# On design and implementation of a contract monitoring facility

Z. Milosevic, S. Gibson, P. F. Linington[*], J. Cole, S. Kulkarni.

*Distributed Systems Technology Centre,*
*University of Queensland,*
*Brisbane, QLD 4072, Australia.*
*{zoran, sgibson, colej, sachink}@dstc.edu.au*

[*]*University of Kent,*
*Kent, CT2 7NF, UK.*
*pfl@kent.ac.uk*

## Abstract

*In this paper we present a solution to the problem of designing and implementing a contract monitoring facility as part of a larger cross-organisational contract management architecture. We first identify key technical requirements for such a facility and then present our contract language and architecture that address key aspects of the requirements. The language is based on a precise model for the expression of behaviour and policies in the extended enterprise and it can be used to build models for a particular enterprise contract environment. These models can be executed by a contract engine that supports the contract management life cycle at both the contract establishment and contract execution phases*

***Keywords:*** *Contract Monitoring, Contract Language, Contract Architecture*

## 1. Introduction

Most interactions between business are conducted according to the rules and policies stated in legally binding agreements or contracts. Contracts specify obligations for the signatories to the contract, as well as their permissions and prohibitions, and may state penalties in cases where these policies are violated. They may also state rewards for outstanding performance.

In spite of the importance of contracts as a governance mechanism for business collaborations there is currently inadequate support for using contract information to manage cross-organisational collaborations, including the management of the contracts themselves. We refer to both of these activities as enterprise contract management (ECM). At present, ECM functionality ranges from manual activities, such as regular checks of contracts stored in filing cabinets, via the use of spreadsheets and databases to record information about contracts, possibly including simple notification triggers, to the more complex support as part of Enterprise Resource Planning (ERP) systems.

Even in cases where there is electronic support for ECM, the focus of these systems has been ‘inward’ – on internal enterprise data and processes. However, the requirements of the extended enterprise, which includes collaborative arrangements between a company and its trading partners, increasingly demand a more ‘outward’ perspective on ECM. This would allow more transparency of the data, processes and performance of trading partners in addition to within the organization itself. Further, this needs to be done in a real-time manner and with minimum latency for information about partners’ data and behaviour.

In response to these demands, several vendors have begun offering standalone ECM functionality that increasingly supports the cross-organisational environment. [2][3][4][8]. A common feature to all these products is their aim to support full contract life cycle management. This ranges from collaborative contract drafting and negotiation (mainly exchanging electronic documents), via storage of contracts and milestone-driven notifications, to analytic features.

However, these systems generally follow the database approach typical of most ERP systems and the contracts semantics is implicitly encoded as part of various data and processes. This is perhaps because there is currently a lack of an overall model that expresses semantics of contracts as a governance mechanism for cross-organisational collaboration. In addition, the functionality of these systems is focused primarily on one type of contract and is built around the processes that accompany it, e.g. procurement contracts, service contracts etc. This can be a limitation because organisations typically deal with various types of contract with varying requirements and future ECMs should be able to support multiple kinds of contract simultaneously, as part of one system.

This paper presents our solution to the problem of expressing contract semantics in a way that enables more efficient and flexible ECM in the extended enterprise. This consists of a *contract language* specifically developed for the contracting domain and a *contract engine* that supports common contract management activities. The language, called the Business Contract Language (BCL) has a particular focus on supporting event-based monitoring of business activities associated with contracts. The engine is based on our architecture model, the Business Contract Architecture (BCA), initially proposed in [6]. The BCL is used to express behaviour and constraints of specific contracts and to design the corresponding specific contract models. This, in combination with the contract engine, which interprets these models, provides a basis for our model based implementation approach.. It also enables dynamic updates of the model to reflect new changes in business rules and structures.

The paper begins (in section 2) with a discussion of requirements for contract monitoring in a cross-organisational environment. Section 3 provides an overview of the community model, which is the foundation for BCL. BCL is presented in section 4. Section 5 describes how BCL is used in conjunction with our contract engine, based on an contract architecture model, Business Contracts Architecture (BCA). Section 6 describes related work and section 7 summarises our approach and outlines direction for our future work.

## 2. Contract monitoring requirements

One of the key phases in automated contract management where a precise expression of contract semantics is needed is contract monitoring. This is because contract monitoring requires expression of the required behaviour of the contract signatories. For example, the required, or permitted, sequences of events that the signatories are expected to exhibit in fulfilling their obligations must be stated in the contract. Another example is compliance with regulatory guidelines as for example those related to the recent Sarbanes-Oxley Act [11] rules in the USA. These requirements have implication for both the contract language design and contract architecture components.

***Expressive contract language –*** a contract monitoring language is required to check the past and current behaviour associated with the execution of activities related to the contract and ultimately the behaviour of the signatories to the contracts. If the contract activities are reported using events, then a language of high expressive power is needed to cater for many possible relationships between such events – which we call event patterns. Examples are sequences of events and causal relations between two or more events. The language should allow efficient construction of the models that describe the structure of entities associated with the contract and the processes in which they are involved. Ideally, this language should represent contract related concepts in a form that would allow domain experts to enter contract related data to support their contract management activities. In addition, the language should allow entry of such information while the system is in operation, including the addition of updates to the existing models as needed. Our solution for such a language is presented in section 4 and the way this language is executed by the corresponding contract engine is discussed in section 5.

In addition to the expressive monitoring language, one also needs reliable event generation and reporting mechanisms dealing with factors such as the accuracy of the reported events and impact of the event generation process and organizational threats from it. These issues discussed in detail in [10] and are summarised here.

***Accuracy of event reports*** – one of the challenges in designing reliable monitoring systems arises from the problem of maintaining a consistent view of time in distributed systems. The implication of this is that there may be some variability in the monitoring mechanism, e.g. the ordering of events reported close together in time, or the relative ordering of an event and a timeout for its receipt. Thus, when designing the monitor, one may need to agree an acceptable latitude for timings, taking into account the knowledge of the properties of the infrastructure in use, and of the contract details.

***Performance issues –*** performance bottlenecks can arise in situations when the monitoring is used on a large scale and solutions need to be provided to deal with this, such as the local processing of events to generate higher level events and summary reports of activity.

***Security issues*** – many security issues are associated with the lack of absolute trust in e-contracting and in the design of the monitoring mechanism it is important to consider the trustworthiness of all the parts of the system. This includes the trust of ECM system users, who need to establish confidence in:

- an event reporting mechanism employed by the monitor, which can be addressed by including a proof of authenticity and a guarantee of non-repudiation in the events;

- a party who owns the monitor (possible choices are trusted third party or an agent for one of the participants);
- the components of the infrastructure they use, such as repositories holding contract information;
- the monitoring system to preserve confidentiality of information; an example is a requirement by a service provider for the non-disclosure of their actual performance.

A more detailed discussion about the issues of trust as part of contract management and some initial solutions are presented in [14].

***Integration with other enterprise systems***. One of the requirements of an ECM is its ability to operate in heterogeneous environments and to be relatively easily integrated with other enterprise systems with minimal disruption to the existing systems. One way of doing this is using Web Service technology as an integration mechanism.. In terms of monitoring, it is necessary to provide as least intrusive mechanism as possible for intercepting messages between parties and a possible approach to this is to use an event listener/monitor, which receives notifications of contract related behaviour from the components using a publish/subscribe mechanism. Finally, the use of open standards such as XML, Web Services and OASIS legalXML e-contracts will allow deploying ECM systems on various platforms.

## 3. Community model

Our approach for the expression of contract semantics is based on the community model inspired by ODP standards [7] and further refined in [9] [10].

A community is a configuration of objects defined to express some common purpose or objective [7]. It is introduced to capture the organizational structure of the enterprise and the various constraints within it – which in combination can be used to capture and specify common reusable patterns of constraints.

This organizational structure is described in terms of roles so that the structure is independent of the individual objects representing actors and resources in the extended enterprise. A community defines constraints on the behaviour of these roles, and in any instance of the community these roles are each filled by specific object instances. Note that a community instance may have some roles unfilled – and this is useful when modelling situations when some organisational positions are vacant. In some cases a community may place additional constraints on how a single role is to be filled. For example, a separation of duties requirement may be expressed by prohibiting a pattern of role-filling in which two particular roles are filled by the same object.

Typically, an extended enterprise is modelled as a number of different communities to capture different aspects of its behaviour and one object can fill roles in different communities. For example, one object might be fulfilling roles in a procurement process community, an authorization community and an auditing community.

Community behaviour has two aspects. The first aspect deals with the specification of basic behaviour [9], which. is expressed in terms of a sequence of actions that are always carried out by the parties filling the roles and various styles of constraints on these actions, including temporal constraints. An example is a sequence of actions that characterize this community behavior. The second aspect of community specification is concerned with defining the bounds of reasonable behaviour and with expressing preferred choices within them. This aspect covers modal constraints, such as permissions, prohibitions or obligations on the objects filling the roles, rather than giving a single acceptable sequence of actions.

In general, 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 [15].

In addition to the construction of business rules by the peer-to-peer 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 [15].

Another structuring technique in the modelling of inter-organizational processes is the definition of policies. The main idea here is to acknowledge the fact that the structures being defined are 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 [15].They can be expressed in modal terms, as obligations, permissions, prohibitions, and authorisation. 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 signalled, or how various kinds of foreseeable violations, such as late payment, are to be acted upon.

Policies can also apply 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 [15].

For a more detailed description of the community modelling see [8] [11].

## 4. Contract Language

This section presents the main features of our solution to the monitoring requirements listed in section 2. This solution, the BCL, is developed based on the precise modelling concepts of a community model introduced earlier.

### 4.1 Main characteristics

***Domain specific*** BCL was developed to enable expression of contract semantics – primarily for contract monitoring purposes. In this respect, the BCL is a domain specific language that introduces modelling abstractions which correspond directly to the contract terms used in the contract management domain. It allows the typically unstructured text of contracts, stated in natural language to be re-expressed in a structured form, which is amenable to automated processing (see Fig. 1). BCL is aimed at, what is in [12] referred to as 'ultimate pair programming', which involves a domain expert and expert developer working together,. We note, that the BCL concepts are generic in nature and can be used to express monitoring of any business activity within or across organisations, not only those directly related to contracts. Although the BCL is based on the precise community model and event pattern semantics of [5] we note that there is currently no formal mathematical underpinning to the language and this is an area of future research.

***Declarative***– BCL is primarily a declarative language whose notation allows the expression of contract domain concepts in a manner close to the way domain experts think. This allows the user to express explicitly their intention, the *what* of the problem, while the language processor takes care of the *how*. A BCL language processor embodies the semantics of the language notation. A small subset of BCL is imperative in nature and this subset was developed as a supporting mechanism to the declarative aspects of the language.

***Event-driven –*** most of the BCL execution is triggered by events. For example, states are update in response to events, policy checking is triggered by events and generation of internal events is driven by other events. The event-driven characteristic of the language is an important contributor to the declarative nature of the language. This approach again facilitates the expression of what should be done in response to some occurrence and the engine will take care of detecting the occurrence and triggering the execution.

***Model-based*** - we follow a model-based philosophy to ensure rapid and predictable development and deployment for specific contracting environments. This entails the use of:

- *models* to describe rules, structures and constraints of a particular contracting environment by using BCL modelling constructs; the models are used to parameterize the contract framework described below
- an e-contract *framework*, which is a body of code that implements the aspects that are common across the entire contracting domain; this framework consists of *i)* a pre-defined contract engine, which implements the semantics of BCL processing and *ii)* other components defined in BCA; the role of the BCL models is to facilitate instantiation of specific contracting scenarios using the generic contracting functionality provided by the framework. Currently we use the J2EE platform to implement our framework.
- templates to represent *patterns* of structure and behaviour such as community modelling concepts described in section 3.

BCL configuration models are used to parameterise the framework, producing a specific contract management system. This model-interpreter paradigm can be considered as one specific style of model-driven development.

### 4.2 BCL modelling concepts

BCL language concepts can be grouped in three categories, described in the order of higher level abstractions to lower level of abstraction, .

#### 4.2.1. Community and Policies

A set of BCL concepts that includes the definition of communities and policies is introduced to define organizational, basic behavioural and modal constraints associated with contracts. These concepts constitute the highest level of abstraction in the BCL as they directly map onto the contracting domain - namely onto the terms expressed in natural language expressions of contracts.

*Organizational constraints* can be expressed using a community model that specifies the roles involved in a contract and their relationships, including hierarchical relationships. The roles can represent organizations as part of their collaboration governed by an overarching community, representing the contract, or structures within organizations so that it is possible to model internal organisational 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 conditions for the creation of a contract, as explained in the example in section 4.3.

*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 4.2.2.

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 the community and policy set of BCL concepts is to define collaborative arrangements between parties. We note that, although the community and policy aspects of the BCL were developed for the contracting domain, they also have wider applicability such as for example the description of internal policies within organizations.

In order to support reuse of community definitions, we define a community template which enables automated support for the creation of community instances based on it. Part of this template is the specification of an instantiation rule which contains an event pattern which defining how to create an instance and how to parameterise that instance

As with other aspects of BCL, these language descriptions are stored in the Notary and will be used by the Contract Monitor and Business Activity Monitoring (BAM) engine to initiate contract monitoring activities. See section 5 for the details of these components.

#### 4.2.2. Events and States

BCL concepts of Events and internal States are used to describe detailed behaviour constraints included within basic behaviour and policies associated with the community models. 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 - for detecting specific occurrences related to the contract either as a single event or as multiple events related to each other; events can result from the actions of entities filling community roles, can occur as a result of deadlines, or can be generated within the system
- internal states and their changes in response to the events;
- event types to be created as actions when certain conditions have been matched, e.g. creation of contract violation or contract fulfilment events; this is similar approach as in Event-Condition-Action paradigm [20].

The purpose of BCL's set of event and state related 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 fulfilments.

In terms of states, this evaluation can, for example, consist of checking whether a certain internal state related to a contract has occurred; an example might be detecting whether the total number of cost-free withdrawals per month has reached its maximum. We note that the changes in state occur as a result of the corresponding events or event patterns and that the concept of state covers both the changes in the values of individual variables, such as total number of transactions in this month, or the changes associated with finite state machine transitions.

In terms of event patterns, the evaluation can involve checking whether one or several events have occurred and if so, we say that the event pattern is satisfied. 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 BCL provides a rich set of options for expressing relationships between events, however their full description is beyond the scope of this paper. We provide representative examples of event pattern expressions [15]:

- 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 any 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 has as its causal parent some previously recognised 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 Purchase Order 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, as described in [5].

#### 4.2.3. General language concepts

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

## 4.3 BCL example

As an example to illustrate some of the BCL concepts, consider basic ‘draw-down’ (authorised purchase order) requests against a master agreement. The master agreement defines an agreement between a purchaser and supplier. There is a maximum value of funds available for this contract and the purchaser must ensure that total draw-downs do not exceed the available funds reserved for this agreement. Any purchase order value over a predefined threshold must also be insured. In addition to maintaining these specific contract clauses it would be desirable to be able to monitor other activities to assist in managing the business.

In the following example there are a number of BCL constructs defined to perform the following monitoring activities.

First, Community Template **`Draw-downsMasterAgreement`** defines activities related to the master agreement, with a sub-community template, **`PurchaseOrderTemplate,`** to handle monitoring for each individual purchase order.

The **`Draw-downsMasterAgreement`** template includes the expression of the following states and policies:

- State, **`CumulativeTotalofAllPurchaseOrders.`** to maintain a cumulative total of all purchases drawn down against this agreement.
- Policy verifying that the total reserved funds are not exceeded, **`DrawDownFundsVerification`**
- Notification Creation Rule that generates an email notification stating that the predefined threshold has been exceeded giving adequate forewarning before reaching the maximum.
- State, **`MonthlyPurchaseOrderTotal,`** that maintains a total of purchase orders drawn for each calendar month. A new instance of each state is created at the beginning of each month and each state is finalised and stored for statistical purposes.
- Time Window, **`30DayPurchaseOrderThreshold,`** that will trigger when a threshold number of purchase orders is exceeded for any 30 day period. This may be useful for statistically determining busy periods.

The sub-community template, **`PurchaseOrderTemplate`** contains one policy:

- **`GoodsInsuredOverValueThreshold,`** verifying that any purchase order that exceeds some threshold is insured.

This example expressed in pseudo BCL syntax is included below.

```
CommunityTemplate:
      Draw-downsMasterAgreement id: 12345

   InitialisationSpecification:
      CreateMasterAgreementEvent

   ActivationSpecification: StartDate

   Value: PurchaserCompanyName
   Value: SupplierCompanyName
   Value: StartDate
   Value: EndDate
   Value: MasterAgreementTotalFunds
   Value: InsuranceThreshold
   Value: PerCentOfTotalToNotify
   Value: MaxPurchaseOrdersPer30Days

   Role:  DespatchOfficer
   Role:  SupplierMasterAgreementManager
   Role:  PurchaserMasterAgreementManager
   Role:  PurchaseOrderOfficer

   State: CumulativeTotalofAllPurchaseOrders
      Initialisation: 0
      CalculationExpression:
         UpdateOn: PurchaseOrderEvent
         UpdateSpecification:
            PurchaseOrderCumulativeTotal +=
                   PurchaseOrderEvent.total
      FinaliseOn: EndDate

   Policy: DrawDownFundsVerification
      Role: PurchaserMasterAgreementManager
      Modality: Obliged
      Condition: On PurchaseOrderEvent
         verify
          CumulativeTotalofAllPurchaseOrders
            <  MasterAgreementTotalFunds
```

```
NotificationCreationRule:
   GenerateOn: if
     CumulativeTotalofAllPurchaseOrders
       >( PerCentOfTotalToNotify / 100
            * MasterAgreementTotalFunds)

   NotificationToGenerate:
       Transport: email
       To:
         ContractManager@Company.net
       From: BCASystem@xyz.com
       Subject:
           Master Agreement Notification
       Message:
         "Master Agreement
          CommunityTemplate.id total
          has risen above
          PerCentOfTotalToNotify percent"

State: MonthlyPurchaseOrderTotal
   InitialisationSpecification: 0
   CalculationExpression:
      UpdateOn: PurchaseOrderEvent
      UpdateSpecification:
         MonthlyPurchaseOrderTotal +=
            PurchaseOrderEvent.total
   FinaliseOn: EndOfMonthEvent
   NewInstanceOn: EndOfMonthEvent


TimeWindow: 30DayPurchaseOrderThreshold
   TimePeriodSequence:
      Width: 30 days
      Step: 1 day
   Do:
      FindMatch:
         EventSequence:
           PurchaseOrderEvent
             MaxOccurs:
               MaxPurchaseOrdersPer30Days


CommunityTemplate: PurchaseOrderTemplate

   InitialisationSpecification:
      PurchaseOrderEvent

      ActivationSpecification: IMMEDIATE

   Policy: GoodsInsuredOverValueThreshold
     Role: DespatchOfficer
     Modality: Obligation
     Condition:
         If
          PurchaseOrderEvent.total >
            InsuranceThreshold
         Then Insure goods
```

## 5. Contract Monitoring Architecture

In this section we explain how BCL is executed by an engine which is part of a broader contract management architecture and we provide a brief description of this architecture.

### 5.1 Executing BCL

As Figure 1 shows, the BCL definitions that constitute contract models will follow closely the expression of contract conditions stated in natural language text. For example a statement of obligation will be of the form:

```
 <role> Purchaser
<Modality> obligation
<behaviour> behaviour expression,
```

Here, the last term, `behaviour expression`, is typically an event pattern, e.g. an event sequence, which needs to be satisfied in order for an obligation to be fulfilled. Notice that at this level the BCL definitions will consist mostly of community, policy and basic behaviour expressions. However, considering that both policies and basic behaviour expressions consist of behaviour constraints expressed in turn using event patterns and states, these BCL definitions will also include detailed expressions of event patterns and states. Thus, when defining BCL models the first step is to specify communities and policies, and these expressions will then be refined using event patterns and states, and any other general language constructs as described in section 4.

The semantic model for the execution of these behaviour constraints is realised as part of the Business Activity Monitoring (BAM), which can be distributed, if needed.

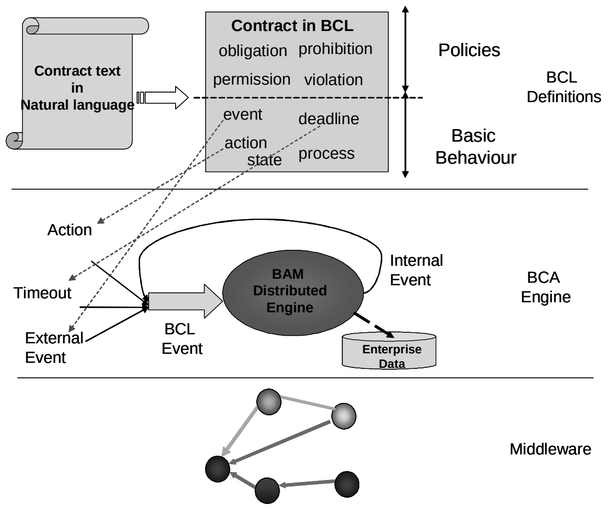

Figure 1: BCL Execution

Once the BCL descriptions are submitted to the BAM engine this engine will respond to events as they occur. As the figure shows, there are different types of events, such as external events resulting from the actions of people or systems, temporal events such as timeouts or internally generated events by the BAM engine. The execution of the BAM does not distinguish the type of these events. Often, as part of a condition evaluation, the BAM engine needs to access data from various enterprise repositories. This monitoring design is quite generic and the BAM engine can be used to monitor execution of any business activity, whether directly related to a legally binding contract, or as part of internal business processes.

Finally, this engine can run on any middleware platform and one obvious choice of value for cross-organisational ECM is the use of Web Services standards.

### 5.2 Overall Business Contract Architecture

The contract monitoring facility is part of a larger ECM system, based on the Business Contract Architecture initially proposed in [6] and further described in [9][14] [15]. In brief, this architecture supports the full contract life cycle and consists of the following roles (Figure 2) [15]:

- A Contract Repository, which stores standard contract templates, and if necessary standard contract clauses as building blocks when drafting new contract templates;
- A Notary that stores evidence of agreed contract instances (and their relationships as needed) after a contract has been negotiated to prevent any of the parties repudiating it;
- An Interceptor, providing non-intrusive interception of business messages exchanged between trading partners for further contract monitoring processing;
- BAM component, that performs the processing of events obtained from the interceptor, management of internal states related to the contract and access to various enterprise data sources needed for policy evaluation performed by the Contract Monitor component;
- 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;
- A Notifier, whose main task is to send human readable notification messages to contract managers. Examples are 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 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 approved by the contract monitor and BAM component.

The architecture components above represent the core functionality needed for most contract management processes. Particular ECM systems may require additional components that can provide further value to the decision makers in the contracting processes. Examples are Contract Enforcer, Contract Mediator and Arbitrator [14] and Contract Validitor [16]. The BCA architecture is easily configurable so that additional roles can be added as necessary.

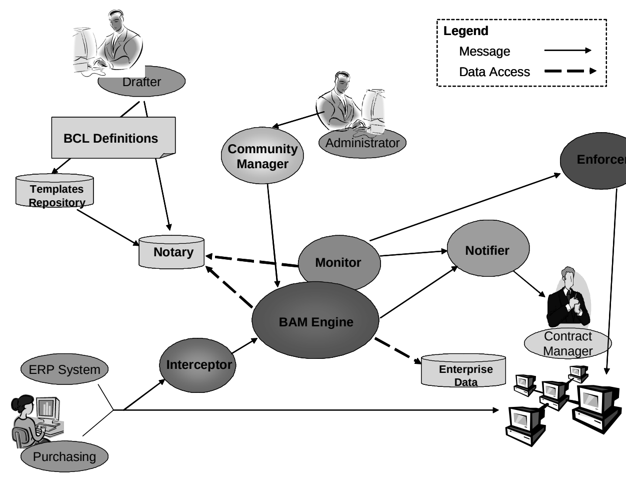

**Figure 2: Business contract architecture**

## 6. Related work

Our work on BCL adopts a similar approach to the early work of Lee on electronic representation of contracts [1]. Lee proposed a logic model for contracting by considering their temporal, deontic and performative aspects. BCL is developed from a different angle – the enterprise modelling considerations related to open distributed systems. Our approach, based on the ODP community concept [7] and inspired by deontic formalisms, gives prominence to the problem of defining enterprise policies as part of organizational structures. 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 [13], who developed a business contract specification language to link specifications of workflow systems. 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. In addition, this monitoring makes use of sophisticated event processing machinery similar to that of Rapide language [5].

Our event-oriented and declarative rule-based language design and the use of XML and Web Service standards have many similarities with BPEL specification [17]. Both approaches express behaviour patterns – a major difference is that we provide more generic expression of behaviour while BPEL concentrates on the business process style of expression.

## 7. Conclusions and Future Work

This paper presented our solution for contract monitoring facility as part of an overall enterprise contract management system. This solution is aimed at dealing with business and legal aspects of contract. It is supported by the BCL language, designed specifically for the contracting domain and together with the BAM engine and other BCA components, the solution is suitable to support cross-organisational ECM.

In the near future we plan to test our solution in a pilot e-business, e-government or e-commerce environment. This would help us confirm the 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-based development to minimize the cost of language maintenance. Another alternative is to consider the suitability of high-level languages to implement BCL constructs. We will also employ emerging Web Services standards and technologies as they get accepted, in particular the BPEL [17] and WSLA [18].

Finally, we expect that some of the BCL ideas can be used as part of OASIS legalXML e-contracts standardization [20] .

## 8. 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.

## 9. References

[1] R. Lee, A Logic Model for Electronic Contracting, *Decision Support Systems*, 4, 27-44.

[2] iMany, www.imany.com

[3] DiCarta, www.dicarta.com

[4] UpsideContracts, www.upsidecontract.com

[5] D. Luckham, *The Power of Events*, Addison-Wesley, 2002

[6] Z. Milosevic. *Enterprise Aspects of Open Distributed Systems*. PhD thesis, Computer Science Dept. The University of Queensland, October 1995.

[7] ISO\IEC IS 15414, Open Distributed Processing-Enterprise Language, 2002.

[8] Oracle Contracts, http://www.oracle.com/appsnet/products/contracts/content.html.

[9] P. 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.

[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, Sep 03.

[11] http://www.sarbanes-oxley.com/

[12] D. Thomas, B. Barry, *Model Driven Development – the case for domain oriented programming*, OOPSLA'03 Companion Proceedings, Oct.2003.

[13] W-Jan van den Heuvel, H. Weigand, *Cross-Organisational Workflow Integration using Contracts*, Decision Support Systems, 33(3): p. 247-265

[14] Z. Milosevic, A. Josang, T. Dimitrakos, M.A. Patton – Discretionary Enforcement of Electronic Contracts. Proc. EDOC '02. pp(s): 39 -50. IEEE CS 2002

[15] Z. Milosevic, P. Linington, J. Cole, S. Gibson, S. Kulkarni, *Inter-organizational collaborations supported by e-contracts*, the IFIP I3E conference, Toulouse, France, 2004.

[16] 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

[17] www-106.ibm.com/developerworks/library/ws-bpel/

[18] www.research.ibm.com/wsla/

[19] www.oasis-open.org/committees/legalxml-econtracts/charter.php

[20] S. Ceri and P. Fraternali, *Designing database Aplicatoins with Objects and Rules*, The IDEA Methodology, Addison-Wesley, 1997