Automating Support for e-Business Contracts

Peter F. Linington.

University of Kent,
Canterbury, Kent, CT2 7NF, UK.
pfl@kent.ac.uk

Abstract

This position paper considers the steps necessary to provide sufficient automation in the support of e-Business contracts for them to become widely used. It focuses on the role of models, taking a model-driven approach to development and discussing the transformational pathways and metamodels needed to support contract-based business processes.

1. Introduction

In the real world, business activities in which organizations cooperate are regulated by contracts – agreements on the patterns of behaviour needed to achieve mutually agreed goals, and of contingencies and sanctions to be applied if the expected behaviour is not performed. These contracts are governed by rules or laws established by the society concerned. It is highly desirable for the ICT infrastructure supporting business activities to be controlled directly by some expression of these contracts, so that correct operation is assured with a minimum of human intervention.

However, each organization has its own agenda and, although the contract represents a mutually acceptable outcome, this is not generally the most advantageous outcome for any of the organizations considered separately; each must have some assurance that the other is keeping their side of the bargain. Reflecting this division of responsibilities, the infrastructure will consist of parts serving each organization and parts operated by third parties; each party will need some way of checking that the others are indeed operating according to contract.

Previous work has proposed an architecture for contract management within the ICT infrastructure [6], for expression of the contract as a set of policies [2] and for a monitoring component that can be used to check adherence to the contract [4]. Work in [3] has proposed a language for expressing such contracts in a form suitable for the checking component to operate on. However, the proof of concept prototypes constructed in the course of this work were hand built and hand configured. A better solution is needed for electronic contracting to be cost effective; one option is Model Driven Development, and that approach is what this position statement discusses.

2. Model Driven Development

The key to the flexible evolution of ICT systems is automation, particularly automation of the production of implementation detail. What needs to be done is to establish an implementation style for elaboration of a high level design so that future modifications to the business design are carried through mechanically, with the minimum of human intervention, into changes to the detailed implementation of the infrastructure. This is the concept behind the model driven development movement.

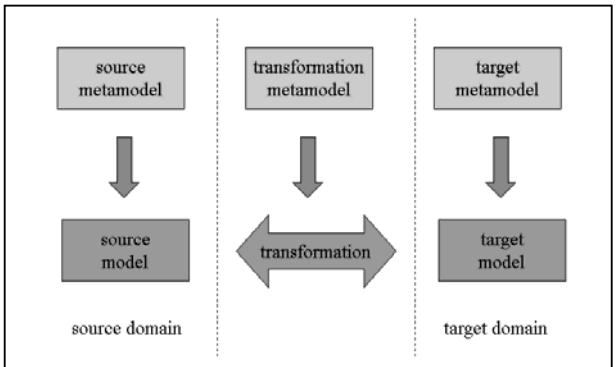

Figure 1: Model Transformation

In this approach (figure 1), the system designers have two kinds of task to perform. Firstly, they have to generate

a design in terms of a model that abstracts away from the details of the supporting infrastructure – a model in business terms; they will create this model using a suitable domain-specific language, or metamodel. Second, they will need to define a transformation from their model to a solution using available infrastructure components. If the business metamodel is stable and reasonably well known, these two tasks can be preformed by different specialists within the team.

In model driven development, it is assumed that the source and target metamodels are stable, so that transformations expressed between them become reusable, and the tools generated to perform them are then themselves long-lived. The transformations themselves are likely to be constructed from reusable components in the form of broadly accepted templates or patterns.

To apply this style to any particular domain, we need to have available suitable target metamodels, and encouragement of reuse dictates that these should have as broad a scope as possible. They are likely to be produced to reflect the properties of the available platform architectures or of general-purpose components. One way of looking at the process is to see the target metamodel or metamodels as defining a virtual machine on which the source behaviour is to be executed.

3. Model Driven Contract Support

How can these ideas be applied to electronic contracts? It is possible to identify a number of different ways in which the representation of a contract will be used (see figure 2). They can be used:

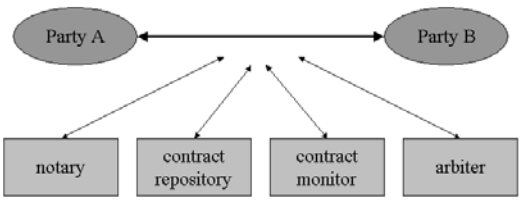

Figure 2 – Use of Contracts

a) to record the results of negotiation between the parties involved, or their agents, thereby creating a contract defining some activity that is to be undertaken; this can be on a one-off basis, or it can result in a contract that will be applied more than once;

b) to steer the performance of activities while carrying out the contract; the contract is used in identifying obligations and in scheduling resultant actions or identifying situations where a response is needed to violations;

c) as a basis for run-time monitoring activities, carried out by components separate from the parties directly involved in the contract;

d) during subsequent arbitration of disputes arising from the contract; this is likely to be based on the audit trail established by the participants, together with non-repudiable statements lodged by them in a mutually agreed trustworthy place.

A single model should be able to represent the contract in each of these cases, but in each of these cases, a specific model and metamodel for contract processing will be needed; the actions to be taken based on interpretation of the contract will differ in each case.

4. Requirements on the Metamodels

4.1 More detailed requirements

To clarify the roles played by different metamodels, let us consider the monitoring process and the environment in which it is carried out in more detail. We assume for simplicity that a contract has been negotiated, and that it has been signed by a Notary and lodged in a trusted contract repository. From there, it is accessed independently by the contracting parties to guide their activities and by the monitor to verify that the actions taken are consistent with the contract.

The contracting parties need to be able to determine at any point which actions are permitted, which are definitely required by some obligation, and for these, how soon action is required and how severe the penalties for failure to comply with the contract are; they also need to identify which actions participants are obliged to report, and to whom. The reporting requirements are likely themselves to form part of the contract, and may imply reporting either to a specific entity or to a well-known channel, to make key events visible to some or all of the participants on an opt-in basis.

The monitor needs to be able to determine whether observed actions are valid at the time where they occur, and what effect they have on the state of, and progression of, the contract. The monitor needs to record enough of the state of each activity to be able to perform this kind of validation.

There may well be a mismatch between the events observed and the actions declared in the contract. This can occur because the contract is expressed in more abstract terms than the actions reported, so that the monitor has to

match patterns representing the more abstract events in order to recognise them. These patterns need not be fixed in the contract, since contracting parties will generally have autonomy in determining how they are to perform contractual actions. Another reason for there being a non-trivial mapping between observation and contractual action is delegation, for example to a sub-contractor, where there would again be flexibility as to how the contractual action is to be achieved in detail.

In any of these cases, a mechanism is needed to support the dynamic binding of detailed behaviour to the contractual actions. What the binding actually is might be determined by pre-registration or by inspecting the parameterisation of initial exchanges in the contract, where details are being negotiated.

4.2 Correlation Requirements

Another area where significant flexibility is needed is in identifying when new instances of the contractual behaviour begin, and which instance of the contract subsequent actions are to be associated with. This is quite similar to the problem of identifying correlation sets in a choreography language like BPEL [8], but with the additional problem that hierarchical interpretation may require several steps in an action binding to be interpreted before the key information for identifying the correlation set is available.

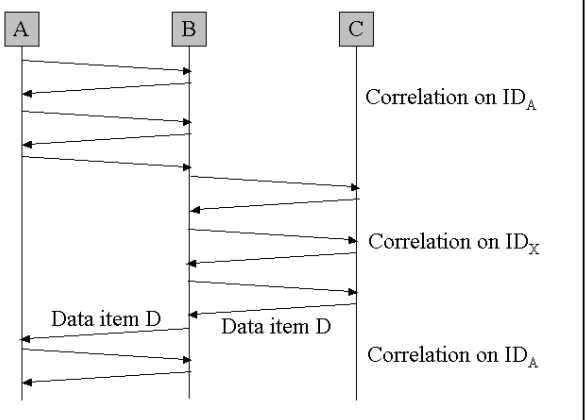

Figure 3 – Late resolution of correlation

In more complex cases, this can lead to a need for the monitor to carry forward a number of possible interpretations, and to prune incorrect guesses when further information becomes available. Consider, for example a contract that includes a sequence of actions involving some part of the infrastructure that, for legacy reasons, does not support the correlation identifier used in the initial activities of the contract instance (see figure 3).

The initial exchanges use as the correlation identifier a value ID_A included in the initial message by party A. However, the legacy exchanges between party B and party C cannot convey this item, and so correlation is based on the value ID_X originated by party B; this is not a problem for party B, which maintains a local mapping between ID_A and ID_X, but this mapping cannot be inferred with any certainty by an external observer, particularly when concurrent instances of the contract are in progress. The observer can correlate the final exchanges between parties A and B with the initial ones, but can only correlate exchanges involving A and C if there is some other suitable data item, such as D, which can safely be used as a correlator. Thus the monitor needs to track multiple possibilities, and may never in fact be able to resolve the situation completely if there is no single, complete chain of correlations.

4.3 Continuous Quantitative Constraints

Some contracts, such as Service Level Agreements, express a mixture of discrete behaviour, in terms of actions, and continuous quantitative measures and activities, and this illustrates another way in which processing decisions may need to be devolved to the monitor. Consider, for example, a supplier of raw material, such as orange juice. This is shipped as concentrate by road tanker to a packaging plant, and the telemetry on the receiving dock reports the flow of juice into the plant. The supply contract requires a lower bound to be placed on the rate of supply of juice, averaged over a three-day period.

The average could be calculated whenever a telemetry message was received, but this could place a considerable burden on the monitor and the notification infrastructure. Considerable savings could be made if the monitor were to operate in a pull-mode, in which it queried the packing plant system about deliveries over a suitably chosen recent period. The problem is then the choice of this period.

If, at some point in time, the monitor updates its historical records of juice flow, it can calculate whether there has been a contract violation. It can also make a worst case assumption that the juice flow might have stopped just after this report and then remained at zero. It is then in a position to calculate the earliest time at which the contract could, in these circumstances, have been violated. The earliest possible violation time can then be used as a deadline for the next reassessment.

Although the details of this example are not likely to be found in many different contracts, the need to assess rolling averages subject to domain specific constraints is likely to be found quite often in a variety of supply contracts and

service level agreements. A monitor should therefore be able to support a framework for this kind of assessment of continuous conditions.

4.4 Contract Language

Previous work by the author in collaboration with Milosevic's team at DSTC has proposed the main features of a contract monitoring language [3]. This language supports the structuring of contract definitions by using the ODP Enterprise concept of communities [1]. A community in ODP is a configuration of collaborating objects, representing entities that is formed to meet some goal, and so the parallel with the structure of participants in a contract is quite clear. In the ODP work the modelling is generally assumed to be object based, and so the contracts are expressed as collaborations of objects, but this is not a serious limitation when considering business contracts, because the parties must be reified at least to the degree necessary to assign obligations and responsibilities to them. Indeed, stating that something is a party can be taken to imply that it is an object.

The idea, then, is to identify nested or overlapping communities as corresponding to contracts, subcontracts or broader applicable bodies of regulations. Community definitions are expressed by declaring a collection of roles and stating the behaviour that these roles are involved in. The roles are the formal parameters of the community, and we can think of the community type as a template that is instantiated by filling the roles with suitable objects. There is then a correspondence between these objects in the representative model and the parties to the contract.

The behaviour of the contract as community will generally consist of some straightforward basic behaviour, representing the expected course of normal execution of the contract, and a set of supporting clauses detailing responses to various exceptions and violations The general shape of the behaviour description is similar to many existing process algebra-based notations, with the ability to express sequence, concurrency as interleaving and guarded choice, determined either by the object or the environment.

The language in [3] supports a flexible sliding window construct to express rolling or periodic constraints. From the point of view of the behavioural specification, this is essentially a special kind of iterator with support that allows the iteration process to be driven by temporal constraints and it supports quite general guards, which can be a mix of temporal guards and conditions over historical behaviour within the window defined. It is, therefore, a generalisation of existing control structures and so integrates quite smoothly with the rest of the behaviour specification.

5. Notification Metamodel

The notification metamodel is quite straightforward, and similar in style to any of the commonly used publish and subscribe messaging services (the JMS model [9] might be taken as typical). The main additional requirement is for a more detailed timing and quality of service model than would perhaps be the norm.

Detailed timing information is needed so that there is enough information for the monitor to reconstruct the sequence of events from different sources. To do this, it needs to be able to correlate source timestamps in the presence of variable transmission delays and lack of synchronization of the various local clocks involved (note, for example, that the JMS model does not name the source clock domain). This is a particular requirement for contract monitoring because manipulation of clocks or introduction of artificial transmission delays can form part of fraud by, or malicious attack on, the parties involved. Considerations of this kind of threat have in the past, for example, led to the banks agreeing to use an independent time signal from a trusted third party to mark the end of the day for clearing purposes.

Even with detailed information about timing, there will still potentially be ambiguity about the actual sequence, and the monitor will need to take this into account, allowing for some margin of error before flagging any violation, and considering the possibility of local reorderings before deciding on the most likely state of the systems observed.

The notification metamodel therefore consists of:

a) a message addressing and routing part, describing source and destination identity, message categorization and associated metadata; there will generally be a need to link this with a broader security model

b) a message specific model dealing with the identity and description of the event being reported and with the timing considerations mentioned above. It is important here for the identity and type information to cover both the identity of the contract applied and identity of the event within the contract, since there will, in general, be a need to track a number of nested or overlapping contracts at any particular time. The model should also describe instance data that can be used for message correlation, although not all of the message transport mechanisms will provide this information, leading to the need for recourse to the kind of content-based correlation discussed above.

6. Monitoring Metamodel

The core of the monitoring virtual machine will be an event pattern recogniser, similar to that described in [6], and so the core of the metamodel will be the grammar for the event pattern language it recognises. There will then be multiple instances of this recogniser, linked to reflect both the structuring into subcontracts and the tracking of concurrent instances of contract execution. There will also be a configuration model to couple these various instances to sources of events and indicate actions to be taken when a match is found; the action may be re-injection of a more abstract event or the generation of a progress signal or violation report from the monitor.

The final element of the metamodel is the language for describing constraints over the history of events, to support the ongoing requirements of, for example, service level agreements. The overview of this structure is shown in figure 4.

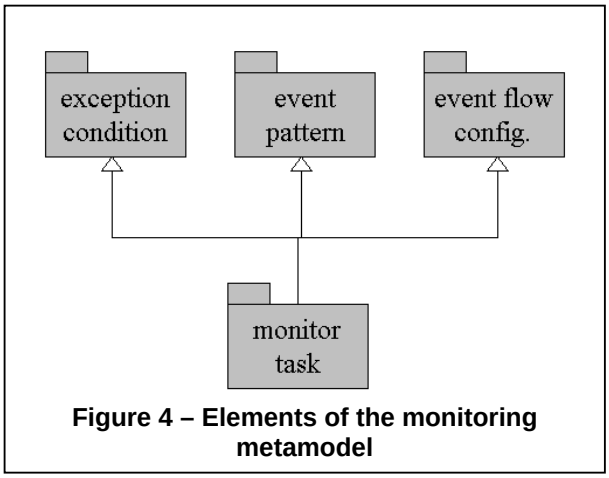

Figure 4 – Elements of the monitoring metamodel

6.1 Event Patterns

The event pattern part of the specification will concentrate on the construction of patterns by the composition of events with behaviour composition operators. A formalization of the BPMN specification [10] would be a good starting point for this (the block structure in BPEL [8] is too restrictive to meet the requirements).

The style intended here would be a recogniser of behaviour expressions in a process-algebra style, similar to CSP [11] or LOTOS [12], in which a behavior is defined as a recursive composition of behaviour fragments, starting with individual actions as primitive pieces of behaviour. The set of operators would include, as a minimum:

a) sequential composition;

b) concurrency by interleaving;

c) guarded deterministic choice, in which the various branches of the choice are determined by the state of the contract, inferred from previous actions and their parameterization;

d) guarded non-deterministic choice, determined eventually by the environment;

e) asynchronous exceptions that override some default behaviour; exceptions of this kind represent a particular problem for monitoring because they are inherently unsafe and subject to race conditions, making timing variations in reporting problematic.

Other variants seen in languages like BPMN, such as compensating actions, are not distinct in the primitive behaviour of the recogniser, but can be constructed.

The internal structure of the monitor is essentially a recogniser for the grammar of a set of token strings derived from the behaviour specification, and most practical cases can be handled by transforming the specification into a state machine. This machine signals recognition of correct behaviour when reaching its final state and it may also be useful for it to recognise key intermediate stages or progress points within the defined behaviour. However, the main function of the recogniser is to signals violations on any event pattern that is inconsistent with the given behaviour. These may be errors of omission or detection of events in the wrong context. Rather than just signalling the fact that there is an error, the behaviour definition will include clauses associating error events with particular predictable departures from the defined pattern.

The basic recogniser will report omissions either by detecting a subsequent event or by time-out. It may seem inconsistent to divide the handling of time into two areas, covering simple timeouts and the more complex service target monitoring described in 4.3 respectively, but in fact they require quite distinct detection strategies and different scopes of observation, making the distinction correspond to different implementation areas.

6.2 Event flow and Configuration

The event flow part would specify how the recogniser inputs are bound to message categories, possibly providing for the specification of name translations to reduce the dependence on application specific details. The actions taken on pattern matches may also imply translations, and may generate notification calls to enforcer components or may generate more human-oriented messages.

In cases where the application reports events with finer granularity than the contract, the hierarchy or recognisers can be extended downwards so as to construct the abstract

events referenced in the contract. Since this may need to be done dynamically based on the observation of negotiation, the monitoring virtual machine must support dynamic binding of recognisers. A dynamic approach also allows the tracking of contracts that depend on short-term sub-contracts or the use of delegation. Similar hierarchical organization can be used to position contracts in appropriate local legal of regulatory frameworks, and this style, in particular, emphasises the need to load contract information from multiple sources and interpret the structure to achieve late binding of names and inheritance of behaviour from separately defined contracts defining local context.

The event flow structure of the recogniser is thus specified in terms of the static and dynamic wiring of a number of primitive pattern recognisers. It expresses the basic structure of the contract into phases and sub-cases, but it also connects exception events to penalty or compensating structures.

Finally, the wiring may express the initiation of actions by boundary components between the automated and non-automated parts of the system, such as the delivery of notifications by e-mail or SMS, or even the generation of solicitor's letters. Conversely, it will also need to handle the injection of events reporting on non-automated processes into the contract monitoring system. This may need to include reports of disruption of the contractual processes, of the infrastructure, or of instances of *Force Majeure*.

6.3 Continuous conditions

Finally, the condition checking part can be expressed by defining constraints on the results of applying assessment functions across defined intervals within the historical record; if such assessment functions can reference the time or age of the element, they can apply any required weightings internally, so a wide range of conditions could be applied by defining a map from the trace items to a result that is accumulated by a small set of built in accumulators such as minimum, maximum or average values. The tactics for steering the polling of continuous values discussed in section 4.3 are subtle and best encapsulated within the virtual machine.

The sliding window mechanism discussed earlier is needed to define the basic timing mechanisms, defining which parts of the contract's history is within the scope of particular constraints, but apart form this the constraints required are expressed in a declarative constraint language relating terms in the contract that can be estimated from the observation of discrete events and continuous quantitative properties of the services being delivered.

6.4 Managing Ambiguity

The virtual machine implementation should also manage the tracking and pruning of alternative interpretations arising from ambiguous event sequences. The implementation described in [6] showed that this can be done efficiently without excessive space costs if alternatives are represented in terms of differences from the state at the point of divergence of interpretation. The implementation maintained a concise single representation of system state for periods sufficiently far in the past for all ambiguities to have been resolved, but generated a record of those parts of the system state affected whenever potential ambiguity was identified by the event pattern recogniser. Thereafter, the different branches were analysed in parallel by the recogniser, with separate state records associated with each branch.

Whenever a branch proved inconsistent, it was pruned, and the intermediate records discarded if no ambiguity remained. The alternatives were also merged if alternatives subsequently converged so that they represented a single state of the system reached via different routes. Duplicating only those parts of the state description where there was divergence has proved to be acceptably efficient in both space and processing usage, and reconciliation can be carried out incrementally without sacrificing the real-time responsiveness of the implementation.

6.5 Combining the pieces

The separation of the contract description into basic behaviour, with a monitor component that matches event patterns and re-injects abstract progress or exception events, configuration of a sequence of such matching recognisers, and constraint monitoring engines operating over a progressive moving windows on the contract's history leads in turn to a modular monitoring implementation.

Thus some quite complex matching mechanisms can be driven from straightforward contract descriptions, with the bulk of the complexity encapsulated within the reusable monitoring components, steered by descriptions produced by transformations of the contract originally negotiated between the parties, and expressed in business terms.

However, the real test of the effectiveness of this approach is to apply it to a larger number of more complex contract examples, and increasing the level of integration will speed the process of investigating different contracts and contract styles. It is to be hoped that wider experience with a range of contractual styles will aid the selection of the basic set of common monitoring features that need to be included within the target metamodel, and will allow

features of limited applicability to be discarded, leading to a tight and efficient reusable core.

7. Conclusions

The main thrust of this position statement is that before a model driven approach to the support of contracts can be successful, we need off the shelf components capable of supporting the monitoring of a large range of contracts, and that the key to reuse of such components is to define a family of metamodels for the event distribution and monitoring functions. If such models exist, they can provide the targets for transformations from the models representing the contracts to the steering information guiding the monitors. This is a general principle, in that application of a model driven approach in other areas will also depend on the creation of a supporting commodity market in components and in the corresponding target metamodels.

These automated transformations, together with the kinds of transformation from business logic to executable processes already given more prominence in model-driven code generation, should make the support of a wide range of different specific contracts tractable at reasonable total cost.

References

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

[2] P. F. Linington, S. Neal, *Using Policies in the Checking of Business-to-Business Contracts*, Policy 2003 Workshop.

[3] P. F. Linington, Z. Milosevic, J. Cole, S. Gibson, S. Kulkarni and S. Neal, *A unified behavioural model and a contract for extended enterprise*, Data Knowledge and Engineering Journal, to be published 2004.

[4] S. Neal, J. Cole, P. F. Linington, Z. Milosevic, S. Gibson and S. Kulkarni, *Identifying requirements for Business Contract Language: a Monitoring Perspective*, in Proc. 7th International Enterprise Distributed Object Computing Conference, Brisbane, Australia, September 2003.

[5] S. Neal, *A Language for the Dynamic Verification of Design Patterns in Distributed Computing*, PhD Thesis, University of Kent, 2001.

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

[7] S. Neal and P. F. Linington., *Tool Support for Development using Patterns*, in Proc. 5th International Enterprise Distributed Object Computing Conference, Seattle, USA, September 2001.

[8] S. Tatte et al., *Business Process Execution Language for Web Service Version 1.1*, BEA Systems, IBM and Microsoft, May 2003.

[9] *Java Message Service 1.1*, Sun Microsystems, April 2002.

[10] S. A. White et al., *Business Process Modelling Notation*, BPMI.org, August 2003.

[11] C. A. R. Hoare, *Communicating Sequential Processes*, Prentice-Hall, 1985.

[12] ISO\IEC IS 8807, *Information processing systems -- Open Systems Interconnection – LOTOS: A formal description technique based on the temporal ordering of observational behaviour*, 1989.