INTER-ORGANISATIONAL COLLABORATIONS SUPPORTED BY E-CONTRACTS

Z. Milosevic[1], P.F. Linington[2], S.Gibson[1], S. Kulkarni[1] and J.Cole[1]
[1]Distributed Systems Technology Centre, The University of Queensland, Brisbane, QLD 4072, Australia; [2]University of Kent, Canterbury, Kent, CT2 7NF, UK.

Abstract: This paper presents a model for describing inter-organizational collaborations for e-commerce, e-government and e-business applications. The model, referred to as a community model, takes into account internal organizational rules and business policies as typically stated in business contracts that govern cross-collaborations. The model can support the development of a new generation of contract management systems that provide true inter-organizational collaboration capabilities to all parties involved in contract management. This includes contract monitoring features and dynamic updates to the processes and policies associated with contracts. We present a blueprint architecture for inter-organizational contract management and a contract language based on the community model. This language can be used to specialize this architecture for concrete collaborative structures and business processes.

Key words: Community Model, Contract Specification, Contract Monitoring, Business Contract Language

1. INTRODUCTON

Business contracts are the key governing mechanism for inter-organizational collaborations and they are increasingly taking a central role in e-commerce, e-business and e-government applications. This is driven mostly by business demands for more transparent, cost efficient and accountable processes and for the preservation of corporate knowledge associated with contract-related procedures and artifacts. As a result, there is a need for a new generation of contract management systems that go beyond the intra-enterprise contracting focus as typically supported by today's Enterprise Resource Planning (ERP) systems or even more frequently, by numerous spreadsheets or simple databases that many organizations use to record their contract information. Increasingly, organizations require new contract management capabilities to facilitate collaborative aspects in cross-organizational arrangements – to enable better insight into capabilities, activities and performance of their partners.

This paper presents our generic contract architecture solution for building a new generation of contract management systems. This solution makes use of Web Services to support the cross-organizational nature of collaborations and to integrate contract management services into the overall business processes between organizations. The solution consists of:

- a repository of contracts to provide access to contract related information such as start and end date of contract, the status of contracts, parties involved as well as relationships between contracts;
- a contract monitoring facility that performs checking of the fulfillment of obligations and compliance monitoring;
- a contract notification component that sends various contract notifications to the parties involved in contract management;
- other components and facilities to support contract negotiations, enforcement and also dynamic configurations of the system to reflect new business rules and structures

This architecture can be regarded as a blueprint architecture for contract management. Its full potential can be achieved by having a powerful contract language that is used to configure the architecture for a particular contract arrangement. In the paper we also present our Business Contract Language (BCL) developed to support such configuration. The BCL expresses the semantics of contracts although it can be applied to express many other enterprise policies and collaborative arrangements. Essentially, BCL is a domain specific language developed for the contracting domain and can be used to express concrete models for specific contracting environments. Our approach follows the model-driven development philosophy which is

currently being proposed by the Object Management Group (OMG) Model-Driven Architecture (MDA).

The next section provides the description of the community model that provides a basis for describing cross-organizational collaborations. We then present our architectural model for cross-organizational contract management. This is followed by an overview of the business contract language that we developed to support contract monitoring capabilities and an example of a procurement related contract to illustrate this language. The paper concludes with a list of open issues and future research directions.

2. MODELLING OF INTER-ORGANIZATIONAL COLLABORATIONS

Web Services provide a way to integrate applications running across the Internet and are well suited to support cross-organizational interactions. However, collaborative arrangements require the capability to express the business rules and constraints of each enterprise and the rules/constraints of engagement with other enterprises – which is an abstraction layer above Web Services. These rules, be they organizational structure rules, business process rules or enterprise policies, together constitute an enterprise model for collaboration. With emerging tools that support model-driven development it will be increasingly possible to use such an enterprise model to generate collaborative applications that can run on top of any middleware infrastructure, including Web Services. The power of a model-driven approach derives from the ability to flexibly and efficiently add new business rules or modify existing ones.

In this paper we present one such enterprise model, a community model, which was developed based on the ODP standards[1,2]. The aim of this model is to capture, in an object based way, the organizational structure of the enterprise and the various localized constraints within it. The community is the basic element of specification, and so is the element used to capture common reusable patterns of constraints[3].

A community is a configuration of objects defined to express some common purpose or objective[1]. It is decoupled from the individual objects representing actors and resources in the distributed enterprise by the use of the role concept. A community defines constraints on the behaviour of the roles it declares, and in any instance of the community these various roles are each filled by particular objects. By forcing its member objects to honour the constraints defined for the roles they fill, the community progresses its objectives. A number of separate communities can be defined to capture

different aspects of the community behaviour, so that a particular object might be fulfilling roles in a business process community, a security management community and an auditing community; the result is an enterprise with behaviour satisfying all the different aspects.

The behaviour defined for a community can include, but is not limited to, simple sequences of algorithmic steps. Much of the behaviour specification is concerned with defining the bounds of reasonable behaviour and expressing preferred choices within them. Because of this, many of the constraints are modal in nature, expressing permissions, prohibitions or obligations on the objects filling the roles, rather than giving a single acceptable sequence of actions.

In general, however, the definition of a community in terms of a set of roles allows great flexibility in deciding how the roles are to be filled, leading to considerable flexibility for the reuse of communities to express, for example, common contract elements. However, in some cases a community may also place additional constraints on how a role is to be filled. For example, a separation of duties concern may be expressed by prohibiting a pattern of role-filling in which two particular roles are filled by a single object.

In addition to the construction of business rules by the parallel composition of communities indicated above, there can be hierarchical composition, so that a single role in a high-level community is filled by an object that has resulted from the definition of some smaller-scale community. For example, a single role in confirming the correctness of a tender in some bidding process might, in detail, be filled by a community formed by a quality assurance team.

Another structuring technique in the modeling of inter-organizational processes is the definition of policies. The main idea here is to acknowledge the fact that the structures being defined are organic and evolving, and to distinguish between parts of the specification that are essential to the process being described, and so cannot be varied without effectively starting over again, and those parts that can be expected to vary, either by local choice or by a foreseen process of renegotiation. These circumscribed areas of variability are the policies associated with the enterprise communities. In an e-contracting environment, policies can be a very powerful tool for tailoring general contract behaviour to the specific circumstances in which the contract instance is to operate. A policy can be defined, for example, to indicate how the progress from stage to stage is to be signaled, or how various kinds of foreseeable violations, such as late payment, are to be acted upon.

Policies can also be defined to control the extent to which the structure of the contract can be allowed to evolve with time, indicating, for example,

whether the way objects fill roles can be updated, or even whether the number of instances of some general kind of role can be increased or decreased to accommodate changing levels of interest, and if so whether there is a specific limit to ensure a sensible quorum for the activity.

The community specifications discussed here are templates, in the ODP sense, in that they are generally parameterised, and that they are used to create community instances by applying a set of instantiation rules derived from the context of the creation action; the term template is used in this paper to highlight the distinction from the more neutral term model.

A more detailed description of community model is described in our earlier paper[3] and also in our recent publication[4].

3. BLUEPRINT CONTRACT ARCHITECTURE

3.1 Extended Enterprise: role of contracts

Inter-organizational collaborations in the extended enterprise increasingly require tighter electronic links between organizations while preserving their individual processes and practices as an element of their competitiveness. This means that organizations are to be involved in cross-organisational business processes but the nature of such processes is different from the nature of internal business processes.

In the cross-organisational space the emphasis is on coordinating message exchanges sent between organizations that typically carry business documents, as shown in Figure 1. Messages can be created as a result of various events, such as actions of objects filling roles, deadlines events or arrival of other messages. Here, there is no centralized engine that coordinates message transfer – rather every organization implements its own decision logic about how to process incoming messages, what internal activity is to be carried out and where and when to send outgoing messages. There are several standardization activities that are attempting to define how Web Services can be used in the cross-organisational business process context such as BPEL[5]. We note that the focus of internal processes is primarily on the control flow and data flow between tasks in a business process.

Contracts are the key mechanisms to govern cross-organisational collaboration. From a legal point of view contracts state what obligations, permission, or prohibitions parties have in respect to each other and what actions are to be undertaken in cases of contract violation, either as a result

of a contract breach or due to circumstances in which force majeure is applied.

The legal jargon in contracts can to some extent be mapped onto a number of more formalized modeling concepts which can be used to facilitate integration of the contracts with cross-organisational business processes and other enterprise systems. However, this mapping is a non-trivial problem and in this paper we present our solution for expressing contracts in terms of modeling concepts suitable for supporting automation in cross-organisational collaborations. These modeling concepts are based on the community model introduced in section 2, and can be grouped in three broad categories:

- expression of roles and their relationships as part of a contract; roles can then be included as part of the basic behavior concepts and policies listed below
- expressions of basic behaviour, e.g. a set of actions carried out by the parties filling the roles and being involved in business transactions and various styles of constraints on these actions including temporal constraints;
- expressions of policies such as obligations, permissions and prohibitions as refinement of basic behaviour; both policies and basic behaviour expressions use more primitive behaviour expressions such as states, events and event relationships

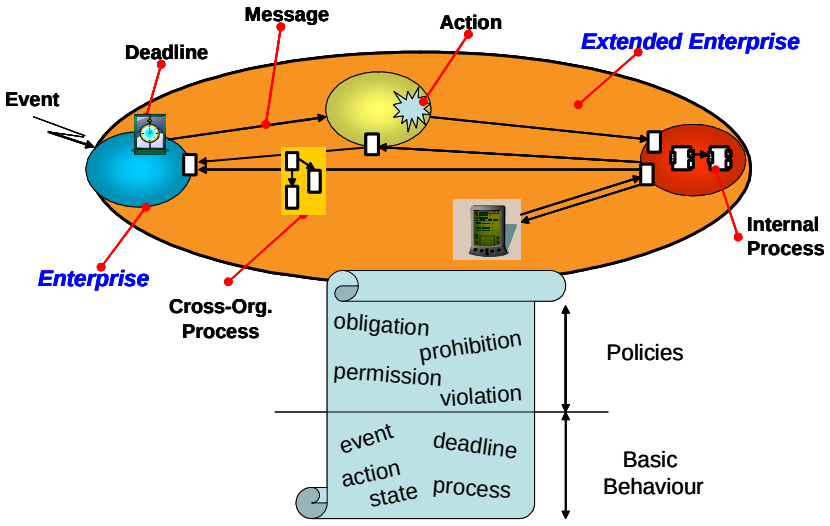

Figure 1. Contracts and cross-organisational interactions

The electronic representation of contract templates can be stored in appropriate repositories and it can be used either for accessing and

navigating information related to a contract or for real-time monitoring of contract execution. The latter includes monitoring of events that are occurring (or not occurring) as part of business transactions carried out in the related enterprise systems, such as e-procurement, payment systems and so on.

3.2 Contract architecture components

To support the full contract life cycle and satisfy the most common contract management procedures we propose a minimum number of architectural components that can be deployed either within one or more collaborative organizations or as a stand-alone system. This Business Contract Architecture (BCA), originally proposed by Milosevic[6], consists of the following core components (see Figure 2).

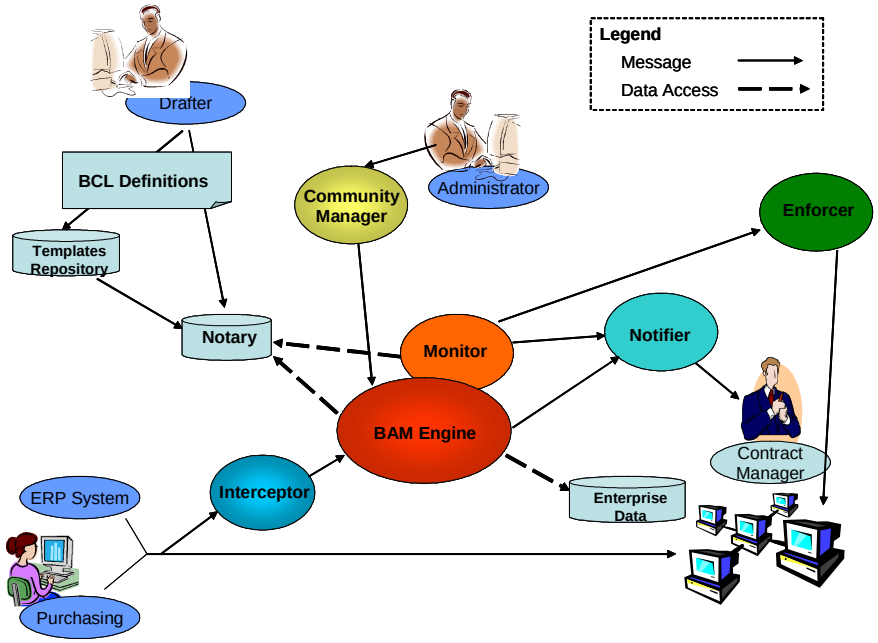

Figure 2. Business Contract Architecture blueprint

- A Contract Repository, which stores standard contract forms (or contract templates), and if necessary standard contract clauses that can be used as building blocks when drafting new contract templates; there are several deployment options for the Contract Repository role – it can be deployed

within either or both of the trading partners or it can be owned by a trusted third-party authority;

- A Notary that stores evidence of agreed contract instances after a contract has been negotiated to prevent any of the parties repudiating it; this component can also store relationships between contracts as necessary;
- An Interceptor, whose purpose is to provide non-intrusive interception of specific messages exchanged between business parties so that they can be further processed for contract monitoring purposes; this is a plug-in component allowing integration with any enterprise system and will vary from one implementation to another, as it implements different message protocols;
- A Business Activity Monitoring (BAM) component, which facilitates the processing of events obtained from the interceptor, managing internal states related to the contract and access to various enterprise data needed for policy evaluation performed by the Contract Monitor component; we note that this component represents an extension of the original BCA in order to enable more powerful event-based monitoring capability;
- A Contract Monitor, that performs the evaluation of contract policies, to determine whether parties' obligations have been satisfied or whether there are violations to the contract; this component makes extensive use of the BAM component for event pattern and state processing; it then sends appropriate messages to the Notifier component mentioned below;
- A Contract Notifier, whose main task is to send notification messages (human readable format) to contract managers such as reminders about the tasks that need to be performed, warnings that some violation event may arise or alarms that a violation has already happened;
- A Contract Enforcer, which can perform some corrective measures such as preventing further transactions if some violation has been detected.

The architecture components above represent core functionality needed for most contract management processes. A contract architecture can also have additional components that can provide further value to the decision makers in the contracting processes such as:

- Contract mediator and arbitrator roles that can be used for discretionary contract enforcement capabilities[7]. The contract mediator essentially collects evidence of parties' behaviour according to the contract. In case of some dispute it can be used as an intermediary to assist the signatories to the contract in determining a future course of corrective actions to ensure contract compliant behaviour. A contract arbitrator can be used in conjunction with a contract mediator as a party that makes decisions about who is at fault (just as judges make their decisions) and whose decisions must be obeyed by a party determined to be at fault. These two

roles are to be used as an alternative or in combination with the non-discretionary enforcement capabilities of a contract enforcer;

- A Contract negotiator, which is a role that facilitates negotiation between contracting parties, possibly as a third party mediator that might have access to business information of relevance for future contracts, and which is not accessible to either of the parties;
- A Contract validitor which can perform a range of activities to ensure that a contract that is being negotiated is valid; this can include checking consistency of contracts[8], or checking the competence aspect of a contract[9];
- A Contract performance repository, that stores various information of relevance to the performance of parties to the contract and that can be used when future contracts are to be negotiated;
- A Contract approval manager, which ensures that only parties with corresponding privileges can execute actions governed by a contract such as role-based or price-based purchase order issuance;
- A Community manager, which allows the contract administrator to make dynamic updates of roles, policies and other community model elements; these updates will need to be checked for their validity and approval by the contract monitor and BAM component.

Our architecture is easily configurable so that additional roles can be added as necessary.

Thus, BCA identifies the main components involved in contract creation, execution and monitoring, but it leaves great flexibility in the way responsibilities can be assigned to organizational units. For example, the trust model associated with the monitor will vary depending on whether there is first, second or third party monitoring. Similarly, the event management infrastructure may be associated with the participants or run by a trusted third party, and this will alter the way that events are analysed.

We note that in the inter-organizational setting these components can be integrated using Web Services technologies. For example, in our prototype the back-end system for Contract Repository and Notary are implemented using IBM Web Sphere platform and the front-end for manipulating and viewing data in the repositories is implemented using Microsoft's ASP.Net technology.

4. BUSINESS CONTRACT LANGUAGE CONCEPTS

The Business Contract Language (BCL) currently under development[4,10] is aimed at describing contract semantics for the purpose of automating contract management activities. Although BCL covers the structural aspects of contracts, describing their composition in terms of contract clauses and sub-clauses, in this paper we concentrate on the part of BCL that is concerned with support for the automation of contract monitoring during contract execution, i.e. after a contract is agreed and the fact stored in the Notary. This automation is aimed at supporting various contract management roles during a contract's lifetime in their activities and decision-making.

BCL is a domain language specifically developed to express contract conditions needed for contract monitoring and to some extent contract enforcement. BCL is a largely declarative language with a minimum number of imperative fragments. BCL interpreter is embedded as part of the BAM and contract monitor components of which implementation details are beyond the scope of this paper.

The BCL language concepts can be grouped in three categories as described next and shown in the figure below:

Figure 3. Business contract language modeling concepts

4.1 Community and Policies

BCL concepts related to communities and policies define organizational, basic behavioural and modal constraints that apply to inter-organisational interactions. Of all of the BCL concepts they are closest to the domain of contracting as they resemble natural language terms and expressions used in contracts.

Organizational constraints can be expressed using a community model that specifies the roles involved in a contract and their relationships,

including hierarchical relationships (through the notion of a nested community or sub-community). The roles can represent organizations as part of their collaboration governed by a larger community, viz contract, or structures within organizations so that it is possible to model internal relationships as well. In order to support the notion of a contract template as a basis for the creation of the corresponding contract instances we introduce the concept of a community template and instantiation rules that specify condition for the creation of contract, as explained in the example below.

Basic behavioural interactions between roles in a contract express the ordering of their actions or steps in a business process carried out by the signatories in a contract. In BCL most basic behaviour constraints are expressed using event patterns as described in section 4.2. Similarly, policies apply to the roles involved specifying refinement of their behavior, in particular modal constraints such as obligations, rights, permissions, prohibitions, accountability, authorizations and so on. As with basic behaviour, policy conditions can be expressed in terms of event patterns.

The main purpose of this group of concepts is to define collaborative arrangements between parties. We note that, although community and policy aspects of the BCL language are developed for the contracting domain, they also have wider generality such as for example the description of internal policies within organizations.

As with other aspects of BCL, these language descriptions are stored in the Notary and will be used by the Contract Monitor and BAM engine to initiate contract monitoring activities.

4.2 Events and States

BCL concepts covering the definitions of Events and internal States are used to describe detailed behaviour constraints that are used as part of community and policy descriptions in the community model. These are fundamental behaviour concepts that can be used for most Business Activity Monitoring (BAM) applications, and are not related only to business contracts. This group includes concepts for the expression of:

- event patterns which are to be used to detect certain occurrences related to the contract either as a single event or as multiple events related to each other;
- internal states and their changes in response to the events;
- event types to be created when certain conditions have been matched, e.g. creation of contract violation or contract fulfilment events

The purpose of BCL's event and state concepts is to support real-time evaluation of the execution of basic behaviour and policies as stated in the contract with the aim of detecting contract violations or contract fulfillments.

In terms of states, this evaluation can, for example, consist of checking whether a certain internal state related to a contract has been reached, such as detecting whether the total number of cost-free withdrawals per month has reached its maximum.

In terms of events, the evaluation can also involve checking whether one or several events have occurred. In BCL an event represents an occurrence of a certain type. An event can be atomic or it can have a duration. In the case of multiple events the BCL provides a rich set of options for expressing relationships between events, namely event patterns. BCL provides a rich set of event pattern expressions and their full description is beyond the scope of this paper. We provide here some examples of event pattern expressions:

- Sequence of events - the event pattern is satisfied when all the events have occurred in the order specified in the sequence
- Disjunction of events - the event pattern is satisfied when either of the events have occurred
- Conjunction of Events - this pattern is satisfied when all the events have occurred
- Quorum – this pattern is satisfied when a specified number from the set of all events have occurred
- Event Causality - the event pattern is satisfied when the currently matched event is causally derived from a specific preceding event.

A special kind of event pattern is introduced to allow for the detection of certain conditions that need to be determined during some 'sliding' period of time. This event pattern is called a sliding Time Window event pattern. The time window is defined by the window's width, the specific condition that needs to be checked within that window (e.g. maximum number of PO requests issued per day), the expressions stating what to do when a condition is found or is not found, and if, appropriate, how to move the window forward.

The event pattern mechanism in BCL has many similarities to the specification of complex event processing[13]. Most of the event pattern language concepts are implemented as part of the BAM component. This component uses event subscription mechanism to listen for the events generated either by external system (through the Interceptor component) or internally from within BCA (e.g. timeout events). Some of the events would require further processing such as the evaluation of policies by the Monitor or creation of new events by an Event Condition Action mechanism. The flexibility of our design and implementation comes from the fact that the interceptor can subscribe to any events such as the events generated by

sending and receiving of messages in the cross-organizational settings, either initiated by machines or by humans.

4.3 General language concepts

While the Communities, Policies and BAM aspects of BCL are used to express key concepts of the contracting domain we needed additional language constructs familiar in most programming languages to support assignment of mathematical or logical expressions to variables, control of loops, conditional constructs, and so on.

5. EXAMPLE: E-PROCURMENT SCENARIO

Consider a simple e-procurement scenario that focuses on a process around the issue of a purchase order (PO) and dispatch of the requested goods. A community template is defined to describe this cross-organisational behaviour involving purchaser and supplier roles, and this may be specified in an umbrella contract.

The contract clauses outline the following behaviour fragments:

- Purchaser is obliged to issue the PurchaseOrder whose integrity must be correct with regard to quantities and pricing.
- Once a PurchaseOrder is received then the goods must be dispatched within some number of days of receiving the purchase order.
- Payment must then follow within so many days of the goods being dispatched.
- If the total of the purchase order is above some threshold then the goods must also be insured.
- Once a cumulative total of purchase orders is reached some discount may then be applied.

This example has been kept simple for reasons of brevity. Realistically it should be extended to handle other likely possibilities such as partial payment and delivery, shipping problems and a plethora of other atypical but possible events and scenarios.

We first introduce a contract template that corresponds to this e-procurement umbrella contract. Since we have defined only a template then the actual values must be defined during some negotiation phase to create a contract instance. These values will include the roles involved, durations for dispatch and payment and thresholds for insurance and discounts. We provide a community instantiation rule that specifies the event which will trigger creation of a community instance. Note that we also define an

activation rule to specify a condition after which this contract (i.e. community instance) may start to be monitored say for the purpose of checking whether the above policies are satisfied.

This example also involves the definition of a nested sub-community for each purchase order (PO) in order to handle monitoring for each individual PO instance separately. Note that the example also shows our policy expressions which follow the spirit of deontic constraints and that some policies are defined in the context of a main community and others as part of a sub-community. We also show how the internal states to the contract are expressed and updated in response to events. This example expressed in pseudo BCL syntax is included below.

```
CommunityTemplate: E-Procurement

 InitialisationSpecification:
                CreateE-ProcurementContractEvent

 ActivationSpecification: StartDate

 Role:  Purchaser
 Role:  Supplier
 Value: StartDate
 Value: DespatchThreshold
 Value: PaymentThreshold
 Value: InsuranceThreshold
 Value: DiscountThreshold
 Value: PurchaseOrderCumulativeTotal

 Policy: POverification
    Role:     Purchaser
    Modality: Obligation
    Condition: On POEvent verify content

 State: CumulativePoTotal
    InitialisationSpecification: 0
    CalculationExpression:
            POCumulativeTotal += POEvent.total

 --- Purchase order sub-community defined below -

 CommunityTemplate: PO
    InitialisationSpecification: POEvent
```

```
ActivationSpecification: OnInitialisation
  EventPattern: GoodsDespatchDeadlineEvent
    GenerateOn:
             POEvent + DespatchThreshold DAYS

  Eventpattern: PaymentDeadlineEvent
    GenerateOn: GoodsDespatchEvent
                   + PaymentThreshold DAYS

Policy: GoodsDespatchWithinThresholdPeriod
  Role: Supplier
  Modality: Obligation
  Condition: GoodsDespatchEvent
        BEFORE GoodsDespatchDeadlineEvent

Policy: PaymentMadeWithinThresholdPeriod
 Role: Purchaser
 Modality: Obligation
 Condition: PaymentEvent
            BEFORE PaymentDeadlineEvent

Policy: GoodsInsuredOverValueThreshold
 Role: Supplier
 Modality: Obligation
 Condition:
   If PurchaseOrderEvent.total GREATERTHAN
      InsuranceThreshold
   Then Action (Insure Goods)

Policy: ApplyDiscountOverCumulativeTotal
 Role: Supplier
 Modality: Obligation
 Condition:
  IfPurchaseOrderCumulativeTotal GREATERTHAN
       DiscountThreshold
  Then
       Action (Apply discount to goods)
```

Note that this example only shows a small set of key BCL concepts and that a more detailed description of BCL features is presented elsewhere[4].

6. CONCLUSIONS AND FUTURE WORK

In this paper we have presented our solution to the problem of integrating contracts as part of cross-organizational collaborations. The solution consists of a generic architecture based on our earlier work[6], which can be tailored to specific contract situation by using Business Contract Language developed for contract domain. This architecture and this language used together facilitate fast deployment of enterprise contract management systems to fit specific organizational requirements. These systems are needed to support important collaborative processes as part of broader inter-organizational arrangements. In particular they support more effective and efficient activities of people responsible for contract management activities.

Our work on BCL adopts a similar approach to the early work of Lee [11] on electronic representation of contracts. Lee proposed a logic model for contracting by considering their temporal, deontic and performative aspects. BCL is developed from a different angle – the enterprise modeling considerations related to open distributed systems. Our approach, based on the ODP community concept [1,2] and inspired by deontic formalisms, gives prominence to the problem of defining enterprise policies as part of organizational structures. Further, we treat contracts as a group of related policies that regulate inter-organizational business activities and processes. In this respect we take a similar approach to that of van den Heuvel and Weigand[12], who developed a business contract specification language to link specifications of workflow systems.

In addition, we consider contracts as the main coordination mechanism for the extended enterprise and, considering possible non-compliance situations, we provide architectural solutions to the problem of monitoring the behaviour stipulated by a contract as firstly proposed in the BCA solution[6]. In addition, this monitoring makes use of sophisticated event processing machinery similar to that of Rapide language[13].

In near future we plan to test our solution in a pilot e-business, e-government or e-commerce environment. This would help us determine expressive power of the language and its acceptability by contract domain experts and practitioners. We also plan to explore the use of existing and emerging tools that support model-driven development to minimize the cost of language maintenance. Another alternative is to consider suitability of high level languages to implement BCL constructs. Finally, we expect that some of the BCL ideas can be used as part of OASIS legalXML e-contracts standardization[14].

ACKNOWLEDGEMENTS

The work reported in this paper has been funded in part by the Co-operative Research Centre for Enterprise Distributed Systems Technology (DSTC) through the Australian Federal Government's CRC Programme (Department of Industry, Science & Resources).

This project was supported by the Innovation Access Programme-International Science and Technology, an initiative of the Government's Innovation Statement, Backing Australia's Ability.

REFERENCES

1. ISO/IEC IS 10746-3, Open Distributed Processing Reference Model, Part 3, Architecture, ISO 1995
2. ISO/IEC IS 15414, Open Distributed Processing-Enterprise Language, 2002
3. P.F. Linington, Z. Milosevic and K. Raymond, Policies in Communities: Extending the ODP Enterprise Viewpoint, in Proc. 2nd International Workshop on Enterprise Distributed Object Computing (EDOC'98), San Diego, USA, November 1998.
4. P.F. Linington, Z. Milosevic, J. Cole, S. Gibson, S. Kulkarni, S. Neal, A unified behavioural model and a contract for extended enterprise, Data Knowledge and Engineering Journal, Elsevier Science, to appear.
5. Business Process Execution Language for Web Services, 1.1, May2003, http://www-106.ibm.com/developerworks/library/ws-bpel/
6. Z. Milosevic. Enterprise Aspects of Open Distributed Systems. PhD thesis, Computer Science Dept. The University of Queensland, October 1995
7. Z. Milosevic, A. Josang, T. Dimitrakos, M.A. Patton – Discretionary Enforcement of Electronic Contracts. Proc. EDOC '02. pp(s): 39 -50. IEEE CS 2002
8. Z. Milosevic, G.Dromey, On Expressing and Monitoring Behaviour in Contracts, EDOC2002 Conference, Lausanne, Switzerland
9. Z. Milosevic, D. Arnold, L. O'Connor - Inter-enterprise contract architecture for open distributed systems: Security requirements. Proc. of WET ICE'96 Workshop on Enterprise Security, Stanford, June 1996
10. S. Neal, J. Cole, P. F. Linington, Z. Milosevic, S. Gibson, S. Kulkarni, Identifying requirements for Business Contract Language: a Monitoring Perspective, IEEE EDOC2003 Conference Proceedings, to appear.
11. R. Lee, A Logic Model for Electronic Contracting, Decision Support Systems, 4, 27-44.
12. W-Jan van den Heuvel, H. Weigand, Cross-Organisational Workflow Integration using Contracts, Decision Support Systems, 33(3): p. 247-265
13. D. Luckham, The Power of Events, Addison-Wesley, 2002
14. OASIS LegalXMLTC http://www.oasis-open.org/committees/legalxml-econtracts/charter.php