Computational Aspects of the FLBC Framework

Aspassia Daskalopulu

Department of Computer Science
King's College London
The Strand, London WC2R 2LS, U.K.
aspassia@dcs.kcl.ac.uk

Marek Sergot

Department of Computing
Imperial College London
180 Queen's Gate, London SW7 2BZ, U.K.
mjs@doc.ic.ac.uk

Abstract

Recent research has sought to develop formal languages for business communication as more expressive, flexible and powerful alternatives to current EDI standards, with potential benefits both for business-to-business exchanges in e-commerce and for general intra-organizational communication. A prominent approach in this area has become known as FLBC and is grounded on speech-act theory, event semantics and thematic rôles, and first-order logic (FOL). In this paper we discuss some of the specific technical choices for the representation of messages in the original FLBC framework and propose two modifications. The first eliminates a problematic modal logical component from the representations of messages; the second transforms the message representation into Skolemised clausal form. Focusing on two different computational tasks we illustrate how existing computational methods can be employed directly on the resulting representation for messages. We also propose an alternative formulation for messages using C-logic and discuss possible extensions to the resulting modified FLBC framework, for example in establishing whether an exchange is meaningful and in compliance with the setting in which the parties have pre-agreed to operate. Finally we consider some open problems and identify directions for future developments.

Keywords: formal languages for business communication, electronic data interchange, EDI, electronic commerce, speech acts, event semantics, thematic rôles, logic programming.

1 Introduction

Recent research has sought to develop formal languages for business communication as more expressive, flexible and powerful alternatives to current EDI standards, with potential benefits both for business-to-business exchanges in an e-commerce setting and for general intra-organizational communication in the context of office automation systems. A prominent approach in this area has become known as FLBC (Formal Language for Business Communication). First proposed by Kimbrough and his associates (see e.g. [12–17]), FLBC is grounded on speech-act theory, event semantics, and first-order logic (FOL). Work on its further development and deployment continues.

The main premise underlying FLBC is that messages exchanged between parties are made up of statements that can be analysed and represented in the F(P) framework of speech act theory. Speech act theory, whose origins are traced in the work of Austin [1] and Searle [22], is concerned with the analysis and representation of utterances. In Austin's terms, each utterance has a *locutionary* aspect (what is being said, the truth-functional content, the P-component), an *illocutionary* aspect (the force in saying something, the speaker's attitude towards the content, the F-component) and a *perlocutionary* aspect (what is achieved by saying something, what effect the speaker intends the utterance to have on the hearer). Speech act theory seeks to identify the main types of utterances in terms of their illocutionary aspect and to provide a systematic explication for each of these types in terms of the information conveyed between a speaker and a hearer and the conditions in which it is used successfully. Illocutionary forces are identified by verbs, which may be explicitly used by a speaker or alluded to in a given context. Hence, there are as many ways for a speaker to express the same attitude towards a given content as there are available appropriate verbs. For example, to issue a promise to meet his interlocutor a speaker might say "I promise to meet you", or "I will meet you", or "I will be there" (in the context of a dialogue) and so on. Speech act theory offers general categorization schemes for illocutionary forces (for example [1,2,23]). The relative merits and disadvantages of each scheme are beyond the scope of this paper. However, in all categorization schemes, clusters of specific verbs are grouped together under the illocutionary force that they are all meant to convey and they all share the perlocutionary aspect associated with that illocutionary force. For illustration, Table 1 summarizes Austin's (1962) classification of illocutionary forces [1] and shows some examples of specific verbs associated with each type.

Illocutionary Force	**Explication**	**Performative verbs**
Verdictive	Giving a verdict by a jury arbitrator or umpire, or giving an estimate, reckoning or appraisal. An exercise of speaker's judgement. Committing hearer/others to certain future conduct.	Acquit, convict, grade, assess, locate, measure, find (as a matter of fact), rule, diagnose etc.
Exercitives	Assertion of influence or exercising of powers or rights. Giving a decision that something is to be so. Creating obligations, permissions or prohibitions for hearer/others.	Appoint, vote, order, urge, advise, warn, dismiss, demote, name, bequeath, proclaim, resign, nominate, recommend etc.
Commissives	Committing speaker to a course of action. Assuming of an obligation or declaring of intention.	Promise, undertake, intend, plan, shall, adopt, oppose, guarantee, consent etc.
Behabitives	Describing speaker's reaction to other people's behaviour or states of affairs. Adopting an attitude.	Apologize, thank, commiserate, resent, welcome, protest, challenge etc.
Expositives	Describing views, clarifying reasons, arguments and communications.	Affirm, deny, state, identify, inform, postulate, interpret, agree etc.

Table 1 Austin's classification of illocutionary forces [1]

In contrast to current EDI representations, FLBC attempts to represent messages by making explicit their primary function (whether they are offers, acceptances to offers, promises, requests for services or goods, instructions for payment and so on—the F-component) and their locutionary content (the P-component). The F-component of a message is identified and represented using event semantics and thematic rôles [21] expressed in FOL, essentially in the same way that 'semantic cases' of case grammars [6,7] have been used in FOL representations in Artificial Intelligence (see e.g. [18, Ch.2], [3, Ch. 4]). As Kimbrough points out [13], in this way verbs can be represented as FOL predicates which take events as their arguments, and verb modifiers (thematic rôles) correspond to FOL predicates that associate individuals and events. Kimbrough demonstrates through an example how all predicates required for such representation fall into three categories, namely application-specific predicates, thematic rôles and kernel vocabulary. This, Kimbrough argues, makes the approach all the more attractive as it indicates that it is feasible to construct public, broad-utility lexicons and goes some way towards providing standards to replace current EDI schemes. In FLBC each message is also associated with various conditions. Such conditions explicate, for instance, what makes a promise 'kept', a request 'honored', an assertion 'veridical' and so on, depending on the type of the message.

In this paper we do not question the motivation and general spirit of the FLBC approach, which seem to us to be clear and well established, but rather discuss some of the specific technical choices in the original FLBC scheme for the representation of messages. Some of our comments are based on our previous experiences in modelling exchanges that establish contractual relations between parties [5]. Such exchanges typically include statements such as requests, promises, and assertions of a promissory nature as well as information-seeking questions. The aim of the paper is to suggest some simplifications of the FLBC representation scheme and identify prospects and directions for further development. In what follows we concentrate mainly on the representation of promissory statements, but the points we wish to make apply generally to the FLBC representation scheme. For illustration purposes, we use an example that is similar to those used by Kimbrough [13]. It should be noted that some of the issues raised in this paper are discussed also in recent work by Kimbrough. See e.g. [14] which may be seen as endorsing some of the points raised here.

2 Event descriptions and thematic rôles

Consider the following exchange between Peter and Susan:

> Peter: I would like to order a pizza from your menu please.
> Susan: Certainly. What kind of pizza would you like and what size?
> Peter: The "Good Earth Vegetarian"* please, but without onions. Large, please.
> Susan: Very well, that will be £13.95, cash please. What is the address?
> Peter: 12 Hunger Lane. How long will that be?

Susan: It is now 7 pm and we promise to deliver within half an hour. If our driver takes any longer than that, we deduct £1.00 from your bill.
Peter: Ok, thank you.

**The menu description of "Good Earth Vegetarian": mushrooms, onions, red and green peppers, all topped with mozzarella.*

The utterances that make up the exchange, such as Peter's request that initiates it, Susan's acknowledgement and promise for delivery, and Peter's promise for payment, among others, can be analysed in order to identify their primary function and then represented in FOL. The individual utterances of the exchange are represented in speech act theoretic terms using event semantics and thematic rôles. Let us first illustrate Kimbrough's idea by showing how one of the utterances (Peter's request for the delivery of a pizza) can be represented. The representation we show is different in several important respects from the one originally proposed by Kimbrough. We discuss these differences and possible variations in later sections.

Kimbrough's idea is essentially the following: Every utterance conforms to the F(P) framework of speech act theory, that is, every utterance U can be rewritten in F(P) form (U $\Rightarrow$ F(P)), where F is an illocutionary force (such as 'assert', 'promise', 'request', 'confirm' etc.) and P is the propositional content of the utterance. For example the F(P) structure of the utterance "Peter requested that Susan deliver a pizza" is

```
request("Susan delivers a pizza to Peter")
```

For the utterances of interest, both the F-component and the P-component can be represented as events whose attributes are described in terms of thematic rôles [21]. Thus, the statement "Peter requested that Susan delivered a pizza" can be modelled as a requesting event with Peter as its agent, Susan as its recipient, and whose 'theme' or 'content' is another event, the delivering of a pizza. Pictorially, Peter's request can be viewed as shown in Figure 1.

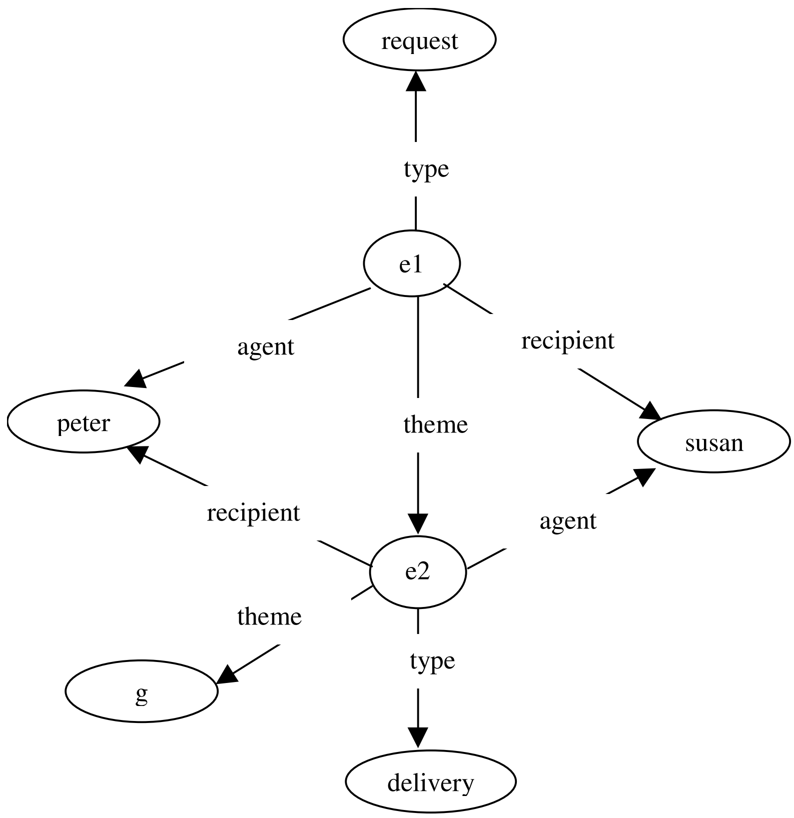

Figure 1 A request for delivery

In Figure 1, the propositional content of the example utterance ("Peter requested that Susan deliver a pizza") is modelled as a delivery event with Susan as its agent, Peter as its beneficiary, and the pizza (which can be described, if desirable, in more detail in terms of its size and toppings) as its 'theme'. A rendition in first-order logic is as follows, where the pizza is left for the moment un-analyzed; we assume that g is a name constant for it:

```
∃e ∃e'[request(e) ∧ agent(e, peter) ∧ recipient(e, susan) ∧
            theme(e, e') ∧ delivery(e') ∧ agent(e', susan) ∧
            recipient(e', peter) ∧ theme(e', g) ∧ pizza(g)]
```

After Skolemization and conversion to normal form this FOL representation gives rise to the following collection of assertions, where e1 and e2 are now Skolem constants naming the request and delivery events:

```
request(e1).
agent(e1, peter).
recipient(e1, susan).
theme(e1, e2).

delivery(e2).
agent(e2, susan).
recipient(e2, peter).
theme(e2, g).
pizza(g).
                                                        (A1)
```

Susan's promise to deliver a pizza to Peter can be represented in similar fashion as an event of type 'promise' whose 'theme' is another event of type 'delivery':

```
promise(e3).
agent(e3, susan).
recipient(e3, peter).
theme(e3, e4).

delivery(e4).
agent(e4, susan).
recipient(e4, peter).
theme(e4, f).
pizza(f).
                                                        (A2)
```

Nested utterances can be represented straightforwardly in the same manner. For example, the utterance "Peter requests that Susan promises to deliver a pizza" can be expressed as a requesting event whose 'theme' is a promising event whose 'theme' is a delivery event (and similarly for longer chains of nested events).

The collection of thematic rôles employed in the original FLBC framework by Kimbrough is drawn from a scheme of Parsons [21]; Table 2 shows some examples of such rôles (or 'semantic cases' in case grammar terms). In the representation fragments (A1) and (A2) above we restricted ourselves to 'agent', 'theme' and 'recipient' for the sake of simplicity. Any collection of thematic rôles can be used as predicates associating events with their individual aspects, as appropriate, and the representation can be easily adjusted to accommodate them. As Kimbrough observes, some standard lexicon of rôles will have to be agreed by the various parties in any given application; there seems to be nothing problematic about reaching such an agreement.

Rôle	**Description**
Agent	Volitional initiator of action
Patient	Object or individual undergoing action
Theme	Object or individual moved by action
Goal	Individual toward which action is directed
Source	Object or individual from which something is moved by the event, of from which the event originates
Experiencer	Individual experiencing some event
Beneficiary	Object that benefits from the event
Location	Place at which the event is situated
Instrument	Secondary cause of event; the object or individual that causes some event that in turn causes the event to take place

Table 2 Examples of thematic rôles from Kimbrough [13]

The representation constructed so far, for two of the utterances of the example, has two features that we wish to comment on at this point. First, we have represented messages (Peter's request for delivery and Susan's promise for delivery) *prima facie*. That is to say, we have constructed logical formulations of the messages as they appear, without reference to the context in which they are exchanged. Such context might include, for example: agreed conversation policies between the parties to an exchange; agreed procedures for interpreting messages and for deriving inferences from them to establish what a party's appropriate subsequent response to a received message might be or to determine what action a party might be expected to take as a result of issuing a message; an agreed lexicon of predicates and terms to be used in messages, and so on. The assumption is that standard routine messages of the type intended for representation in FLBC, exchanged in the normal course of business, will not have to contain explicit representations of all the procedures and conventions agreed by the parties for the conduct of their business. This assumption is a feature of the original FLBC framework and we see no reason to question it.

Second, as demonstrated with the example representation so far, we allow events themselves to be the 'themes' of other events in exactly the same way that (physical) objects can be the themes of events. Thus, the delivery event stipulated by Peter is the theme (or 'direct object') of the requesting event that he initiates, and the delivery event stipulated by Susan is the theme of the promising event that she puts forward, just as a pizza $\mathtt{g}$ is the theme of a delivery event. In this way, the content of an F-event (such as a request or a promise event) is directly accessible in the representation in the same way that the content of a P-event (such as a delivery event) is accessible. Queries of the form "what does Peter request?" can be answered as easily as queries of the form "what does Susan deliver?". This is one important respect in which our representation differs from Kimbrough's original proposal. We will return to this point in section 4 below.

We are aware that in adopting this use of the term 'theme' as a synonym for 'patient' or 'direct object' we are departing from the set of thematic rôles suggested by Parsons [21] and followed by Kimbrough in FLBC. This is just for simplicity: we do not need, for the purposes of this paper, to distinguish between the 'theme', 'patient' and 'direct object' of an event. We choose 'theme' as a suitably neutral term for all three.

In addition to providing means of querying message representations in order to extract the features of interest, it is also straightforward to devise tools to help with the construction of FLBC representations. In particular, it is possible to formulate various constraints that a message must satisfy and arrange for these constraints to be checked as the representation is constructed. For example, one cannot meaningfully deliver something to oneself, one cannot meaningfully promise to deliver something to oneself, one cannot meaningfully promise to perform something in the past, and so on. Such constraints are easily expressed and checked using standard computational methods. These constraints can express common sense features of events, legal requirements, or other requirements that the parties might have agreed to abide by between themselves. The ability to express such constraints is an important advantage of the FLBC representation as we see it, and of great practical significance.

3 Structured descriptions of objects

It might be felt that there is something suspicious about naming events in this manner and then making one event the 'theme' (or 'patient' or 'direct object') of another. Indeed, even the idea of naming events—some of which may never actually happen—may already seem to be philosophically suspect.

We now wish to argue that there is nothing problematic about the use of these devices. The argument hinges on the difference between constants that name specific individuals and constants that are used essentially as internal system identifiers on which to build representations of complex structured objects.

Let us turn first to the content ('theme') of the delivery event requested by Peter, the pizza. This has been represented by a name constant $\mathtt{g}$, that is, by a unique identifier for a *specific* pizza. Now this may seem strange: in the context of ordering a pizza, is it much more likely that Peter requests an instance of a particular *type* of pizza rather than a specific pizza by name ('that one')—but the latter is not impossible. Although it is rather fanciful to imagine customers requesting delivery of a specific pizza by name, it is very easy to think of other similar examples where ordering by name is normal. When John sends to Mary an offer to buy her car he is not offering to purchase an instance of a *type* of car but a specific car that he will identify by name. When Jim, a car dealer, orders a Diablo Avenger from his supplier, on the other hand, he does not order a specific Diablo Avenger by name but an instance of a car of that type. In general we shall need to distinguish in the representation between constants that are names of specific objects and constants that are identifiers for unnamed instances of a given type of (structured) object.

In the pizza example, it is likely that the content of the requested delivery event is not a specific named pizza but an instance of a particular *type* of pizza. That being so, the theme of the delivery event is naturally represented in FOL by the existentially quantified statement:

```
∃p [theme(e2,p) ∧ pizza(p) ∧ topping(p, mushroom) ∧ topping(p, peppers) ∧…]
```

We earlier suggested that the constant `g` in the representation could be seen as a name for a pizza. But it can also be seen as the Skolem constant introduced by Skolemisation of the existentially quantified statement above. We assume that there will be some convention for distinguishing name constants from Skolem constants: henceforth in this paper we shall point out Skolem constants where they are not obvious from context. The constant `g` in the representation earlier was intended to be a Skolem constant.

Exactly similar considerations arise in relation to the representation of events. Constants `e1`, `e2`, … are intended to be seen as internal identifiers used for building a structured representation of an event. They can also be seen as Skolem constants resulting from Skolemisation of existentially quantified expressions listing the attributes (thematic rôles) of events.

The following examples illustrate how such Skolemisation can express a variety of different forms:

(a) Susan promises that a named pizza, `f`, will be delivered (without specifying by whom):

```
∃e [promise(e) ∧ agent(e, susan) ∧ recipient(e, peter) ∧
      ∃e'[theme(e, e') ∧ delivery(e') ∧
             recipient(e', peter) ∧ theme(e', f) ∧ pizza(f)]]
                                                                    (A2.a)
```

The corresponding collection of assertions is:

```
promise(e2).                           delivery(e3).
agent(e2, susan).                      recipient(e3, peter).
recipient(e2, peter).                  theme(e3, f).
theme(e2, e3).                         pizza(f).
```

(b) Susan promises that a pizza (of a type) will be delivered (by her):

Replace the named pizza `f` by a Skolem constant (`p1`, say); add `agent(e3, susan)`

```
promise(e2).                           delivery(e3).
agent(e2, susan).                      agent(e3, susan).
recipient(e2, peter).                  recipient(e3, peter).
theme(e2, e3).                         theme(e3, p1).
                                       pizza(p1).
```

In FOL, with existential quantifiers:

```
∃e[promise(e) ∧ agent(e, susan) ∧ recipient(e, peter) ∧
      ∃e'[theme(e, e') ∧ delivery(e') ∧ agent(e', susan) ∧
             recipient(e', peter) ∧
                    ∃p[theme(e', p) ∧ pizza(p)]]]
                                                                    (A2.b)
```

(c) Susan promises that a pizza (of a type) will be delivered by an employee of hers:

Replace `agent(e3, susan)` by an assertion about Skolem constant `a1` (say) instead:

```
promise(e2).                           delivery(e3).
agent(e2, susan).                      agent(e3, a1).
recipient(e2, peter).                  employee_of(a1, susan).
theme(e2, e3).                         recipient(e3, peter).
                                       theme(e3, p1).
                                       pizza(p1).
```

In FOL, with existential quantifiers:

```
∃e[promise(e) ∧ agent(e, susan) ∧ recipient(e, peter) ∧
      ∃e'[theme(e, e') ∧ delivery(e') ∧
            ∃a[agent(e', a) ∧ employee_of(a, susan)] ∧ recipient(e',peter) ∧
∃p[theme(e', p) ∧ pizza(p)]]]
                                                                    (A2.c)
```

(d) Susan promises that a pizza (of a type) will be delivered by John at 7:00 pm:

Replace the Skolem constant `a1` by the name constant `john`; assert the time of the promised event `e3`.

```
promise(e2).                    delivery(e3).
agent(e2, susan).               agent(e3, john).
recipient(e2, peter).           recipient(e3, peter).
theme(e2, e3).                  theme(e3, p1).
                                pizza(p1).
                                time(e3, 7).
```

In FOL, with existential quantifiers:

```
∃e[promise(e) ∧ agent(e, susan) ∧ recipient(e, peter) ∧
      ∃e'[theme(e, e') ∧ delivery(e') ∧ agent(e', john) ∧ time(e', 7) ∧
             recipient(e', peter) ∧
                    ∃p[theme(e', p) ∧ pizza(p)]]]
                                                                    (A2.d)
```

Event names are naturally seen as Skolem constants, that is, as placeholders on which the structured representation of an event can be built, rather than as name constants.

3.1 Logics of complex objects

The distinction between named objects and (named) instances of types of objects is a central feature of object-oriented representations, where instances may be identical in every attribute yet still represent distinct individuals. 'Object logics', developed in the late 80s and early 90s, draw upon the areas of object-orientation and logic programming to provide a framework for representing and reasoning with structured information. From object-orientation they inherit the concepts of object identity, the notion of a complex object and mechanisms for object classification and property inheritance. From logic programming they inherit the concepts of unification and answer substitution and a strategy for deductive query processing. See e.g. [9] for a survey of the main approaches, beginning with O-logic [20], and following with further developments such as extended O-logic [11], F-logic [10], and C-logic [4]. The field seems to have fallen dormant recently, subsumed to a large extent under the development of 'description logics' (though the emphasis there is slightly different). For present purposes, C-logic is particularly convenient. It provides a shorthand for FLBC representations which is both practical and natural, and—perhaps more importantly—allows us to expand on our remarks about the use of Skolem constants as identifiers for structured representations of complex objects.

In object logics, a complex object is represented as a term of the language. In C-logic, object descriptions take the form:

$$\texttt{ClassName : ObjectIdentifier[Attribute}_1 \Rightarrow \texttt{Value}_1\texttt{, ..., Attribute}_k \Rightarrow \texttt{Value}_k\texttt{]}$$

The value of an attribute may be simple, an enumerated type, or another object description. For example, the following C-logic term represents an instance of type pizza:

```
pizza: p[
      size⇒large,
      base ⇒ thin,
      toppings ⇒ {mushroom, peppers, cheese}
      ]
```

Informally, a C-logic term can be read as asserting the existence of an object of the specified type and structure; in the example above, a pizza object with the stated structure and attributes. Formally, the semantics in C-logic is defined directly in terms of object, attribute and value structures. There is also a translation of each C-logic term into an equivalent set of FOL conjunctions in which object identifiers become constant symbols, class names are represented by unary predicates, and attributes are represented by binary predicates that take object identifiers (or rather the constants introduced in the translation) and values as their arguments. The general form of a C-logic object term shown above is thus translated into the following set of FOL sentences:

```
ClassName(ObjectIdentifier).
Attribute₁(ObjectIdentifier, Value₁).
:
:
Attributeₖ(ObjectIdentifier, Valueₖ).
```

Peter's request for delivery of a pizza of a specific type can be represented in C-logic as follows:

```
request:e1[
      agent ⇒ peter, recipient ⇒ susan,
      theme ⇒ delivery:e2[
                     agent ⇒ susan, recipient ⇒ peter,
                     theme ⇒ pizza:p1[
                                    size ⇒ large, base ⇒ thin,
                                    toppings ⇒ {mushroom, peppers, cheese}
          ]                ]         ]
                                                                  (C1.1)
```

His request for delivery of a specific *named* pizza g, on the other hand, can be represented as follows:

```
request:e1[
      agent ⇒ peter, recipient ⇒ susan,
      theme ⇒ delivery:e2[
                     agent ⇒ susan, recipient ⇒ peter,
                     theme ⇒ pizza:p2[
                                    name ⇒ g,
                                    size ⇒ large, base ⇒ thin,
                                    toppings ⇒ {mushroom, peppers, cheese}
         ]                 ]         ]
                                                                  (C1.2)
```

Similarly, Susan's promise for delivery of a pizza of a specific type can be represented as:

```
promise:e3[
      agent ⇒ susan, recipient ⇒ peter
      theme ⇒ delivery:e4[
                     agent ⇒ susan, recipient ⇒ peter,
                     theme ⇒ pizza:p3[
                                    size ⇒ large, base ⇒ thin,
                                    toppings ⇒ {mushroom, peppers, cheese}
          ]                ]         ]
                                                                  (C2.1)
```

It should be easy to see how C-logic constructs can be written for other variations of Susan's promise that were discussed earlier. (In all of these examples, persons Susan and Peter are represented using simple constants. Naturally they could also be represented as structured objects of type person whose name attributes have the values 'Susan' and 'Peter' respectively.)

As examples (C1.1) and (C1.2) illustrate, such a representation affords an explicit distinction between two kinds of constants, namely those that name object/event instances and those that name object/event types. There are further advantages to using C-logic terms for the representation of messages. The formulation in C-logic is more concise than the formulation via collections of assertions and the relation between F-events and the P-events they bear as their themes is easier to note. The representation allows deductive retrieval of information to the level of detail that is of interest. We can, for example, pose queries to retrieve the agent, or recipient or theme, or all of these features of a promising or requesting event. As regards queries about the theme of such promising or requesting events, we can in turn retrieve the whole structured term or any particular attributes of interest. For example evaluation of the query "What did Susan promise?" against (C2.1) could yield an answer at the required level of granularity, ranging from "a delivery", to "a delivery of a pizza", to the full description of the stipulated delivery e4 and its pizza p content.

The fact that C-logic expressions have both direct semantics and a translation into FOL makes it possible to mix C-logic expressions (used for syntactic convenience) with FOL expressions that represent additional information about the messages and the context in which parties exchange them or constraints that can be used to establish whether the exchange is meaningful. In this way, *prima facie* representations of messages can be syntactically distinguishable from context and constraint representations while at the same time directly usable by the same computational methods.

4 The α-β-γ-δ representation

We turn now to the original FLBC framework [13] and in particular to its method for associating the representation of events such as requests, promises, and other speech acts, with representations of the propositional content of those acts. Kimbrough [13] observes that statements about promissory utterances such as “Susan promised Peter to deliver a pizza” can be paraphrased as “there is a promising event in which the speaker (agent) is Susan and the recipient is Peter, and in which Susan keeps this promise if and only if she does something that causes there to be a delivery of pizza to Peter”. The representation of Susan’s promise in the original FLBC framework takes the following form. For simplicity we assume that Susan promises to deliver a specific named pizza `f`—nothing turns on this.

```
∃e[promise(e) ∧ agent(e, susan) ∧ recipient(e, peter) ∧
      □(kept(e) ↔ ∃e′[delivery (e′) ∧ agent(e′, susan) ∧
                          beneficiary(e′, peter) ∧ theme(e′, f) ∧
                          sake(e′, e)])]
                                                                  (K1)
```

Notice that there is no *explicit* association of the promise event `e` with the delivery event that is promised; that relationship is captured implicitly through the set of conditions specifying what renders the promise ‘kept’. The `sake(e′,e)` condition is required to ensure that several different promises to deliver (the same kind of) pizza are not all automatically kept by one single delivery. This condition is sometimes omitted when similar examples are discussed in Kimbrough’s papers; we understand (personal communication) that such a condition is intended to be included. We conjecture later, in section 5.1, why this condition is sometimes included, sometimes not—there is a certain ambiguity in the term ‘kept’.

The □ in the representation (K1) is a modal necessity operator, to be read as ‘it is necessarily the case that’ or ‘in all possible worlds it is the case that’. The use of the modal operator □ raises a number of very difficult problems. We discuss this aspect of the representation separately in the following section.

The general form of representations such as (K1) is the following:

$$\alpha \wedge \Box(\gamma \leftrightarrow \delta)$$

Here, α corresponds to the conjunction specifying the type of message (promise, request etc.) and attributes such as its agent and its recipient (but not its ‘theme’). γ is the device by which the α-event is associated to its content: promises can be ‘kept’, requests can be ‘honored’, assertions can be ‘veridical’, and so on. δ stands for the conditions that need to be satisfied in order for the given message to be accorded status γ; in the representation (K1), δ corresponds to the set of conditions describing the promised delivery event. More generally still, it is sometimes convenient to attach further conditions to the content of the α-event. The general form of an FLBC message is therefore (‘the α-β-γ-δ form’) as follows:

$$\alpha \wedge \Box(\beta \rightarrow (\gamma \leftrightarrow \delta)) \tag{S}$$

The β component stands for the extra conditions, empty in all the examples considered so far. We will discuss this component later in section 5.

4.1 Elimination of the modal logical component

The use of the modal operator □ in the α-β-γ-δ representation raises a number of what seem to us to be very severe problems. First, there is the question of which specific modal operator to employ. What kind of logical properties should it exhibit? It has been suggested that nothing much may turn on the choice: it can be left to personal preference. Yet this seems very unsatisfactory. Surely it cannot be completely arbitrary. There must be some principles on which to choose between candidates. This is a far from trivial matter, however, especially when, as here, details of interactions between the modal operator and quantifiers have to be considered as well. Second, besides the choice of operator itself, the representation is no longer in FOL. Standard FOL methods—for reasoning with the representation, answering queries, checking constraints, and so on—are not immediately applicable.

Kimbrough suggests that the modal operator □ can be eliminated by translating expressions such as (K1) into FOL by encoding the accessibility relation in the semantics of □ in the representation. Thus the statement "Susan promised Peter to deliver a pizza" can be paraphrased as "there is a promising event in the actual world, `a*`, of which the agent is Susan and the recipient is Peter, and for any possible world `w` `accessible` from the actual world `a*`, the promise is `kept` in that world, if and only if there is a delivery event `e′` in that world, such that Susan is the agent of the event, the beneficiary of the event is Peter, and the theme of the event is the pizza". This leads to the following representation in FOL:

```
∃e[promise(e, a*) ∧ agent(e, susan, a*) ∧ recipient(e, peter, a*) ∧
      ∀w [accessible(w, a*) → (kept(e,w) ↔ ∃e′ [delivery(e′, w) ∧
                                   agent(e′, susan, w) ∧
                                   beneficiary(e′, peter, w) ∧
                                   theme(e′, f, w)
                                   sake(e′, e, w)])]]
                                                                  (K1.1)
```

But this raises two further issues, namely: (i) what is an appropriate definition for the accessibility relation ? (which is another way of asking what properties the modal operator □ should exhibit); and (ii) how can this representation be used (queried, checked)?

Here is an alternative suggestion. Instead of quantifying over possible worlds, let us quantify over possible events. This requires no additional machinery, simplifies the representation, and brings it back within the scope of standard computational methods for FOL. Quantifying over possible events suggests the following representation in place of the original (K1):

```
∃e[promise(e) ∧ agent(e, susan) ∧ recipient(e, peter) ∧
      ∀e′(kept(e, e′) ↔ [delivery (e′) ∧ agent(e′, susan) ∧
                         beneficiary(e′, peter) ∧ theme(e′, f) ∧
                         sake(e′, e)])]
                                                                  (qK1)
```

The binary predicate `kept(e,e′)` can be read as expressing that promise `e` is 'kept' (perhaps 'fulfilled' might be better) by the occurrence of the event `e′`. Clearly the method can be generalised to any similar representation in the general α-β-γ-δ form. In place of

$$\exists e[\alpha(e) \wedge \Box(\beta(e) \rightarrow [\gamma(e) \leftrightarrow \exists e'\delta(e,e')])]$$

we are suggesting the FOL formulation:

$$\exists e[\alpha(e) \wedge \forall e'(\beta(e) \rightarrow [\gamma(e,e') \leftrightarrow \delta(e,e')])]$$

We are not claiming of course that the two representations (K1) and (qK1) are formally equivalent; that would depend on detailed properties of the operator □ amongst other things. We are suggesting that (qK1) preserves the spirit of the original FLBC whilst offering considerable advantages in terms of simplicity and ease of use.

Besides providing a means of circumventing the limitations of material implication (→), the main function of the modal operator □ in the α-β-γ-δ representation is to provide a device for creating intensional contexts. It is generally accepted, for example, that promising, in common with most other types of speech acts, creates intensional contexts in which substitution of equivalents cannot be done confidently: "…[T]o promise that you will come to the party is not the same as to promise that you will skip Esmeralda's wedding, even if you skip Esmeralda's wedding if and only if you come to the party" [13]. If *x* promises *P* and *P* is logically equivalent to *Q*, it does not necessarily follow that *x* promises *Q*. If *Q* turns out to be false, however, then it does follow (according to Kimbrough, and we agree) that the promise of *P* is broken (not kept). The modal □ operator is Kimbrough's device for creating a suitable intensional context. Our suggestion is that quantifying over possible events also provides a means for creating intensional contexts.

In Skolemised form (qK1) can be written as follows.

```
promise(e3)
agent(e3, susan)
recipient(e3, peter)
∀e′(kept(e3, e′) ↔ [delivery (e′) ∧ agent(e′, susan) ∧
                          beneficiary(e′, peter) ∧ theme(e′, f) ∧
                          sake(e′, e3)])
                                                                    (qK1′)
```

Because of the translation between C-logic and FOL, the representation (qK1′) can also be rewritten using the structured C-logic syntax to specify the attributes of the event e3 The details are simple and so we do not show them here.

The biconditional in the last sentence can be re-written as two separate implications:

```
∀e′(kept(e3, e′) ← [delivery (e′) ∧ agent(e′, susan) ∧
                          beneficiary(e′, peter) ∧ theme(e′, f) ∧
                          sake(e′, e3)])
                                                                    (qK2)
```

```
∀e′(kept(e3, e′) → [delivery (e′) ∧ agent(e′, susan) ∧
                          beneficiary(e′, peter) ∧ theme(e′, f) ∧
                          sake(e′, e3)])
                                                                    (qK3)
```

Both these can be converted to clausal form and the resulting sets of clauses can be used for computational purposes in different ways, as we now discuss.

4.2 Retrieval method

We take it that the purpose of FLBC is to make perspicuous what is being requested, promised, delivered and so on in a message, within the context of agreed predicates for the types of speech acts employed, the name constants used and so on. The formulation of messages in the α-β-γ-δ scheme makes the handling of retrieval queries awkward. Suppose that one wants to determine *what* Susan promises. One way is by inspecting the representation, that is, by scanning or parsing the representation to extract the component δ following the $\leftrightarrow$ symbol. If that is to be the retrieval method, however, then there is no particular value in having adoped a logical formulation in the first place: any convenient syntactic device would do just as well.

It may seem that the conditions δ representing the content of the promise cannot be retrieved from the α-β-γ-δ representation using standard (deductive) query evaluation methods. It is true that a query on the γ-component, in this example (where e3 is the constant serving as the object identifier for the promising event brought about by Susan) a query of the form:

```
?-kept(e3, X).
```

does not work. Depending on the query processing mechanism employed, possible answers generated by this query are either yes/no, or a conjunction of conditions δ' such that

```
∀e(kept(e3, e) ↔ δ′(e3, e))
```

is a logical consequence of the representation. Even in the latter case, there is no guarantee that the retrieved conditions δ' will be identical to the δ-conditions as written in the representation: δ' will be logically equivalent to δ but not necessarily identical to δ. This is clearly undesirable. As already discussed, promising, and many other kinds of speech acts, create intensional contexts in which substitution of equivalents cannot be performed confidently: to know what was promised we need to be able to retrieve exactly the conditions δ and not merely something logically equivalent to δ.

However, there is a way of using standard computational methods (such as Prolog or C-logic) to retrieve elements of the message from its representation. As already observed, the biconditional defining ‘kept’ in (qK1′) can be rewritten as two separate implications, both of which can be re-written in clausal form. For retrieval purposes, the useful half is the following

```
∀e'(kept(e3, e') → [delivery (e') ∧ agent(e', susan) ∧
                        beneficiary(e', peter) ∧ theme(e', f) ∧
                        sake(e', e3)])
                                                                (qK3)
```

which is re-written in clausal form as follows:

```
delivery(E) ← kept(e3, E).
agent(E, susan) ← kept(e3, E).
beneficiary(E, peter) ← kept(e3, E).
theme(E, f) ← kept(e3, E).
sake(E, e3) ← kept(e3, E).
                                                                (qK3')
```

Here we have employed the standard notation for clauses as used in logic programming, and the Prolog convention that strings beginning with upper case letters are variables. As usual, all variables in clauses are implicitly universally quantified.

The clauses (qK3′) give us a simple means of retrieving any element of the description of the content of `e3`. In order to retrieve it, we add to the representation the (temporary) additional assertion:

```
kept(e3, e999).
```

where `e999` (say) is any new constant not appearing in the representation. This temporary assertion is necessary to provide a value for the E variable and to satisfy the body of the clauses that essentially specify the content of the promise `e3`. The value of any attribute of interest of the theme of `e3` can be retrieved by formulating queries at the required level of detail. For instance to determine the agent, beneficiary and theme of the promised event (the delivery), the query is:

```
?- agent(e999, X), beneficiary(e999, Y), theme(e999, Z)
```

This works very simply in Prolog and can be easily combined with C-logic notation if desired. Nested events ("Peter promises Jim that he will request Susan to …") can also be treated in this way though some care needs to be taken to keep track of the temporary assertions. The procedure of making the temporary 'kept', 'honored', (more generally γ) assertions, evaluating the query, and then removing the temporary assertions, can be packaged up for the user's convenience. In similar fashion, it is also possible, though a little more awkward, to use the method for checking that a message under construction satisfies any constraints.

So this then is our first suggested modification: abandon the modal operator and quantify over possible events instead; rewrite the $\gamma \leftrightarrow \delta$ component in clausal form. In order to retrieve elements of a message from its representation: make a temporary assertion γ, and then query the δ attributes of interest, either in FOL or using C-logic notation (or some other variant thereof).

4.3 Discussion

How does this modified α-β-γ-δ representation using quantifiers compare to the representation scheme introduced in the opening sections, where the content of a promise (or request, or other message type) was represented by making an explicit assertion that one event is the 'theme' of another? In fact the two representation schemes are much more closely related than may at first appear, as we now show.

Suppose that for the purposes of retrieval, instead of packaging up temporary 'kept' assertions, we add once and for all the assertion:

```
kept(e3, th(e3)).
```

or more generally the clause

```
kept(E, th(E)) ← promise(E)
                                                                (θ)
```

So now when we wish to extract, say the beneficiary of the event promised by Susan in e3, it is enough to evaluate the following query:

```
?- beneficiary(th(e3), X)
```

There is no need to make any temporary assertions of any kind.

Notice that clause (θ) is the Skolemised form of the following general statement:

```
∀e [promise(e) → ∃e′ kept(e, e′)]
```

The intended reading is that for any promising event `e` there is an event `e′` which, *if it occurred*, would render the promise in `e` kept. It is *not* to be read as stating that every promise is *actually* kept. Again, it may be felt that there is something philosophically suspect about the naming of hypothetical events in this manner. What if event `e′` never occurs, or never could occur? What happens (Steve Kimbrough, private communication) when the impossible is promised (as it often is)? As discussed earlier in section 3, we have no difficulty in naming hypothetical events. There is nothing any more problematic in our view about referring to an event which does not, could not, exist than there is in referring to a pizza object which does not, or could not, exist. The names are simply place-holders used for building structured representations.

The representation (qK3′) can be further simplified. Since `e3` is a promising event, the `kept` condition in the clauses of (qK3′) can be resolved away, yielding the following set of simpler clauses:

```
delivery(th(e3)).
agent(th(e3), susan).
beneficiary(th(e3), peter).
theme(th(e3), f).
sake(th(e3), e3).
                                                  (qK3″)
```

Now one can see that this is essentially identical to the representation we employed in the earlier sections of the paper, except that there we used a constant `e4` to identify the theme of `e3` instead of the functional term `th(e3)`, and we associated `e4` to `e3` by means of the predicate `theme`. We could have written `th(e3)=e4`; that is, in place of the version above of (qK3″) we could have:

```
th(e3)= e4.
delivery(e4).
agent(e4, susan).
beneficiary(e4, peter).
theme(e4, f).
sake(e4, e3).
```

We find it more convenient (and flexible) to write `theme(e3, e4)` instead. We also avoid the problems of reasoning explicitly with equality in the representation.

Clearly the method can be generalised to other kinds of speech acts: for every requesting event `e` there is an event `th(e)` which, if it occurred, would make the request 'honored'; for every asserting event `e` (that asserts the occurrence of an event), there is an event `th(e)` which, if it occurred, would make the assertion 'veridical', and so on. In clausal form:

```
honored(E, th(E)) ← request(E).
veridical(E, th(E)) ← assertion(E).
```

This then is our second suggested modification: for practical purposes, instead of defining 'kept' (or 'honored' or 'veridical', or more generally γ) for an event `e` in full, simply express the corresponding δ conditions as a set of assertions specifying the required properties of event `th(e)`. The term `th(e)` stands for the (unnamed) event that is the 'theme' of `e`. As a further modification, replace `th(e)` by a new constant `e′`, and add `theme(e, e′)` to the representation. This last step is optional (though we prefer it since it is much more flexible and easier to deal with).

5 Monitoring the performance of business exchanges

It is natural to explore whether we could also make use of the other part of the 'kept' definition, i.e.

```
∀e'(kept(e3, e') ← [delivery (e') ∧ agent(e', susan) ∧
                       beneficiary(e', peter) ∧ theme(e', f) ∧
                       sake(e', e3)])
                                                              (qK2)
```

in order to provide an extended representation system in which one could check whether a promise had in fact been kept, or more generally, in which the performance of promises, requests, replies, the evolution of the message and business exchanges generally, could be monitored. This seems to be outside the original motivation for FLBC but let us consider it.

Whether or not we employ the original α-β-γ-δ representation or the suggested alternatives discussed in previous sections, we have to be able to distinguish, in the representation, between descriptions of events that *could* happen and descriptions of events that have *actually* happened.

So let us add another possible attribute to the representation of events: `mode(e, actual)` will represent that event `e` has actually happened. (We could have used a unary predicate 'actual' just as well. The binary predicate `mode` fits better with C-logic syntax.)

Now we have to be careful. If as suggested in the previous section we read `kept(e,e')` as saying that (promise) `e` would be kept by `e'` if `e'` occurred, we need another predicate, say `actually_kept`, defined as follows:

```
actually_kept(e, e') ↔ kept(e,e') ∧ mode(e', actual)
```

We shall have reason to adjust this definition presently. (Naturally we could also reserve the use of the predicate 'kept' for this sense of actually kept, and use a different predicate for encoding the δ conditions defining the content of the promise. We leave it like this.)

For use in combination with the clausal versions of the α-β-γ-δ representation in the previous section, the definition above produces the following clause:

```
actually_kept(E, th(E)) ← promise(E), mode(th(E), actual).
```

if we choose to use the `th` device, or

```
actually_kept(E, E') ← promise(E), theme(E, E'), mode(E', actual).
```

if (as we do) we prefer the use of a `theme` assertion to the use of the function symbol `th`.

We have omitted the clause corresponding to the 'only if' half of the definition of `actually_kept` on the assumption that the resulting representation is to be executed as a logic program, in Prolog or C-logic or some other variant. The 'only if' part is then provided implicitly by the usual semantics of logic programs.

Exactly similar considerations apply to formulating conditions for determining when requests are (actually) honored, when directives are (actually) obeyed, and correspondingly for other speech act types of interest.

We noted earlier in section 4 that in the original α-β-γ-δ representation the `sake` condition is sometimes included and sometimes not when Kimbrough discusses similar examples. We conjecture that perhaps one reason for this is that there is a certain ambiguity in the intended use of 'kept' in the α-β-γ-δ representation of promises in the original FLBC framework. The `sake` condition is only really necessary when determining whether a promise has actually been made; as already observed, this is outside the original motivation of FLBC though sometimes implicit in some of the discussions.

The `actually_kept` definition above, and its corresponding clausal representation, however, embody an important over-simplification, which we turn to now.

5.1 When is a promise kept?

In the example message exchange of section 2, Peter requests (`e1`) a particular kind of delivery event (labelled `e2` in the representation shown). Susan responds (`e3`) by promising a delivery event (labelled `e4` in the representation). Suppose that a delivery event `e16` (say) is added to the representation, recorded as being in response to Susan's promise.

In C-logic syntax, the representation of Susan's promise is:

```
promise:e3[agent ⇒ susan, recipient ⇒ peter, mode ⇒ actual,
            theme ⇒ delivery:e4[
                   agent ⇒ susan, recipient ⇒ peter,
                   theme ⇒ pizza:p4[
                                 size ⇒ large, base ⇒ thin,
                                 toppings ⇒ {mushroom, peppers, cheese}
            ]                    ]      ]
```

The representation of the delivery event `e16` is as follows (C-logic syntax):

```
delivery: e16[agent ⇒ susan, recipient ⇒ peter, mode⇒ actual,
                   theme ⇒ pizza:p16[
                                 size ⇒ large, base ⇒ thin,
                                 toppings ⇒ {mushroom, peppers, cheese}
                                        ]
                   sake ⇒ e3
            ]
```

Notice that the identifiers for the two pizzas are different; we are assuming here that the pizza promised was not identified by name.

It would seem natural to say that, according to the representation, Susan's promise in `e3` has been (actually) kept by the delivery event `e16`. However, the earlier formulation of `actually_kept`, that is

```
actually_kept(E, E') ← promise(E), theme(E, E'), mode(E', actual).
```

does not work: the theme of `e3` is `e4`, not `e16`. What we require is a refined definition to express the intuition that "a promise is kept if something that *matches* what was promised becomes actual". In clausal form:

```
actually_kept(E, E') ←
                          promise(E),
                          theme(E, E'),
                          mode(E", actual),
                          matches(E", E'),
                          sake(E", E).
```

The predicate `matches` will be defined separately, below.

It seems to us that recording the occurrence of the delivery event simply by asserting:

```
mode(e4, actual).
```

is quite unworkable. For one thing, we would then be unable to record extra attributes of the delivery event, such as the name of the driver, the time of delivery, the person accepting delivery, and so on, without thereby altering the representation of what it was that Susan promised in `e3`. For example, adding

```
agent(e4, dave).
```

to the representation would indicate not only that Dave delivered the pizza but that Susan had promised that Dave would deliver the pizza (which she did not). Moreover, in the example, Susan's promise `(e3)` to deliver pizza `(e4)` was made in response to a request `(e1)` from Peter that pizza be delivered (`e2`). Why then record the actual delivery of the pizza by the assertion `mode(e4, actual)` and not by the assertion `mode(e2, actual)`? Why the difference between `e2` (the content of Peter's request) and `e4` (the content of Susan's promise) in the first place? In the representation style that has been adopted throughout, event naming constants are merely devices for building structured representations, whether they are viewed as Skolem constants as in FOL or as object identifiers as in C-logic. `e2` and `e4` and now `e16` are different event descriptions, for the same reason that the pizza identifier p4 in the description of event e4 cannot be the same as the pizza identifier p16 in the description of event e16.

How then do we determine that the actual occurrence of event e16, and the delivery of pizza p16, matches the event e4 promised by Susan and the promised pizza p4? As a first shot, we can at least require that the stipulated attributes of the promised event (Y in the definition below) are all features of the actual event (Z):

```
∀Z∀Y [matches(Z,Y) ↔
                    [delivery(Y) → delivery(Z)] ∧
                    ∀X[agent(Y,X) → agent(Z,X)] ∧
                    ∀X[recipient(Y,X) → recipient(Z,X)] ∧
                          :
                    ∀T1 ∀T2[theme(Y, T1) ∧ theme(Z, T2) →
                                 matches_object(T2, T1)]
    ]
```

Here matches_object will be defined in similar style to specify when two (here, pizza) descriptions match one another; again, as a first shot we can require that all the attributes of the stipulated (pizza) description T1 must be present in the actual pizza description T2. To be written in full these definitions of matches and matches_object require a list of the possible attributes for each type of object, but that is easily supplied.

Athough such definitions will be adequate for simple purposes, one can see that they are only approximations. First, suppose buyer X orders a pizza with mushroom, cheese and peppers topping, but is actually delivered a pizza that has not only mushrooms, cheese and peppers topping, but bacon and pepperoni topping too. Does the pizza match the specification of the pizza that was ordered? Of course, one could say that the value of the topping attribute of the ordered pizza is {mushroom, cheese, peppers} and the value of the topping attribute of the delivered pizza is {mushroom, cheese, peppers, bacon, pepperoni} and the values of these two attributes are not the same. (This cannot be distinguished in C-logic but could be done using some other variant.) But that is really beside the point. Suppose that in addition to the ordered toppings the pizza was delivered with slices of ham laid over the top. Would such a pizza be regarded as matching what had been ordered? Not if it had been ordered by a vegetarian perhaps. Or suppose that the promised pizza was supposed to come with mushrooms in the topping but when delivered it is found to contain a topping made almost entirely of cheese and peppers with just the tiniest slivers of mushrooms present. Would that pizza match the description of what had been ordered?

Second, deciding when to deem a promise as 'kept', an obligation fulfilled, even a pizza 'delivered', are in general far from trivial matters. Much of the litigation that arises in the course of a business exchange will centre on resolving conflicting views of what renders a promise 'kept'. Much of the detail in trading conventions, such as the UN convention governing the sale of goods, is concerned with spelling out the conditions under which an item is deemed to have been 'delivered', the time a message is deemed to have been 'received', and so on.

Of course it is difficult to imagine how some of these points could arise in connection with promises to deliver a pizza. But not all business messages will be about pizzas. Suppose Susan's promise had been about the delivery not of pizza but of some other very expensive perishable object. Or suppose Peter had requested and been promised the delivery of a thousand pizzas to be delivered every day at a specified time and place. In those circumstances we surely would be concerned with determining much more precisely the exact specification of each pizza, and what exactly it means to 'deliver'. In our experience of representing contracts in areas of construction and engineering [5], much of the detailed content of a contract was concerned with precisely this issue. And not only are the details spelled out, there are agreed mechanisms for testing that each item (each pizza for our example here) meets the specified standard, agreed equipment to be used for measurements, agreed arbitration mechanisms and so on. During the contract formation stage, the parties try to anticipate such eventualities and agree explicit terms that will determine the conditions under which a promise will be deemed as having been 'kept'. At the very least they will agree on the arbitration mechanisms to be used should any dispute arise during the performance of the contract. During the resolution of disputes, knowledge of common business practice, common sense, existing legislation, even precedent might be used to assist in establishing whether requirements have been met.

In general terms, then, the predicate matches is essentially a representation of the concept of 'counts as' discussed by Jones and Sergot [8] in the context of formalising qualification/classification norms, that is to say, the rules that specify the conditions under which, within a given context agreed by two or more parties (their contract), a promise is *deemed* as kept, or more generally, a state of affairs *A* is deemed to count, for the purposes of the agreement, as state of affairs *B*.

We are not suggesting that all this needs to be represented in an FLBC message. On the contrary, our understanding is that FLBC is intended for representation of routine business exchanges that take place *within some agreed context.* We have already made this point in the introduction. For simple messages about simple things, `matches` as defined above is going to be adequate. As we move to more complicated messages, we will need to consider whether to represent something of the procedures and mechanisms by which the performance of the business exchanges will be determined. The definition of `matches` will then become increasingly case-specific, and of arbitrary complexity. However, this does not mean that the formal representation of such mechanisms is necessarily *difficult*. In practice we see little difficulty in formalising the application-specific conditions that define `matches` (and hence `actually_kept`). Our point is that these conditions, which are required for the purpose of monitoring the performance of business exchanges, are not the same as the conditions that appear in the definition of `kept`, the device by which the content of a promise (or other type of speech act) is represented in FLBC.

6 Other extensions and open problems

The modified FLBC framework can be extended further in a number of ways, some of which are straightforward and do not pose significant technical or theoretical problems. For example, should it be desirable, times can be associated with events by extending the set of thematic rôles (attributes in C-logic formulations) and an underlying temporal framework may be used to reason about temporal relations between events and the states of affairs created by those events. The event calculus [19] fits very easily in the FLBC framework, for example.

For some applications, such as those aiming to support contractual activity, there may be value in having an explicit representation of the obligations that are assumed by a party as a result of a promise issued to a counter-party. One way of representing this explicitly is to use a framework such as the event calculus to specify the effects of promising events in terms of the obligations and other relationships that they initiate and terminate. Another way, simpler but less expressive, is via the use of general rules applicable to promissory messages, such as the one shown below:

```
∀x∀a∀r∀t[(promise(x) ∧ agent (x, a) ∧ recipient(a, r) ∧ theme(x, t)) →
      ∃y[obligation(y) ∧ bearer(y, a) ∧ counter_party(y, a) ∧ theme(y, t)]]
```

Similarly we could develop general rules to express the relationship between keeping a promise and fulfilling its associated obligation. We leave it to future research to determine the practical benefits of such extensions. When formulating constraints on what makes an exchange of messages well-formed, for example, it may be easier and more natural to frame some of these constraints in terms of obligations rather than in terms of the promising events that create them.

Apart from such relatively easy extensions, there are harder issues that need to be addressed. The content of a promise (and other types of messages) is often much more complicated than a single one-off event. For instance, in the terms of the pizza-ordering example, Susan might promise to deliver the pizza by motorbike if it is raining. She might promise to deliver either thick-base pizza or thin-base pizza with extra cheese topping, depending on availability. She might promise to deliver pizza every day at 7:00 pm, packed in red heat-insulated boxes. Although some of these examples are again rather fanciful in the context of a simple one-off pizza purchase, they are perfectly normal in other business settings. If Susan is a pizza manufacturer and Peter is a retailer, Susan might well promise to deliver a specified number of pizzas at a set time every day, to varying specifications depending on availability, and with more or less complex pricing arrangements according to the types of pizza delivered. In some of the sample engineering contracts we have previously examined [5], for example those concerning the supply of natural gas, the details of how, how often, what quantity, and what quality of gas is to be delivered were typically given by an extremely complex set of interrelated conditional statements and procedures. They reveal conditional promises, promises issued periodically, promises whose content is to be fulfilled periodically, and more complex constructs exhibiting all of these in combination. Indeed, in the example exchange of this paper, Susan promises to deliver a specified pizza by 7:30 pm, or if after 7:30 pm to deduct £1.00 from the bill. The content of this promise can be represented in different ways: as a single delivery event where the price depends conditionally on the time of the delivery. Or, better perhaps, as two separate promises, one (unconditional) to deliver a pizza, and a second, conditional one, to reduce the bill by £1.00 in case the time of the delivery is later than 7.30 pm. The second is arguably a more accurate representation of what Susan promised.

The treatment of simple conditional promises in FLBC is already an unresolved issue. In the original FLBC framework, the β component of the α-β-γ-δ scheme is intended to be used for representation of conditional utterances of the form "*x* promises that if β then δ (will occur)." Thus, when Susan promises "if fax confirmation is received by 7:00 pm then the pizza will be delivered by 7.30 pm", the β component would be a representation of the condition "if fax confirmation is received by 7:00 pm." Conditional elements of the promise itself, such as how price depends on time of delivery or how mode of transport depends on whether it is raining, can be accommodated within the δ component. And likewise for the representation of conditional requests ("*x* requests that if it is raining *y* should send a taxi"), conditional directives, and other speech acts.

The β component, empty in all the examples discussed earlier, does not hinder translation to the clausal version of the α-β-γ-δ scheme: the clauses of the representation simply include additional conditions corresponding to β. So, for illustration, if the promise (e3) made by Susan had been a conditional one, of the form "if fax confirmation is received by 7:00 pm then the pizza will be delivered by 7:30 pm", the clausal representation (qK3′) would come out as follows:

```
delivery(E) ← kept(e3, E), confirmation_received(e3).
agent(E, susan) ← kept(e3, E), confirmation_received(e3).
beneficiary(E, peter) ← kept(e3, E), confirmation_received(e3).
theme(E, f) ← kept(e3, E), confirmation_received(e3).
sake(E, e3) ← kept(e3, E), confirmation_received(e3).
```

Here `confirmation_received(e3)` stands for the condition "fax confirmation is received by 7:00 pm." There is nothing problematic about expressing this condition in full in the FLBC framework. We have omitted the details so as not to distract from the main point we are seeking to make. (The argument `e3` is necessary to write the conditions out in full.)

In the simplified form corresponding to clauses (qK3″), where the 'kept' condition has been eliminated (and replacing `th(e3)` by an explicit 'theme' assertion in the style we prefer), we obtain:

```
theme(e3, e4).
delivery(e4) ← confirmation_received(e3).
agent(e4, susan) ← confirmation_received(e3).
beneficiary(e4, peter) ← confirmation_received(e3).
theme(e4, f) ← confirmation_received(e3).
sake(e4, e3) ← confirmation_received(e3).
```

This is all quite straightforward. There is a problem, however: we are again faced with the question of how to *retrieve*, from the representation, a description of what it was that Susan promised. A query such as

```
?- theme(e3, X)
```

which worked previously when β was empty now produces at best an answer qualified by conditions β′ logically equivalent, but not necessarily identical, to β. All the points we made earlier in section 4 about the inadequacy of this kind of retrieval apply equally here. Similarly, any conditional elements in the δ component, such as varying prices, modes of delivery, and so on, can be *expressed* in the framework, but cannot be *retrieved* by querying the representation.

Notice that for the purposes of monitoring whether promises have in fact been kept, there is no problem: whether the promise has actually been kept will depend only on whether it is possible to determine from the representation that conditions β are established, and there is nothing problematic about that. Notice also that when we are interested in monitoring whether promises have in fact been kept, there is no need to distinguish in the representation between what we have been calling a conditional promise, a message or utterance of the form

x promises that if β then δ (will occur)

and a conditional expression about the making of an unconditional promise, of the form

if β then *x* promises that δ (will occur)

Clearly there is a difference between these two forms of expression. The difference can be ignored if the only purpose of the representation is to support the monitoring of message exchanges. If the purpose of the representation is to support the storage and retrieval of messages, however, then the difference cannot be ignored: we have to be able to tell whether the message is of the first or second form. And since retrieval by logical deduction cannot be supported (as we have argued), conditional utterances of the first form remain problematic, even for the α-β-γ-δ scheme.

A possible solution is to express the content of a conditional promise (request, directive, assertion, and so on for other types of messages) as another kind of structured object, an object with attributes pre-conditions (say) and consequent, both of which have values which are (representations of) FOL formulas. This is an ugly and unappealing approach, however, which we are not proposing to pursue.

7 Conclusion

We have made two suggested modifications to the original FLBC representation scheme: first, to eliminate the need for the problematic modal logical component by replacing quantification over possible worlds by quantification over possible events, and second, to re-write in Skolemised clausal form in order to enable the application of readily available computational methods. An optional further modification, which we prefer, allows one event description to be recorded as the 'theme' ('patient' or 'direct object') of another event. Kimbrough's latest formulation [14] suggests that he agrees that such modification is useful. The use of a syntax such as that of C-logic can also be useful in the construction of complex structured representations.

We have pointed out two different computational tasks, with differing computational and representational requirements. Task 1 is to retrieve from the FLBC representation of a message or utterance all the components of interest, including in particular a representation of what has been promised, requested, commanded, asserted, as the case may be. Task 2 is to determine, given a representation of what events have actually occurred, whether promises have in fact been kept, requests honored, directives fulfilled, and so on, in order to provide a system for monitoring the performance of business and message exchanges. For simple messages, both of these computational tasks can be supported without difficulty using standard computational methods.

We have also discussed some further extensions and some open problems. The first of the computational tasks—the retrieval of message components from their representation—raises unresolved problems concerning the representation of conditional messages/utterances, that is to say, in speech act terms, utterances whose propositional content contains conditional (and other more complex) constructs. Although simple instances can be expressed in the FLBC scheme, they cannot be retrieved using logical querying methods. For the second computational task, the monitoring of message exchanges, the existing FLBC representation copes adequately with conditional utterances but raises a different set of questions about how to determine whether one event, typically one that has actually occurred, matches the description of another event, typically one representing the propositional content of a promise, request, directive, or other speech act. The resolution of these two sets of problems identifies directions for future developments.

Acknowledgments: We should like to thank Steve Kimbrough for many valuable discussions and an anonymous reviewer for helpful comments on the ideas presented in this paper.

References

[1] J.L. Austin . *How to Do Things with Words*, (1975 edition), Clarendon Press, Oxford.
[2] K. Bach & R.M. Harnish. *Linguistic Communication and Speech Acts*. MIT Press, 1979.
[3] E. Charniak E. & D. McDermott D. *Introduction to Artificial Intelligence*. Addison-Wesley, 1985.
[4] W. Chen & D. Warren (1989). C-logic of Complex Objects. In *Proceedings of the 8th ACM SICACT-SIGMOD-SIGART Symposium on the Principles of Database Systems*, 1989.
[5] A. Daskalopulu A. Logic-Based Tools for the Analysis and Representation of Legal Contracts. Doctoral Dissertation, Department of Computing, Imperial College, University of London, 1999.
[6] C.J. Fillmore. The case for case. In E. Bach & R.T. Harms (eds) *Universals in Linguistic Theory*. Holt, Rinehart and Winston, New York, 1968, pp. 1–88.
[7] C.J. Fillmore. The case for case reopened. In P. Cole & J.M. Sadock (eds) *Syntax and Semantics 8: Grammatical Relations*. Academic Press, New York, 1977, pp. 59–81.

[8] A.J.I. Jones & M.J. Sergot. A Formal Characterisation of Institutionalized Power. *Journal of the IGPL* 4(3):429–445 (1996). Also in: Valdés, Krawietz, von Wright & Zimmerling (eds.), *Normative Systems in Legal and Moral Theory*, Duncker & Humblot, Berlin, 1997.

[9] F.N. Kesim. Temporal Objects in Deductive Databases. Doctoral Dissertation, Department of Computing, Imperial College, University of London, 1993.

[10] M. Kifer & G. Lausen. F-logic: A Higher-Order Language for Reasoning about Objects, Inheritance and Scheme. In *Proceedings of the 8th ACM SICACT-SIGMOD-SIGART Symposium on the Principles of Database Systems*, 1989, pp. 134-146.

[11] M. Kifer & J. Wu. A Logic for Object-Oriented Logic Programming (Maier's O-logic revisited). In *Proceedings of the 8th ACM SICACT-SIGMOD-SIGART Symposium on the Principles of Database Systems*, 1989.

[12] S.O. Kimbrough. On Electronic Commerce, Subatomic Semantics and the Cæsar's Stabbing. In Sprague (ed.) *Proceedings of the 30th Hawaii International Conference on Systems Sciences*, Los Alamitos CA, IEEE Computer Society Press, 1997.

[13] S.O. Kimbrough. Formal Language for Business Communication: Sketch of a Basic Theory". *International Journal of Electronic Commerce*, 3(2):23–44 (1998).

[14] S.O. Kimbrough. Reasoning about the Objects of Attitudes and Operators: Towards a Disquotation Theory for Representation of Propositional Content. *Proceedings of the 8th International Conference on Artificial Intelligence and Law,* St Louis, Missouri, ACM Press, 2001.

[15] S.O. Kimbrough & R.M. Lee. On Illocutionary Logic as a Telecommunications Language. *Proceedings of the 7th International Conference on Information Systems*, San Diego, CA, 1986, pp. 15–26.

[16] S.O. Kimbrough & R.M. Lee. Formal Aspects of Electronic Commerce: Research Issues and Challenges. *International Journal of Electronic Commerce*, 1(4):11–30 (1997).

[17] S.O. Kimbrough & S.A. Moore. On Automated Message Processing in Electronic Commerce and Work Support Systems: Speech Act Theory and Expressive Felicity. *Transactions on Information Systems*, 15(4):321–367 (1997).

[18] R.A. Kowalski. *Logic for Problem Solving*. Elsevier Science, New York, 1979.

[19] R.A. Kowalski & M.J. Sergot. A logic-based calculus of events. *New Generation Computing*, 4:67–95 (1986).

[20] D. Maier. A Logic for Objects. *Proceedings of the Workshop on Foundations of Deductive Databases and Logic Programming*, Washington DC, 1986, pp. 6-26.

[21] T. Parsons. *Events in the Semantics of English: A Study in Subatomic Semantics*. Current Studies in Linguistics, MIT Press, Cambridge MA, 1990.

[22] J.R. Searle. *Speech Acts*. Cambridge University Press, Cambridge, 1969.

[23] J.R. Searle. *Expression and Meaning*. Cambridge University Press, Cambridge U.K., 1979.