o2 (has no type until classified by the 'classifying' occurrence o4)	obliged occurrences
?o3 – this is a conditional query which: • when 'you pay late' exists, returns the occurrence o2 in the row above • when 'you pay late' does not exist, returns nothing	o4 (a classifying)	classified
c6 (an obliging / an in-force obligation)	o4	class
the parties to the contract[16]	o4	classifiers

Table 15: Representing a conditional obligation to pay a penalty of $100 if you pay late

Secondary obligations that come into force upon violation of a primary violation are common in contracts. These can be represented as conditional obligations.

[16] Again, what is stored here is not the text 'the parties to the contract', but rather the query-resolution identifier `?o7`, where `o7` is the query that returns the parties to the contract.

7.5 Authority: Permissions and Power

The notion of authority can be considered in terms of the distinct notions of permission and empowerment. For instance 'the salesperson is authorized to sell items discounted by up to 50% of their regular sales price' could mean one or both of two things:

1. 'The salesperson is **prohibited** from selling items discounted by more than 50% of their regular sales price'. The representation of this prohibition is as depicted in *Table 16* below. Violating this prohibition would subject the salesperson to possible disciplinary action from the company. Violations are discussed in the section on *Breaches* below.

Participant	Occurrence	Role
c1 (the company)	o5 (a prohibiting)	prohibitor
?o1 (the results of the query/description "occurrences of selling where goods are sold at a discount of more than 50% of their regular price")	o5	prohibited

Table 16: Representing a company prohibiting a salesperson from selling goods at a discount of more than 50% of their regular price

2. 'The salesperson is **empowered to** sell (**contractually capable of** selling) only items discounted by less than 50% of their regular sales price'. The representation of this power is as depicted in *Table 17* below. In terms of UK commercial law this means that, provided the company has made reasonable efforts to communicate this constraint on the agent's authority to the consumer, any attempt by the salesperson to sell an item discounted to more than 50% of the regular sales price does not constitute a valid selling occurrence in terms of the company's policies, nor in the eyes of the Court and is consequently not classified as such. In contrast, if there is an occurrence of attempting to sell the item discounted by less than 50% of the regular sales price by the salesperson then the sale is valid and the occurrence is so classified by the Court.

Participant	Occurrence	Role
?o1 (the results of the query/description "occurrences of selling where goods are sold at a discount of less than 50% of their regular price")	o6 (a classifying)	classified
c1 (the company making this classification)	o6	classifier
c4 (a valid selling)	o6	classification

Table 17: Representing the power of a salesperson to sell goods (enact a valid occurrence of selling) only when they are discounted by less than 50% of their regular price

In short, the **power** of an individual to bring about an occurrence means that the occurrence, provided it fits certain criteria, will be construed as (i.e. classified as) an occurrence of a certain type by a governing institution. Jones and Sergot [18] formalize this with the use of a 'counts as' connective; we make use of classifying occurrences which have a similar effect. For instance, an occurrence is classified by the Commercial Law Court as a valid sale (occurrence of 'selling' according to the courts definition of 'selling') if the parties have agreed precise terms and the seller is authorized to sell the item.

In the example "*Salespeople* may not sell goods discounted by more than 50%" notice that '*salespeople*' is a query which returns the set of individuals classified as salespeople. If John Smith is a salesperson, the query resolves to a set of results that includes John Smith, yielding the policy "*John Smith* may not sell goods discounted by more than 50%". Of course, determination of covering-queries – the reverse of query-resolution – may be used to check which policies apply to John Smith. A determination of covering-queries for *John Smith* would reveal that he is covered by *salespeople*, and that *salespeople*, are in turn, mentioned by the nested query (in this case, an occurrence-description) '*salespeople selling goods discounted by more than 50%*', which in turn is covered by the prohibition against such occurrences (i.e. is covered by the 'prohibiting' occurrence that takes the afore-mentioned occurrence-description in the 'prohibited' role). The determination of covering queries is therefore an essential tool in interrogating the contract in order to ascertain which provisions of the contract apply. If there are multiple conflicting provisions we obviously have to choose which one to *apply*: that is, whether a given permission or specific obligation overrides a contradictory prohibition or vice versa. We may, for instance, need to decide whether a given occurrence in which something is obliged, holds as (that is, is classified by a certain institution as) a valid and enforceable obligation – occurrence of 'obliging' – in the case where there is a prohibition to the contrary.

7.6 Breach of Contract

Violations occur when:

- An obligation to perform an occurrence fitting a description has not been fulfilled by a certain deadline. In the case of the obligation of Brian to deliver a particular book to Walt within 5 days, the obligation is violated if there is no such delivery within the requisite time. Such breaches can be represented using the occurrence structure illustrated in *Table 18* below.
- A forbiddance (prohibition) against performing an occurrence fitting a certain description has been flouted. Such breaches can be represented using the occurrence structure illustrated in *Table 19* below, which gives the example of violating the prohibition against selling items at a discount of more than 50%.

Participant	Occurrence	Role
?o7 – the results of a conditional query which: • after the deadline and if there are no occurrences fitting the description of obliged occurrences (i.e. fitting the description in the 'obliged' role of the obligation o4[17] of Brian to deliver the book to Walt within 5 days), returns the obligation o4. • before the deadline, returns nothing.	o8 (a violating / breach)	violated
o8	o9 (a classifying)	classified item
c12 (a violation/breach, in terms of the contract)	o9	class

Table 18: Representing the violation of an obligation (o4[17]) to perform by a deadline (specifically, an obligation of Brian to deliver the book to Walt within 5 days)

Participant	Occurrence	Role
?o10 – the results of a conditional query which: • returns the prohibition occurrence o5[18] if there are any occurrences fitting the description of prohibited occurrences (i.e. fitting the description in the 'prohibited' role in the prohibition against selling items discounted by more than 50%). • otherwise, returns nothing.	o11 (a violating / breach)	violated
o11	o12 (a classifying)	classified item
c12 (a violation/breach, in terms of the contract)	o12	class

Table 19: Representing the violation of a prohibition (o5[18]) against selling items discounted by more than 50%

[17] Refer to *Table 13* for the definition of the obligation (obliging) occurrence o4.
[18] Refer to *Table 16* for the definition of the prohibition (prohibiting) occurrence o5.

7.7 Institutional Interpretations and Systems of Law

Occurrences such as classifying, naming, authorizing, and prohibiting each can take role-players such as the classifying party, naming party, authorizing party and prohibiting party. These role-players can be viewed as the institutions or individuals which make the classifications or define the norms. The ability to afford multiple institutional definitions allows contracts to be subjectively interpreted in terms of alternative institutional interpretations or systems of law which define their own norms relating to classifications, duties, prohibitions, and powers. For example, the lower courts may classify a company as being exempt according to certain contractual provisions and classification criteria set out by the judge of the lower court; the Appeals Court may make a different classification using its own criteria.

7.8 Proposed Terms versus Binding Agreements

It is important to recognize that the act of stating or proposing the terms of the contract does not make the contract binding. The interpretation and enforcement of the terms is provisional upon agreement being reached and recognized by the parties, or being recognized by the court in the event of any dispute. For instance, the stated term 'Brian must deliver the book to Walt' – restated as "Obliged is an occurrence of 'delivering of the book by Brian'" – applies (i.e. is classified as a valid 'obliging' and therefore an in-force obligation by a governing institution) only if a valid sale (selling) has been recognized. The terms and conditions of the sale agreement only come into force (are conditional upon) a valid 'selling' being recognized.

The reinterpretation of proposed terms as binding terms following an occurrence of agreeing can be implemented using the conditional query mechanism introduced earlier. The occurrence schemas used to implement the conditional nature of terms in general would appear as follows:

- [Governing institution] classifying [select occurrences in which something is in the role `obliged`, but only if there is an occurrence `o3` which pertains to the contract and is classified as a valid selling] as valid occurrence of 'obliging'.
- [Governing institution] classifying [select occurrences in which something is in the role `prohibited`, but only if there is an

occurrence o3 which pertains to the contract and is classified as a valid selling] as valid occurrence of 'prohibiting'.

- and so forth.

In each of the occurrence schemas listed in the bullet points above, notice that 'but only if there is an occurrence o3 which pertains to the contract and is classified as a valid selling' is a condition in a conditional query; the conditional query returns nothing if the condition is not satisfied, and consequently nothing is classified as a valid 'obliging' or 'prohibiting' by the governing institution if the condition is not satisfied (i.e. if there is no occurrence of 'selling[19]' for the contract).

8 Performing and Enforcing Contracts

An occurrence-based system performs and enforces contracts by:

- **determining currently applicable obligations**, by finding descriptions of obliged occurrences associated with occurrences currently classified as valid obligings (**diagnosis**)
- **fulfilling its own obligations** by triggering occurrences which it is capable of which fit the descriptions specified in its obligations (**liveness**)
- **avoid violating prohibitions** which cover it, by trying not to trigger occurrences which fit the descriptions specified in such prohibitions (**safety**)
- **monitoring fulfillment** of (primary and secondary) obligations and violation of prohibitions by users, through the determination of covering-queries (**detection**)
- **classifying** new occurrences appropriately according to defined norms, thereby causing certain classifications of occurrences (in line with powers) and suppressing certain classifications of occurrences (**prevention / immunity**)
- **fulfilling any secondary obligations** arising from unavoidable violations (**cure**)

[19] 'selling' is synonymous here with 'agreeing to the sale'.

9 Implementation and Motivation

Figure 4 – Implementation Architecture below shows a schematic architecture of an Edee occurrence store – the store is an active database as new occurrences added to the database are checked against existing stored queries and policies (contracts) and the contractually defined responses are accordingly triggered.

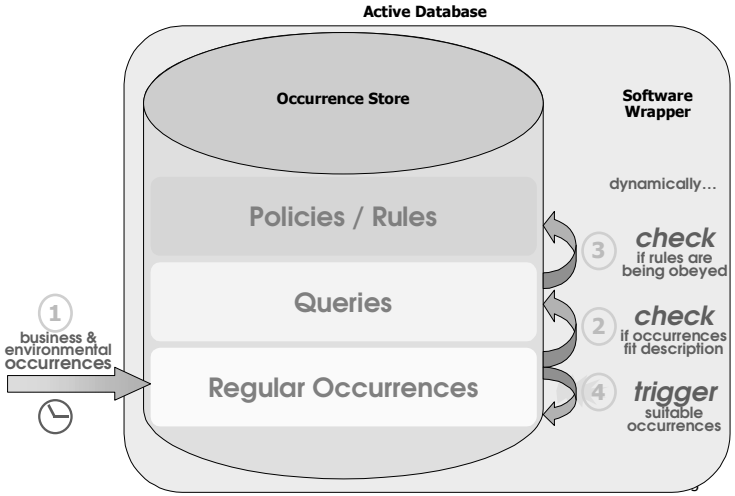

Figure 4 – Implementation Architecture

The simple participant-occurrence-role structure of all occurrences enables us to use a broad variety of back-end data stores to uniformly store occurrences, and consequently queries and policies. The EDEE environment has been successfully tested against Oracle 8.1.7, PostgreSQL 7.1, IBM DB/2 7.1, Microsoft SQL Server 7.0 (SQL Server 2000) and Microsoft Access 2000 running on SunOS, Red Hat Linux 6.2, Windows NT and Windows 2000. The ability to uniformly store queries and policies in a vendor-independent manner provides greater platform-independence than proprietary stored-procedures and triggers; furthermore databases that do not support stored procedures and triggers can still be used for query and policy storage. EDEE also implements its own in-memory data-store in Java which allows caching of results and

substantially more rapid retrieval and determination of covering-queries compared to on-disk data. Both on-disk and in-memory structures can be queried using the EdeeQL query language.

10 Related Work

Existing **policy approaches and languages** are primarily targeted at system or network management [11, 28, 20] or access control [13, 4, 14, 19] and are not capable of representing and enforcing e-commerce contracts and business policies. The expressiveness of each approach varies. For example OASIS [13], LaSCO [14], QCM [19] and PolicyMaker [4] can express only authorization policies, whereas Ponder [11] can express authorization and obligation policies as well as some basic descriptive policies through an associated name service. All adopt a traditional object-oriented framework, above which the policy notation is laid. COLOR-X [7], though not developed as a policy notation, is a conceptual modeling framework that incorporates various policy concepts such as the 'permit' and 'must' modalities. The end-goal of COLOR-X is the generation of *object-oriented code* (classes, attributes, and methods) from a conceptual schema or functional specification, whereas we propose the generation of various types of *occurrences-based policies* to be stored in an occurrence store.

A **commercial rules repository**, like *Business Rule Solutions Inc's RuleTrack* [6] can be used for recording and organizing rules, but is limited to storing textual descriptions of the rules, which are not machine-interpretable. **Commercial rules engines** include *Blaze Software's Advisor* [5], *Vision Software's Vision JADE*, *Usoft*, and *ILOG Rules*. **Web content personalization servers** like *ATG Dynamo*, *BEA Weblogic*, and *Microsoft Site Server 3.0* and *Commerce Server 2000* embed lightweight rules engines or rule generating scripts which typically generate VBScript or JavaScript source code; they implement content personalization rules via a combination of XML (for tagging) and hooks to an existing object model. Traditional rules languages are implemented above object-oriented or procedural fundamentals or as SQL triggers above relational database, and invariably use the Event-Condition-Action (ECA) model where events (occurrences) are statically typed. Being constrained by a static object model, they provide little or no support for subjective interpretation (classification) and naming of items and occurrences. Furthermore, traditional rule languages do not formalize the essential contractual notions of duty, authority, and power[20] as provided by EDEE. These notions, which according to Hohfeld [15] are among the fundamental legal conceptions, are essential constructs for representing legal relations between parties in contracts.

Most current **event monitoring services** [2, 3, 8, 21, 25] focus on event detection and neglect event actuation. GEM [21] has been integrated with the Ponder policy language, and CEA has been integrated with the OASIS access control policy language. In both cases, procedural or object-oriented languages are used as the basis. These event monitoring services allow composite events to be fired. However, aside from the notable exception of GEM, these event monitors provide no actuation facilities or operators for triggering structured patterns of events which would be necessary for workflow management in e-commerce applications. Event monitoring systems monitor for the occurrence of certain events and notify interested parties; in the absence of integration with a policy system, event monitoring systems cannot check which policies are applicable in the current circumstances.

Workflow languages, as exemplified by the Process Specification Language (PSL) specification [24], allow routing policies to be defined based partially on the fulfilment of obligations derived by the company from its contracts and internal procedures. However, workflow languages tend to neglect notions such as those of descriptive and prescriptive norms (classifying, obliging, prohibiting) that are crucial to coherent contract specification. Our occurrence-based policy language is sufficiently expressive to model the *join* and *split* primitives for process control typically provided by workflow products:

- A join (synchronization, or rendezvous) point involves *forbidding* the next (successor) process from starting until the condition (e.g. 'and', 'or', or complex condition) has been fulfilled. The successor process is then said to be 'released'. Some processes may be required to finish simultaneously, in which case the end of one process *obliges* the other process to end. A join is actually a DETECT-and-CONTROL mini-workflow that occurs

[20] Recall that the notion of 'power' is implemented by classifying occurrences fitting a certain description as being of a certain type according to a certain institution; this effectively empowers an actor to bring about a certain situation (legal relationship) according to that institution.

concurrently with some other processes. Detection (monitoring) involves determining which processes have started or finished.

- A split (selection) point involves selecting and *obliging* one or more successor processes to start (perhaps simultaneously), which may involve *forbidding* other successor processes from starting. A split is actually a SELECT-and-CONTROL mini-workflow that occurs concurrently with some other processes.

CONTROL (in the case of DETECT-and-CONTROL and SELECT-and-CONTROL mini-workflows) involves obliging other processes to start or finish, authorizing other processes to start or finish, or prohibiting other processes from starting or finishing.

11 Conclusion

We have described a scheme for representing commercial contracts using an abstraction known as the occurrence. Occurrences have participants acting in various roles. The participants in an occurrence may be explicitly specified using an identifier, or may be described using a query. Queries themselves can be stored as occurrences. Contractual notions such as defined terminology, duties, authority, powers, and breaches can be formalized in terms of occurrences of classifying, obliging, authorizing, and violating. Conditional obligations are supported. Furthermore, subjective interpretations in terms of alternative institutional interpretations and systems of law are supported.

Business application specifications are stated in terms of contracts between legal entities, and contracts between the computer system and its users. Contracts can be used to encode the policies governing the behavior of human agents and computerized systems. By providing a formal representation of the core semantics of commercial contracts and business policies through the use of occurrences, the occurrence-centric mechanisms implemented by EDEE constitute a powerful means of directly storing, interrogating, and executing business application specifications and electronic commerce contracts.

12 Acknowledgements

This research is supported by grants from the Cambridge Commonwealth Trust, the Overseas Research Students Scheme (UK), and the University of Cape Town Postgraduate Scholarships Office. Thanks are due to various members of the Opera Group at the University of Cambridge Computer Laboratory for helpful comments on the draft.

13 References

[1] Allen, J., *Natural Language Understanding: Second Edition*, Benjamin/Cummings, 1-41, 248 (1995).

[2] Bacon, J.M., J. Bates, R.J. Hayton, and K. Moody, "Using Events to Build Distributed Applications", in: **Proceedings of the IEEE SDNE Services in Distributed Networks and Environments**, Whistler, British Columbia, 148-155 (1995).

[3] Bacon, J.M., K. Moody, J. Bates, R. Hayton, C. Ma, A. McNeil, O. Seidel, and M. Spiteri, "Generic Support for Distributed Applications", *IEEE Computer*, vol. 33 no. 3, 68-76 (2000).

[4] Blaze, M., J. Feigenbaum, and J. Lacy, "Decentralized Trust Management", in: **Proceedings of the IEEE Conference on Security and Privacy**. Oakland, CA (1996).

[5] Blaze Software, "Blaze Advisor Technical White Paper" (2000), Available from: http://www.blazesoft.com/products/docrequest.html.

[6] Business Rules Solutions, LLC, "BRS RuleTrack" (2001), Available at: http://www.brsolutions.com/ruletrack.shtml.

[7] Burg, J.F.M., and van de Riet R.P., "COLOR-X: Object Modeling Profits from Linguistics", in: **Proceedings of the 2nd International Conference on Building and Sharing of Very Large-Scale Knowledge Bases (KB&KS'95)**, Enschede, The Netherlands (1995).

[8] Carzaniga, A., D.S. Rosenblum, and A.L. Wolf, "Design of a Scalable Event Notification Service: Interface and Architecture", Technical Report CU-CS-863-98, Department of Computer Science, University of Colorado, Boulder (1998).

[9] Davidson, D., *Essays on Actions and Events*, Clarendon Press, Oxford (1980).

[10] Davis, T., "Lexical Semantics and Linking in the Hierarchical Lexicon", PhD Thesis, Stanford University, Department of Linguistics, 17-69 (1996), Available at: http://www-linguistics.stanford.edu/~tdavis/index.html.

[11] Damianou, N., N. Dulay, M. Lupu, and M. Sloman, "Ponder: A Language for Specifying Security and Management Policies for Distributed Systems. The Language Specification. Version 1.11", Imperial College Research Report DoC 2000/1, Department of Computing, Imperial College of Science and Technology, University of London (2000).

[12] Expert Advisory Group on Language Engineering Standards (EAGLES), "Thematic Roles", (2000), Available at: http://www.ilc.pi.cnr.it/EAGLES96/synlex/node62.html.

[13] Hayton R.J., J.M. Bacon, and K. Moody, "Access Control in an Open Distributed Environment", in: **Proceedings of the IEEE Symposium on Security and Privacy**, Oakland, CA, 3-14, (1998).

[14] Hoagland J.A., R. Pandey, and K.N. Levitt, "Specifying and Enforcing Policies using LaSCO: the Language for Security Constraints on Objects", Department of Computer Science, University of California, Davis, Presentation and Position Paper at the Policy Workshop, Hewlett Packard Laboratories, Bristol, UK, (1999).

[15] Hohfeld W.N., *Fundamental Legal Conceptions as Applied in Judicial Reasoning*, Edited by Walter Wheeler Cook, Greenwood Press Publishers, Westport Connecticut (1978).

[16] ILOG Corporation, "Business Rules" (2001), Available at: http://www.ilog.com/products/rules/.

[17] Jurafsky D.S., and J.H. Martin, *Speech and Language Processing*, Prentice Hall, New Jersey, 499-543, 607-629 (2000).

[18] Jones, A.J.I., and M. Sergot, "On Power in e-Institutions", in: **Second International Workshop on Formal Models of Electronic Commerce (FMEC'00)**, Wharton School, University of Pennsylvania (2000).

[19] Kakkar, P., M. McDougall, C.A. Gunter, and T. Jim, "Credential Distribution with Local Autonomy", Department of Computer and Information Science, University of Pennsylvania (1999), Available from: http://www.cis.upenn.edu/~qcm/papers/autonomy.ps.gz.

[20] Koch, T., *Automated Management of Distributed Systems*. Dissertation for the degree of Doktor-Ingenieur at the Fachbereich für Elektrotechnik der FernUniversität Hagen, Shaker Verlag, Aachen, (1997).

[21] Mansouri-Samani, M., and M. Sloman. "GEM: A Generalised Event Monitoring Language for Distributed Systems", *IEE/IOP/BCS Distributed Systems Engineering Journal*, vol. 4 no. 2 (1997). Extended version of a paper presented at ICODP/ICDP '97 conference, Toronto, (1997).

[22] Martin, R.M., D. Davidson, R.J. Butler, and W.C. Salmon. "Proceedings of the Symposium on Events and Event-Descriptions at the University of Western Ontario Philosophy Colloquium 1966", in: **Fact and Existence**, ed. J. Margolis, Basil Blackwell, Oxford (1969).

[23] Palmer, F., *Grammar*, Penguin Books, (1984).

[24] Schlenoff C., M. Gruniger, F. Tissot, J. Valois, J. Lubell, and J. Lee, *The Process Specification Language (PSL) Overview and Version 1.0 Specification*, National Institute of Standards and Technology (2000), Available from: http://www.mel.nist.gov/psl/.

[25] Segall, B., D. Arnold, J. Boot, M. Henderson, and T. Phelps. "Content Based Routing with Elvin4", in: **Proceedings of the Australian Unix and Open Systems User Group Conference (AUUG2K)**, Canberra, Australia, (2000).

[26] Sowa, J.F., *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks/Cole. Pacific Grove, California, USA (2000).

[27] Thorpe, C.P., and J.C.L. Bailey, *Commercial Contracts*, Kogan Page, London (1999).

[28] Wies, R., "Using a Classification of Management Policies for Policy Specification and Policy Transformation", in: **Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management**, Santa Barbara, California, USA (1995).

14 Biographical Details

Alan Abrahams is a doctoral student at the University of Cambridge Computer Laboratory. His research interests include the use of event semantics for e-commerce application specification and development. He holds a Bachelor of Business Science degree with honours in Information Systems from the University of Cape Town, South Africa. Previously, he lectured systems analysis and design at the Department of Informatics, University of Pretoria. He was co-author of a paper on e-commerce application development which won the best paper award at the International Academy for Information Management 14th Annual Conference.

Dr Jean Bacon is a reader in distributed systems at the University of Cambridge Computer Laboratory and a Fellow of Jesus College, Cambridge. Under various EPSRC grants she has investigated open architectures for role-based access control in interworking services, and composite event architectures for asynchronous middleware. She leads the Opera Research Group with colleague Ken Moody. She is the author of Concurrent Systems (Addison Wesley) and Editor-in-Chief of IEEE Distributed Systems Online (http://computer.org/dsonline). Jean lectures on concurrent systems, operating systems, and distributed systems at the Computer Laboratory.

15 Appendix: Structure of Queries Not Shown In Main Body

This appendix describes in detail the structure of queries not shown in the main body.

15.1 Structure of Algebraic Queries

See page 4.

15.2 Structure of Alphabetic Queries

The three basic types of alphabetic query are represented as shown in *Table 20* below, which depicts the queries 'x = Brian', 'x < Niki', and 'x > Peter' respectively (where x is the solution of the query).

Participant	Occurrence	Role
'Brian' (s4)[21]	o1	alphabetic-equals-criterion
'Niki' (s5)	o2	alphabetic-less-than-criterion
'Peter' (s6)	o3	alphabetic-greater-than-criterion

Table 20- Representation of Basic Alphabetic Queries: Equality, Less Than, and Greater Than

As for algebraic queries, the basic types of alphabetic queries can be composed using set-theoretic queries.

[21] Again, due to data-type restrictions, symbols are stored in the separate Symbol table. For readability we show the symbols directly; however, what appears in this column is really the symbol identifier: e.g. `s4`, `s5`, `s6` where `s4`='Brian', `s5`='Niki', `s6`='Peter'.

15.3 Set-Theoretic Queries

Set-theoretic queries for union, intersection, difference, identification, universal set, empty set, and negation are supported.

Union See page 5.

Intersection `?o1 ∩ ?o2 ∩ …` returns the items in *all* of the sets `?o1`, `?o2`, etc. where `?o1`, `?o2`, etc. are the sets resulting from the resolution of the queries `?o1`, `?o2`, etc. The criteria (arguments) to the intersection query can be referred to as 'intersectands'. The representation of a general intersection query is as shown in *Table 21* below:

Participant	Occurrence	Role
?o1	o4	intersectand
?o2	o4	intersectand
…	…	…

Table 21- Representation of an Intersection Query

Difference `?o1 − ?o2` returns the items in the set `?o1` *but not* in (i.e. *excluding* those items in) the set `?o2` where `?o1` and `?o2` are the sets resulting from the resolution of the queries `o1` and `o2` respectively. The criteria (arguments) to the difference query can be referred to as the 'differor' and 'differand'. The representation of a general difference query is as shown in *Table 22* below:

Participant	Occurrence	Role
?o1	o4	differor
?o2	o4	differand

Table 22- Representation of a Difference Query

Universal Set A special query that returns all the concepts in the database. The universal set is represented simply by a specially assigned identifier.

Empty Set A special query that returns nothing. The empty set is represented simply by a specially assigned identifier.

Identification An identification query with the criterion `c1` returns the item identified as `c1` if that item exists in the database, and the empty set otherwise. Identification queries are used to allow identifiers to be wrapped inside queries so as to be nestable inside other queries. The representation of an identification query is as shown in *Table 23* below:

Participant	Occurrence	Role
c1	o4	identificand

Table 23- Representation of an Identification Query that returns `c1` if it exists in the database

Negation See page 5.

Cardinality A cardinality (counting) query is used to count the number of items fitting a description. A cardinality query takes a single criterion which is a set of items to be counted; this set is generally specified using a query which describes the criteria for items falling into the set. For example, a count of 'rich pensioners' would simply specify that what is counted are items fitting the description (i.e. the results of the query) 'rich pensioners'. This can be represented as shown in *Table 24* below:

Participant	Occurrence	Role
'rich pensioners'[22]	o4	counted

Table 24- Representation of the query to 'count rich pensioners'.

15.4 Occurrence-Related Queries

Occurrence query An *occurrence query* specifies the participant and role and can be represented as shown in *Table 25* below; all occurrence-identifiers with this participant in that role are returned.

Participant	Occurrence	Role
?o1	o4	participant-criterion
?o2	o4	role-criterion

Table 25- Representation of an Occurrence Query

Graphically, the 'search window' used to resolve an occurrence query looks like that shown in *Table 26* below:

Participant	Occurrence	Role
[match participant-criterion]		[match role-criterion]

Table 26- A graphical representation of the 'search window' used to resolve an Occurrence Query

[22] What is stored here is the identifier of the results of the query 'rich pensioners' (e.g. '`?o5`') rather than the text 'rich pensioners'. The text 'rich pensioners' which represents the resolution (`?o5`) of the underlying query (`o5`) 'concepts classified as rich, intersection concepts classified as pensioners' is used merely for readability.

Participant query

A *participant query* specifies the occurrence and role and can be represented as shown in *Table 27* below; all participant-identifiers in this occurrence with this role are returned.

Participant	Occurrence	Role
?o1	o5	occurrence-criterion
?o2	o5	role-criterion

Table 27- Representation of a Participant Query

Graphically, the 'search window' used to resolve a participant query looks like that shown in *Table 28* below:

Participant	Occurrence	Role
	[match occurrence-criterion]	[match role-criterion]

Table 28- A graphical representation of the 'search window' used to resolve a Participant Query

Role query

A *role query* specifies the occurrence and participant and can be represented as shown in *Table 29* below; all role-identifiers for the participant in the occurrence are returned.

Participant	Occurrence	Role
?o1	o6	participant-criterion
?o2	o6	occurrence-criterion

Table 29- Representation of a Role Query

Graphically, the 'search window' used to resolve a role query looks like that shown in *Table 30* below:

Participant	Occurrence	Role
[match participant-criterion]	[match occurrence-criterion]	

Table 30- A graphical representation of the 'search window' used to resolve a Role Query

15.5 Conditional Queries

A *conditional query* returns results only when the side-conditions evaluate to true. Conditional queries may use simple conditions, or complex conditions may be constructed using the operators `not`, `and`, `or`, `any-`*K* (i.e. cardinality / some specified number *K*), `all`, or `none`. These operators are not implemented in the usual truth-functional way of classical propositional logic since we are not dealing with propositions. Rather, they are implemented in a set-theoretic way since we are dealing with sets of occurrences, which are more fine-grained than propositions[23]. To evaluate conditions we select event occurrences and count them and then check if the count is correct (i.e. so-to-speak 'true' or false). This evaluation can be achieved using the various query types defined earlier; e.g. occurrence-related queries are used for selecting event occurrences and cardinality queries are used for counting. *Table 31* below demonstrates how a variety of complex conditions are evaluated in a set-theoretic, occurrence-centric manner.

Example Condition	**Set-Theoretic, Occurrence-Centric Evaluation** (Implemented using the Occurrence-Centric Queries and Cardinality Queries defined earlier)
"you pay late"	Resolve "select 'late payings by you' occurrences". If the count of its results is greater than or equal to one, the condition is true, otherwise it is false.
"you do `not`[24] pay the correct person"	If the count of "select 'payments to the correct person' occurrences" yields zero (i.e. such an occurrence does not exist), the condition is true, otherwise it is false.
"you pay late `and` you are a regular customer"	Resolve "select 'you pay late' occurrences" and "select 'you are a regular customer' occurrences". Count the number of results of each of the selects. If none of the selects yield zero (i.e. zero of the selects yield zero results), the condition is true, otherwise it is false. Notice here that we are counting the count events (i.e. counting the count events that yield zero).
"you pay late `or` you pay too little"	If counting the results of the union of "select 'you pay late' occurrences" and "select 'you pay too little' occurrences" yields one or more, the condition is true, otherwise it is false.
"you do `any two or more of`[25]: pay late, change your travel date, or miss your flight"	At first glance "counting the results of (select the union of (select 'you pay late' occurrences and select 'you change your travel date' occurrences and select 'you miss your flight' occurrences)) yields 2 or more results" would seem the obvious way to do evaluate this condition, but is not correct because it would not yield the correct result if you, for instance, changed your travel date twice but do neither of the other two things (i.e. did not pay late nor did you miss your flight). The correct implementation is: "counting the number of (selects which yield one or more results) yields 2 or more results", where the selects that are counted are "select 'you pay late' occurrences" and "select 'you change your travel date' occurrences" and "select 'you miss your flight' occurrences". Again, notice that you have to count your count events (i.e. count the counts that yield more than one) here; and then check that this count of counts yields two or more (as specified in the original condition "`any two or more of`..."). Notice that this condition would be difficult and very inefficient to implement in a traditional truth-theoretic manner, as propositional logic does not cater for counting and furthermore does not reify (individuate) occurrences so would not allow for the counting of occurrences; the condition would have to be implemented as a combination of "(you pay late and you change your travel date) or (you pay late and you miss your flight) or (you change your travel date and you miss your flight)" which is impractical for two reasons: - it would lead to a combinatorial explosion for large numbers of `k`, in '`any-`*k* `of`' expressions, and - it does not allow you to specify conditions like '`any` *X* `of`: …' where *X* is a variable computed at run-time.
you do `all`/`none` of: …	As for `any-k`, but replace *k* with the count of the number of different event-types listed (in the case of `all`) or with zero (in the case of `none`).

Table 31: Evaluating Truth Conditions in a Set-Theoretic, Occurrence-Centric Manner

[23] For example, instead of having merely the proposition 'P' as in classical propositional logic, in an occurrence-centric view we have an identified occurrence (e.g. `o1`) classified as being in the class 'P'.

[24] Words shown in `courier font` (e.g. `not`, `and`, `or`, `any-`*k*, etc.) are connectives for complex conditions

[25] Notice that, because an `any-`*k* can be combined with an algebraic greater-than or less-than query, we can easily implement '`any k or more`' and '`any k or less`'.