Specification of Cooperative Constraints in Virtual Enterprise Workflow

Anne H.H. Ngu

University of New South Wales
School of Computer Science and Engineering
Sydney 2052
NSW, Australia
email:anne@cse.unsw.edu.au

Abstract

Workflow systems are an emerging technology which have become increasingly important in the drive for business to provide better services and increase productivity. Intuitively, workflow applications are processes which automate and regulate the movement and execution of a number of work, across one or more servers, according to business defined rules and routes. Lacking in the current workflow products is a way to specify both the temporal and the obligations constraints that are inherent for effective and flexible flow of information among the activities.

We propose using propositional temporal logic for specifying and reasoning over the temporal constraints in workflow's activities and complement it with deontic logic which is used to specify commitments of participants in the course of the workflow. These two approaches are combined by enforcing all messages to be communicated using a pre-defined speech act primitives. By capturing these two important constraints in workflow, we aim to provide a model that can react and adapt to organizational changes in a controlled way. We demonstrate our specification using a virtual conference planning workflow.

1 Introduction

Workflow applications are processes which automate and regulate the movement and execution of a number of work, across one or more servers (could be a database, a knowledge base, a document server or a WWW service), according to business defined rules and routes. A workflow also typically defines the individual business activity steps, the order and the conditions under which the activities must be executed, the flow of data between activities, the users responsible for the conduct of these activities and the tools used.

As an example of a workflow application, consider the emerging concept of *virtual healthcare enterprises* [22] which interlinks geographically distributed and organizationally disparate hospitals, private practices, clinics, pharmaceutical companies, tertiary care centers that have a legacy of different approaches to providing healthcare. The objective

is to loosely couple these groups and coordinate the administrative, financial and clinical processes of such enterprises, while controlling costs and providing quality healthcare. To meet these requirements, the different patient care and other medical processes in the healthcare industry must be streamlined and the data must be integrated and coordinated. The exchange of electronic business data and documents and the use of standardized clinical, administrative, insurance and financial transactions, can reduce cost and improve the productivity of physicians and health care enterprises.

If a group of enterprise servers are to cooperate to achieve certain business tasks, we need to be able to specify both temporal and obligation constraints among them (i.e. a well established protocol to guarantee the exchange of information).

We aim to develop a declarative model for specifying the temporal constraints in workflow and a mechanism for specifying the obligation constraints among participants in workflow. By integrating the temporal and the obligation constraints, we aim to arrive at a model that is flexible and will enable organisations to rapidly respond to changes in business processes. Effectively, the model provides a way to perform reasoning over alternatives when an obligation is violated to reduce the aborts and recovery in workflow.

We view workflow process as consists of a set of activities or transactions. Each activity (similar to task) consists of a set of valid messages that can be exchanged among the participants (such as different web resources) that participate in the activity. Each activity also has a goal and an exit state. The set of participants communicate by exchanging messages that obey certain temporal constraints (in the sense that message A can only be sent after receiving message B). Each message contains a speech-act primitive and a proposition which could be another sub-activity. A workflow process is valid if the temporal constraints among the activities are consistent, that is there is at least one possible execution trace. We adopted Propositional Temporal Logic (PTL) for modeling the coordination or temporal constraints for exchange of messages among the participants. Each of our activity is an aggregation of some subactivities (messages) which are executed cooperatively by a few agents. This has the advantage of being more intuitive from the designer point of view. It is much easier to be able to specify complex workflow process in a goal-directed way with each goal being collectively achieved by cooperating agents.

Currently, there is no single logical framework to specify and enforce both temporal and obligation constraints in an workflow's activity. We believe that obligations can be extracted and then explicitly modeled in deontic logic if messages are sent and received in an activity using a set of well defined speech-act primitives. We thus restrict messages that will generate a deontic effect to be sent or received with specific speech-act primitives. This allows us to model the obligation constraints among the participants. For example, a message sent with an `order` speech-act obliged the sender to pay for what is being ordered. The same message sends with a `request` speech act does not impose such an obligation. We use deontic logic to document the deontic effect of a speech-act operator (i.e. obligations). The major contribution in this paper is demonstrating that PTL which is a logic oriented towards reasoning about sequences is suitable for specifying all types of dependencies in activities in workflow. Deontic logic can be used to specify and reason about "obligations", i.e., commitments of participants in the course of the workflow.

This paper is organized as follow. In section 2, we discuss why the need to enforce both temporal and obligation constraints in workflow and how these can be specified by

using PTL and deontic logic separately. In section 3, we show through some activities in the virtual conference planning workflow how temporal constraints can be specified using PTL. In section 4, we demonstrate how to extract and document the obligation constraints from the activity's messages in deontic clauses. Section 5 outlines our conclusion and future work.

2 How temporal and obligation constraints can be specified?

The main contribution of our previous work in workflow modeling is the focus on communication and the declarative form of specifying messages coordination (temporal aspect only) in PTL which can be verified formally by a dependency graph [17]. We name this specification as *interoperable transactions*. Our interoperable transaction is essentially the same as an activity in a process workflow model. Here, we only specify the temporal constraints and thus have to assume that all the participants will always honour their obligations, i.e the communication is always reliable. Thus any failure in communication is always treated as a "true failure" regardless of its type. This causes unnecessary rollback and recovery. An unreliable communication resulted from a cooperating participant not fulfilling his/her obligations should be treated as a violation, and should not require a rollback of the activity. For example, if an airline promises a flight at a particular time, and canceled the flight due to bad weather, we said that the airline did not fullfil its obligations, the system should reason over this violation and trigger off appropriate actions, for example, putting the passengers on the next available flight and thus avoid having to abort the activity.

Currently there is no single logical framework available to specify and enforce both temporal and obligation constraints. One possibility is to extend the PTL to include the modeling of obligation constraints. However this will complicate the specification of the temporal constraints and lead to the violation of the completeness of the tableau's verification method and thus the generation of dependency graph. Moreover, it will complicate the declarative PTL specification and also lower the reusibility of the workflow's activity specification.

The speech act theory describes the illocutionary force of the message (assertive, directive, commissive or declarative) together with its authorization claim (power, authority or charity), and a content (a proposition or action). An assertive speech act simply makes a statement about the state of affairs in the world and commits the speaker to the truth of the expressed proposition. A directive speech act tries to get the addressee to do things. A commissive speech act commits the speaker to a future course of action. A declarative speech act brings about some new state of affairs of the world.

By providing a standard set of speech act primitives for communication in workflow, we can have a pre-defined semantics in terms of obligations and authorizations for messages exchanged between participants. For example a message such as `request(reserve(airline-ticket))` involves a directive speech act `request` and the action `reserve` from the sender to the receiver. By enforcing all messages that carried deontic effects to be exchanged via pre-defined speech-act primitives, we can ex-

tract the obligation constraints and document it using deontic logic.

Deontic logic [25] is a form of dynamic logic with two operators, Obl(i,α) and Aut(i,α), for `obliged` and `authorized`, respectively. The semantics of an authorized request to do α is that Obl(i,α) holds (as a postcondition) provided that Aut(i,α) holds (as a precondition). Obl(i,α) means that if a system i does not perform α, then this leads to a `violation`. Independent rules specify what such a violation implies. These rules are usually expressed in terms of other Obl and Aut formulas.

Deontic logic has been used in the context of modeling legal contracts. In [16], deontic logic is used to model the notion of permissions in legal applications. We found that when used in the context of workflow, the three primitive operators Obl(i, α), Aut(i,α) and Acc(i,α) are needed. This stands for obliged, authorized and accomplished respectively. Acc(i,α) indicates that agent i has carried out the stated action α.

The following are the set of speech-act primitives whose deontic effects (obligations) can be defined using deontic logic. These set of speech act primitives are first proposed in [10]. This is not by all mean an exhaustive list.

Message :
| Request (action) /*directive charity*/
| Command (action) /*directive authority*/
| Order (action) /*directive power*/
| Commit (action) /*commissive*/
| Suggest (proposition)/*assertive charity*/
| Assert (proposition) /*assertive authority*/
| Claim (proposition) /*assertive power*/
| Nominate (proposition)/*declarative charity*/
| Declare (proposition)/*declarative authority*/
| Establish (proposition)/*declarative power*/
| Propose (proposition) /*declarative power*/
| Authorize (message)
| Permit (action)
| Confirm (action)
| Promise (action OR Proposition)/*commissive power*/
| Request (Authorize (action))
| Assert (Accept (action OR Proposition))
| Assert (Refuse-to (action, "reason")

The semantics of speech act `order` generates the following obligations. The clause [α]p means that after action α, p holds. The expression NOT α stands for the non-performance of the action α. The clause `obl(i,j,payfor(goods))` stands for i obliged to j for action `payfor(goods)`.

/* if i ordered the good from j and j shipped the goods
/* i is obliged to pay j for the good
order(i,j,goods) => [ship(j, goods)] obl(i, j, payfor(goods))

/* if j is obliged to ship the goods to i and he does not do it,
/* he is obliged to pay a fine and the other party i is
/* authorized to request (other) goods from other source k
Obl(j,i,ship(goods)) => [NOT ship(j,goods)]
Obl(j,i,pay($100)) AND Aut(i,order(i,k,goods)))

The semantics of speech act `command` and `order` are similar, they are both directive speech acts, but only person with certain authority can use the speech act `command`. The `command` speech act automatically creates the obligations for the receivers to carry out the action with high priority. When compared with the speech-act `order`, the `request` speech act does not express the need to have the propositional contents of the speech act be realised by the addressee. For example, there is no need to provide an answer to a request, unless the speaker has an authority (or special relationship) over the addressees.

By including `obligation` constraints in the communication model of workflow, we aim to deliver a mechanisms which emulate the way human beings interact. It is known that the coordination behaviour among agents (human being/computer system) always requires some form of agreement and mutual commitment. If for whatever reason, an agent does not execute an action it has itself committed to, this causes a violation of agreement. When there is a violation of obligation, a rescheduling of activities in the workflow is achieved by reasoning over the consequences of those violations. Our model will be able to react "actively" to dynamic changes of workflow (when participants in the workflow behave less than ideally). This provides a crucial flexibility in execution of workflow, which is not possible with traditional workflow technology as described in [20].

3 Examples of temporal constraints specification

Consider as an application the communication between participants involved in planning a conference for hosting an International Data Engineering Conference. Since it is an international conference, the executive committee members can come from different parts of the world. For example, the conference location could be in Sydney, while the program chairs could come from Europe, Asia and USA. Thus part of the administrative work will be done in Sydney and part of it will be done in Europe, USA or Asia. Information pertaining to the conference can come from different parts of the world. For example the information regarding paper submission could be stored in a web server located in Europe, while the financial and registration information are stored in Sydney. The papers submitted should be accessible to the assigned program committee members in any part of the world. The result of reviews by different program committee members could be processed by any of the three program chairs. Authors of the submitted papers should be able to check the status of their papers from any part of the world too. For the organization of the conference to be successful, it is critical that committee members cooperate and adhere to the specified roles and deadlines.

The following are example specifications of activities for an IEEE conference. Keywords are in boldface. The activity `Plan-Conference` is an aggregation of five sub activities. The messages within an activity are sent with appropriate speech-act primitives. Each activity is aimed at performing a specific task (goal). The constraints represent the synchronisation of the activities or messages. We use the `Plan-Conference` , `Conference-Registration` and `Reserve-Conference-Venue` to illustrate our modelling framework. The details of other activities can be found in the appendix.

In the activity `Plan-Conference`, the constraint "˜Plan-publication `UNTIL` Plan-

technical-session" indicates that the task for printing the actual proceedings can only happen after the reviewing of the papers and the details of the final technical program is being finalised. The absence of any constraint between two activities imply they can be executed in any order. For example `Reserve-Conference-Venue` and `Plan-publicity` are independent. The `Goal` statement specifies what make the activity successful and the `Exit` statement specifies what make the activity failed or aborted. In the activity `Plan-Conference`, both the sub-activity `Plan-technical-session` and `Reserve-Conference-venue` must succeed for this activity to be declared a success. `Plan-Conference` activity fails, if either one of the sub activity `Reverve-Conference-Venu` or `Plan-technical-session` fails.

Activity Plan-Conference
aggregate of:
Conference-Registration
Plan-technical-session
Plan-publicity
Plan-publication
Reserve-Conference-Venue
Constraints:
/*˜ stands for NOT
/* No constraint is specified between Reserve-Conference-venue
/* and Plan-publicity. Thus they can occur in any sequence
˜Plan-technical-session `UNTIL` Reserve-Conference-Venue
˜Plan-technical-session `UNTIL` Plan-publicity
˜Plan-publication
`UNTIL` Plan-technical-session
/* Do not start the registration process until call for participation
/* notice is sent
˜Conference-Registration
`UNTIL` Plan-publicity.send(call-for-participation)
Goal = {Plan-technical-session AND
Reserve-Conference-venue}
Exit = {˜Plan-technical-session OR
˜Reserve-Conference-venue}
End Activity

Activity Conference-Registration
agents: t:treasurer, p:publication-chair, r:registrar-chair,
c:technical-pg-chair, d:delegates,
r can send messages:
request(list-of-authors) to c
print(labels-for-delegates) to self
request(get(proceedings)) to p
assert(register(delegate-in-advance)) to self
register(delegate-on-site) to self
request(payment) to d
confirm(register(delegate)) to d
assert(deliver(payment)) to t
assert(cancel(registration)) to t
c can send messages:
confirm(list-of-authors) to r
p can send messages:
confirm(deliver(proceedings)) to r

t can send:
authorize(refund-to(delegate)) to d
confirm(deposit(payment)) to r
d can send:
request(registration) to r
confirm(payment) to r
request(cancel(registration)) to r
Constraints:
˜print(labels-for-delegates) UNTIL
confirm(list-of-authors) AND
confirm(register(delegate))
˜register(delegate-on-site)
UNTIL assert(register(delegate-in-advance))
/* delegates can send a request to cancel registration after
/* request for registration
ALWAYS (request(registration) => SOMETIMES(request(cancel(registration)))
/* delegates can send a request to cancel registration after
/* sending the confirm(payment) message
ALWAYS (confirm(payment) => NEXT (TRUE OR request(cancel(registration)))
Goal = confirm(register(delegate))
Exit = {request(cancel(registration)) OR
(˜ confirm(payment))}
end Activity

Activity: Reserve-conference-venue
agents: h:hotel, t:treasurer, a: organizing-chair
a can send:
request(quotation-for-venue) to h
request(approve(cost-for-venue)) to t
request(reserve(venue)) to h
assert(pay(deposit)) to h
h can send:
send(quotation-for-venue) to a
reserve(venue) to self
confirm(reserve(venue)) to a
assert(no_available(venue)) to a
promise(venue) to a
t can send:
confirm(approve(cost-for-venue)) to a
confirm(refuse(cost-for-venue)) to a
Constraints:
˜request(reserve(venue)) UNTIL
request(confirm(approve(cost-for-venue))
˜confirm(reserve(venue)) UNTIL reserve(venue)
˜assert(send(venue-details))
UNTIL confirm(reserve(venue))
ALWAYS (request(reserve(venue)) =>
NEXT (SOMETIMES(confirm(reserve(venue))
XOR assert(no_available(venue)))
Goal = confirm(reserve(venue))
Exit = confirm(refuse(cost-for-venue))
XOR assert(no_available(venue))
End Activity

The registration chair has to coordinate with the treasurer (to deposit the payments from delegates), with delegates (to confirm registration and ensure receive of payment), with publication chair (to get the proceedings for distribution and pass the information about which authors who have not pre-registered), with program chairs (about technical program for printing suitable labels for delegates). Different kinds of information is needed at different period of time. For example, printing of labels only need to happen a week before conference. However, it must occur after knowing who are the presenters, the session chairs etc from the technical-programme.

Each sub activity consists of messages that can be exchanged among the participants to accomplish a task in that activity. It is a non-trivial task to write correct communication constraints between many participants. Thus a declarative specification of constraints should be used. Propositional Temporal Logic (PTL) [26] is chosen for specifying the constraints among the participating agents. Our initial result of using PTL for this purpose is reported in [17]. The temporal operators `SOMETIME`, `NEXT`, `ALWAYS` and `UNTIL` provide the necessary semantics to cover the various kinds of dependencies between events/messages in a communication process. For example, `SOMETIME` conveys the obligation to honour an event in a future state. The `NEXT` operator is similar to the concept of `trigger` used in active databases; it guarantees that an event occurs after the current one. The PTL formula `~B UNTIL A` (`~` is the symbol for `NOT`) expresses the constraint that the event B cannot happen until event A has happened. The unary operator `ALWAYS` is used to express cyclic sequences. For example, `ALWAYS C` means that the PTL formula C will always have the value `TRUE`. This is because the condition C is always being regenerated. The following example shows two mutually exclusive events `A` and `B` specified in PTL (A_trigger stands for the event that triggers A). The corresponding graph is shown in figure 1:

```
ALWAYS (A => NEXT (~(A OR B) UNTIL A_trigger))
ALWAYS (B => NEXT (~(A OR B) UNTIL B_trigger))
```

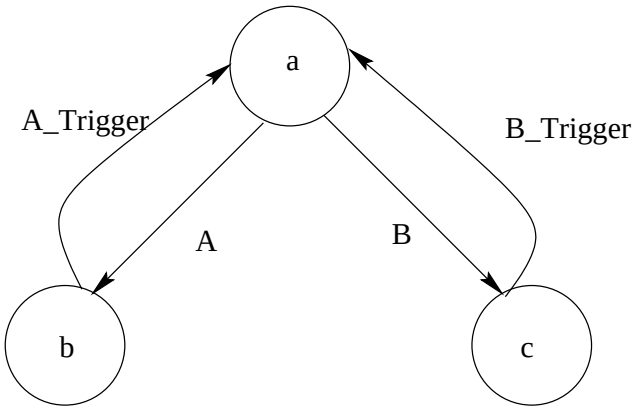

Figure 1: Dependency graph for Mutual Exclusive events

PTL specifications can be verified formally by using the well known tableau method. Such a verification system has been implemented using Allegro Common Lisp in [9].

We can easily use it to verify each of the above sub activities. A dependency graph for `Plan-Conference` is shown in figure 2.

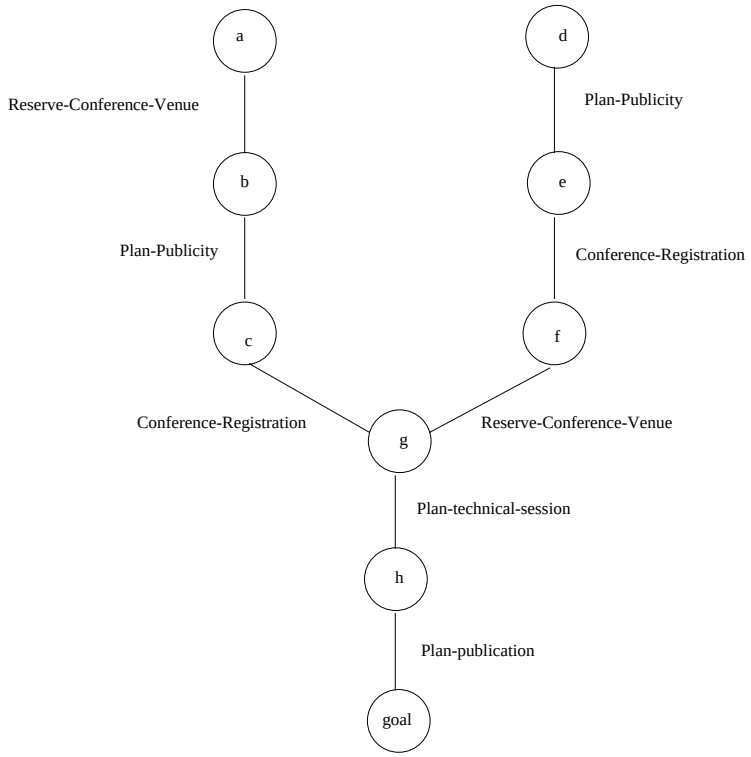

Figure 2: Dependency graph for Plan-Conference

Note that the specification of messages is in functional style. It starts with one of the speech act primitive and ends with an action or a proposition. For example in the `Reserve-conference-venue` activity, the fact that a hotel `promise` a particular venue for the conference, and conference organizer had paid the deposit to secure it are modeled by `promise(venue)` and `assert(pay(deposit))` respectively. The occurrence of `promise(venue)` will create the obligations for the hotel to hold the venue (i.e., not to give the requested venue to another customer) for the conference organizer and the occurrence of `assert(pay(deposit))` remove this obligation, but create a different obligation for the hotel towards the organizer. We illustrate how to extract obligation constraints from the messages embedded in a speech-act primitive and explicitly represented them in deontic clauses so that it is possible to reason over it in the next section.

4 Example of obligation constraints specification

The need for specifying obligation constraints in the context of our communication-based workflow model has been discussed in section 2. The logic to reason about obligations and related concepts is called deontic logic. In theory, two types of obligations can be distinguished: the temporal constraints interpreted in a deontic sense, i.e., the responsi-

bilities or the role of participants involved in accomplishing a task; and the obligations that are created by the basic speech act primitives while communicating for actions. We only deal with the obligation semantics of the later type. In [24], we introduced the notion of contract to document obligations and deal with violation of obligations using deontic logic. It is not clear how to capture both temporal and obligation constraints in one framework. In this paper, we indirectly force users to specify obligation semantics by providing a standard set of speech-act primitives to be used for communication among the participants of an activity. It is possible to limit the type of obligations that we are dealing with and it is possible to document and track them. In our context, any message sent without using the authorized set of speech-act primitives implies no inherent obligation involved between the sender and the receiver.

Thus the first step in our workflow modelling is to determine what types of speech-act primitives are required for the communication. We then go through an activity and extract the obligations of both service providers and service receivers by checking what type of speech act are used by them to communicate with one another. The type of speech-act used bind them to certain type of obligations. We document the obligations in terms of deontic clauses (which is a conjunction of deontic formula).

The following example illustrates the obligations that arise between `hotel(h)` and the `conference organizer(o)` in the activity `Reserve-conference-venue`. We use a set of deontic clauses to track the state of the obligations. Each deontic clause is reached by one or more messages and is left again by other messages.

```
/*obligation of hotel in sending the message promise(venue)*/
s0: obl(h, keep(reserved_venue))
    in promise(venue)
    goal assert(accept(conference_guest_booking)) goto s4 & s1
    exit
        (o, request(cancel(reserved_venue))) goto s2
        assert(cancel(reserved_venue)) goto s3

/*obligation of conference organizer in accepting the
/*quotation for the venue*/
s1:obl(o, pay(deposit))
    in promise(venue)
    goal (h, assert(receive(deposit))) goto s7
    exit
        declare(cancel(reserved_venue)) goto s2

/*obligation of hotel to refund the deposit if cancellation from
    conference organizer is with sufficient notice*/
s2:obl(h, refund(deposit))
    in declare(cancel(reserved_venue))
    goal (o, collect(deposit)) goto s0
    exit
        (o, assert(cancel(with_no_sufficient_notice))) goto s6

/*authorization that hotel gives to conference organizer
    in the event that hotel withdraws the venue*/
s3: aut(o, request(alternative_venue) & collect(deposit))
    in assert(cancel(reserved_venue))
```

goal (h, confirm(reserved_venue)) goto s0
exit
declare(cancel(reserved_venue)) goto s2

/*accomplishment of hotel in promise(venue)*/
s4: **acc**(h, keep(reserved_venue))
in assert(accept(conference_guest_booking))
goal goto s0

/*obligation of hotel in venue cancellation */
s5: **obl**(h, pay(fine))
in assert(cancel(reserved_venue))
goal goto s0

/*authorization that conference organizer gives to hotel
/*when insufficient notice is given
s6: **aut**(h, permit(keep(deposit)))
in(o, assert(cancel(with_no_sufficient_notice)))
goal goto s0

/*hotel obliged to provide the venue after receiving the deposit
s7: **obl**(h, provide(venue))
in assert(receive(deposit))
goal goto s0
exit goto s5

The obligation is identified by a unique state number such as `S1, S2, S3 and S4`. The `in` part refers to the speech-act action or message that leads to this state. The `goal` and `exit` have the effect of moving to another deontic state (a different obligation). The current state is no longer valid.

The speech-act `promise` commit the hotel(h) with obligation labelled as state `s0`. This obligation is removed by the goal state i.e., occurrences of action `assert(accept(conference_guest_booking))`. This obligation is violated by the exit state i.e., if hotel unilaterally canceled the promised venue. The violation will trigger some other obligations, in this case state `s3` which states that the conference organizer is authorized to seek for an alternative venue and collect the full refund of deposit. Alternatively, the conference organizer may seek to cancel the conference completely and hotel is obliged to pay a penalty (`S5`).

The difference between the goal and the exit is that one involves the fulfillment of the obligation whereas the other involves a violation or non-fulfillment. The fact that the hotel promised to keep the `reserved_venue` obliged the conference organizer(o) to pay the deposit. This corresponds to obligation labelled as `s1`.

5 Related Work

Various formalisms have been proposed in the past for the specification of constraints between objects. In [14], the concept of `activity` is used to specify the communication behaviour and interactions in the framework of object oriented databases. In the `event` model by King and McLeod [11], a finite state diagram is used to model communication

paths. In the 90's, the Flexible Transaction Model [5] has been proposed for the specification of coordination in multi-systems applications. More recently, coordinative workflow specification based on process algebra is proposed in [1].

Systems like the METEOR [18], TSME [7], Tractor [8], ASSET [2] take the approach of providing sublanguage primitives for specifying the coordination in workflow and the corresponding execution environment for application-specific workflow systems. However, we want to be able to specify and verify constraints in workflow formally. PTL has a well proven verification method. Primitives used in the above systems tend to be ad hoc and informal, it can't enforce the correct execution of workflow transactions. That is, the correct behaviour of those primitives cannot be ensured without a formal approach.

Our choice of using the Propositional Temporal Logic for specifying the temporal constraints in workflow not only has the advantage of being declarative but also has the added benefit of having a well-established verification method. Furthermore, by using the full Propositional Temporal Logic system as a specification language, we avoid imposing any practical restrictions on the types of dependencies that can be expressed. For example, it is not possible to specify cyclic sequences such as A, B, A, B, ... Dependencies such as *A will always follow B, A will never follow B* and *B will never follow A* are not catered for. The temporal operators such as `ALWAYS` and `SOMETIMES` provide the means to specify both partial order and cyclic dependencies in our approach. Thus we are not restricted to a directed graph structure. In [9] PTL has been shown to be adequate to express all the time interval operators as proposed by Allen and it has been used to successfully generate concurrent communication systems.

A number of declarative approaches to workflow modelling have also been proposed in [21, 3, 19]. However, none of these approaches address the issues of obligation constraints and in our view lack a mechanism to react and capture dynamic changes in workflow processes. For example, workflow is being modelled, verified and executed in one framework in CTR (Concurrent TRansaction Logic). This implies that CTR is the ultimate workflow engine for the workflow. It is unclear how this approach can be extended to dynamically modify the execution sequences in reaction to new constraints (violation of obligations).

The obligation constraints, which is imperative for workflow systems to be reactive to unanticipated changes need to be specified. A pioneering work in specifying obligations aspects of workflow is found in ActionFlow which makes use of the speech-act theory [15]. This has the advantage that obligations can be stated in a concise manner and we can perform reasoning with the specification. Obligations that cannot be fullfilled can be detected and resolved. However, system that emphasises on obligations tends to ignore the temporal constraints.

Our emphasis is on the modelling of both temporal and the obligation aspect of workflow. We are not addressing the computation or transactional aspects of workflow. In fact, our ultimate goal is to specify, validate and then translate our specification into a format that can be imported and executed in any open-nested transaction management facilities like in [5], [4], [13], [6], [12], [23].

6 Conclusions & Future Works

In this paper, we have outlined an approach to workflow modeling which makes use of both temporal and deontic logics. The emphasis is on the modeling of both temporal and obligation constraints. We demonstrate our workflow modeling using a virtual conference planning example. We believe that though temporal constraints alone can capture all the coordination constraints in a workflow, coordination constraints is not sufficient to make the workflow reactive to business environment which requires a continuous adaptation to possible ”hiccups” along the process.

We believe that the correctness of any specification for controlling the data flow in a workflow system is of vital importance, as any apparently minor, local problem can have disastrous effects which are distributed among the communicating activities. The writing of such specification requires great attention to detail and is prone to errors. The use of propositional temporal logic not only has the advantages of being more declarative as compared to the flexible transaction model [6], but also lends itself to a well-proven verification method. Furthermore by using the full propositional temporal logic system as specification language, we avoid imposing any practical restrictions on the types of dependencies that can be expressed.

There aren’t any single logical framework that can captures both temporal and obligations constraints. We propose to capture obligation constraints in workflow through the use of a pre-defined set of speech-act primitives for communications. It is possible to declaratively specify obligations inherent in speech-act primitives using deontic logic. For example, a speech act `order` and `request` can be specified as having different deontic effect. Thus messages that are sent with `order` can be interpreted different from messages that are sent with `request`.

When there is a failure in communication in the workflow (this means an agent did not act upon the message or an agent unable to act upon the message because of new constraints), it is possible to analyse the situation using the underlying deontic contract between the communicating agents. If there is a violation of obligation, specific rules can be followed at that point (new workflow instance started, compensating task launched etc).

Propositional temporal logic has been shown to be appropriate for specifying coordination constraints at the activity level, while deontic logic is shown to be suitable for specifying the obligation constraints. Currently, the extraction of obligation is done manually. One immediate work is to investigate how to efficiently extract obligation constraints from the speech-act primitives.

Appendix

Some examples activities from the virtual conference planning workflow:

Activity: Plan-technical-session
agents: p:publicity-chair, a:author, ch:conference-chair
m:pg-committee-member, c:technical-pg-chair
tu: tutorial-chair, sp:invited-speakers
c can send:

request(authorize(committee-members)) to ch
request(print(call-for-papers)) to p
confirm(received-paper) to a
count(received-paper) to self
request(review(received-paper)) to m
organize(committee-meeting) to self
request(attend(committee-meeting)) to m
request(authorize(list-of-accepted-papers))) to m
confirm(accepted-paper) to a
finalize(logical-program) to self
declare(logical-program) to p
request(present(keynote-speech))) to sp
confirm(list-of-tutorial-speakers) to tu

m can send:
confirm(received-paper) to c
assert(reviews-for-papers) to c
confirm(list-of-accepted-papers) to c

tu can send:
propose(tutorial-speakers) to c

a can send:
assert(send(copies-of-paper)) to c
request(paper-status) to c
request(withdraw(copies-of-paper)) to c

p can send:
confirm(print(call-for-papers)) to c

sp can send:
confirm(agree-to-be(keynote-speaker)) to c

ch can send:
confirm(committee-members) to c

Constraints
˜send(organize(call-for-papers)) UNTIL
organize(committee-members)
˜finalize(logical-program) UNTIL
(confirm(keynote-speakers)
AND confirm(tutorial-speakers))
˜send(print(call-for-papers)) UNTIL
(confirm(approved(committee-members))

, **Goal =** declare(logical-program)

Exit = count(received-papers) $<$ Threshold

End Activity

Activity: Plan-publicity

agents: p:publicity-chair, a:potential-authors,
ch:conference-chair, ph:publication-chair,
b:potential-bus-attendance

p can send:
send(call-for-papers) to a
setup(conference-web-site)
design((conference-poster)
request(approve(conference-poster)) to ch
send(conference-posters) to a
send(call-for-participation) to a
send(call-for-exhibition) to b
request(print(advanced-pgm)) to ph

request(print(day-pgm)) to ph
setup(conference-signage) to self

ch can send:

confirm(approve(conference-poster)) to p

Constraints:

~send(call-for-papers) AND
~setup(conference-web-site) AND
~design(conference-posters) UNTIL
confirm(committee-members)
~confirm(approved(conference-poster)) UNTIL
request(approve(conference-poster))
~request(approve(conference-poster)) UNTIL
design(conference-poster)
~send(call-for-participation) AND
~send(call-for-exhibition) UNTIL
send(call-for-papers)
~(request(print(advanced-pgm)) AND
~(request(print(day-pgm)) AND

Goal = TRUE

Exit = declare(cancel(conference))

End Activity

Activity: Plan-publication

agents: t:treasurer, ph:printing-house
pc:pg-chair, r:registration, p:publication-chair

p can send:

request(camera-ready-copy) to a
request(quotation-for-printing) to ph
request(print(proceedings)) to ph
request(pay(printing-cost)) to t
confirm(payment) to ph

ph can send:

confirm(quotation-for-printing) to p
print(proceedings)
confirm(print(proceedings)) to p
assert(refuse-to(print(proceedings),
"schedule-conflict")
assert(deliver(proceedings)) to p
request(payment) to p

t can send:

confirm(pay(printing-cost)) to ph

a can send

confirm(camera-ready-copy) to p

pc can send:

assert(send(reminder)) to a

Constraints:

~print(proceedings) UNTIL
confirm(camera-ready-copy)
ALWAYS (request(camera-ready-copy)) =>
NEXT (SOMETIMES(confirm
(camera-ready-copy))
XOR assert(send(reminder)))

Goal = declare(print(proceedings))

Exit = assert(refuse-to(print(proceeding),

”schedule-conflict”)
End Activity

References

[1] A. Barros and A. t. Hofstede. Formal Semantics of Coordinative Workflow Specifications. Technical Report 420, Department of Computer Science & Electrical Engineering, University of Queensland, Brisbane, Australia, Dec. 1997.

[2] A. Biliris, S. Dar, N. Gehani, H. Jagadish, and K. Ramamritham. Asset: A system for supporting extended transactions. In *Procs. ACM SIGMOD international Conference on Management of Data*, pages 44–54, Minneapolis, Minnesota, 1994.

[3] A. Bonner and M. Kifer. Concurrency and communication in transaction logic. In *Joint International Conference and Symposium on Logic Programming Languages*. MIT Press, 1996.

[4] A. Buchmann, T. Ozsu, M. Hornick, D. Georgakopoulos, and F. Manola. A transaction model for active distributed object systems. In A. K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, pages 123–158, San Mateo, California, 1992.

[5] A. Elmagarmid, Y. Leu, W. Litwin, and M. Rusinkiewicz. A multidatabase transaction model for interbase. In *Proceedings of the 16th VLDB Conference*, August 1990.

[6] H. Garcia-Molina, D. Gawlick, J. Klein, K. Kleissner, and K. Salem. Coordinating multiple transaction activities. Technical Report CS-TR-247-90, Technical Report, Princeton University, Dept of Computer Science, Feb 1990.

[7] D. Georgakopulos and et al. Specification and management of extended transactions in a programmable transaction environment. In *Proceedings of the 10th Data Engineering Conference*, Houston, USA, February 1994.

[8] M. Haghjoo, M. Papazoglou, and H. Schmit. A semantics-based nested transaction model for intelligent and cooperative information systems. In *First International Conference on Intelligent and Cooperative Information Systems*, Roterdam, Netherland, May 1993.

[9] D. Kieronska. A system for the synthesis of concurrent programs: an algorithmic approach to state graph constructions and transformations. Technical report, PhD thesis, Department of Computer Science, University of Western Australia, 1991.

[10] S. Kimbrough and S. Moore. On automated message processing in electronic commerce and work support systems: Speech act theory and expressive felicity. *ACM Transactions on Informarion Systems*, 15(4):321–367, October 1997.

[11] R. King and D. Mcleod. A database design methodology and tool for information systems. *ACM Transactions on Office Information Systems*, 3(1):2–21, January 1985.

[12] H. F. Korth and G. Speegle. Formal aspets of concurrency control in long-duration transaction systems using the nt/pv model. *ACM Transactions on Database Systems*, 19(3):492–535, 1994.

[13] E. Levy, H. Korth, and A. Silberschatz. An optimistic commit protocol for distributed transaction management. In *Procs. ACM SIGMOD international Conference on Management of Data*, May 1991.

[14] L. Liu and R. Meersman. The building blocks for specifying communication behaviour of complex objects: An activity-driven approach. *ACM Transactions on Database Systems*, 21(2):157–207, 1996.

[15] R. Medina-Mora, T. Winograd, Rodrigo, and F. Flores. The action workflow approach to workflow management technology. In *CSCW 92 Proceedings*, pages 281–288, November 1995.

[16] R. V. D. Meyden. The dynamic logic of permission. *Journal of Logic Computation*, 6(3):465–479, 1996.

[17] A. Ngu, R. Meersman, and H. Weigand. Specification and verification of communication constraints for interoperable transactions. *International Journal of Intelligent and Cooperative Systems*, 3(1):47–65, 1994.

[18] N.Krishnakumar and A.Sheth. Managing hetergeneous multi-system tasks to support enterprise-wide operations. *Distributed and Parallel Databases*, 3:1–33, 1995.

[19] A. S. Paul C. Attie, Murindar P. Singh and M. Rusinkiewicz. Specifying and enforcing intertask dependencies. In *Proceedings of the 19th VLDB Conference*, Dublin, Ireland, 1993.

[20] A. Sheth. From contemporary workflow process automation to adaptive and dynamic work activity coordination and collaboration. In *Proceedings of the Workshop on Workflows in Scientific and Engineering Applications*, 1997.

[21] M. P. Singh. Synthesizing distributed constrained events from transactional workflow specifications. In *Proceedings of 12th International Conference on Data Engineering*, New Orleans, Louisiana, February 1996.

[22] B. Vivier, I. Haimowitz, and J. Luciano. Workflow requirements for electronic commerce in a distributed he althcare enterprise. In A. Sheth, editor, *NSF Workshop on Workflow and Process Automat ion in Information Systems*, May 1996.

[23] H. Wachter and A. Reuter. The ConTract Model. In A. K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, pages 220–263, San Mateo, California, 1992.

[24] H. Weigand and A. H. Ngu. Flexible specification of interoperable transactions. *Data and Knowledge Engineering*, 25(5):327–345, 1998.

[25] R. Wieringa, J. Meyer, and H. Weigand. Specifying dynamic and deontic integrity constraints. *Data and Knowledge Engineering*, 2(4):157–191, 1989.

[26] P. Wolper. Specification and synthesis of communicating processes using an extended temporal logic. In *Proceedings of the 9th Annual ACM Symposium on principles of Programming Languages*, pages 20–33, 1981.