Active Meta Modeling Support for Evolving E-contracts

P. Radha Krishna[*] and Kamalakar Karlapalem[+]

*Institute for Development and Research in Banking Technology, Hyderabad, India.

prkrishna@idrbt.ac.in

[+]International Institute of Information Technology, Hyderabad, India.

kamal@iiit.ac.in

Abstract

An e-contract is a contract modeled, specified, executed, controlled and monitored by a software system. E-contract evolves over a period of time and there are many scenarios of changes in e-contract environment that can adversely affect the execution of e-contracts. Hence, there is a need for evolution operations for e-contracts that can be applied to conceptual model, which can be propagated down to the logical and implementation levels. Since, e-contracts are complex in nature, a more effective way of handling the evolution can be achieved through meta-model. Meta modeling helps in adapting a data model to new requirements. In this paper, we develop an active meta-modeling approach by introducing the taxonomy of evolution operations and handling meta-events in order to facilitate the structural and behavioural conformance of modeling of e-contracts evolution.

1 Introduction

A contract consists of a number of activities and contract clauses, each addressing a specific concern in the business interaction. The enactment of a contract requires contract specification and management, and run-time support for contract environment. An e-contract is a contract modeled, specified, executed, controlled and monitored by a software system. In general, the specification and management of e-contract is handled at conceptual level by a data model, at logical level by a Database Management System (DBMS) and the run-time support for e-contract is handled by both DBMS and Workflow Management System. E-contracts are complex in nature. Our previous experience, focusing on modeling and enactment of e-contracts, indicates that several contracts are similar in structure, for example, clauses related to payment, and are often repeatable. Hence, we employed a meta-modeling approach to support e-contract modeling and enactment. The ER^{EC} model presented by Karlapalem, Dani and Krishna (2001) provides a meta-model for describing the data, relationships and the associated information such as rules and exceptions to model electronic contracts. Details about the ER^{EC} model can be found in the works by Krishna, Karlapalem and Chiu (2004), and Krishna, Karlapalem and Dani (2005).

1.1 Evolving E-contracts

Due to technology advancements, new market requirements and new laws, organizations need to constantly refine their contracts in order to effectively meet the changing constraints and opportunities. E-contract evolution is a basic step towards flexibility in e-contract enactment systems, which requires a consistent and effective monitoring of workflows of parties while they are executing.

There are two major factors that influence the contract evolution namely, the changes in the e-contract design-time environment and the changes in the e-contract run-time environment. *Here, the design-time environment (i.e., mini-world) refers to the e-contract environment that is influenced by factors that affect e-contract enactment (for example, legal, government and social implications).*

During e-contract system design, the requirements are collected from the contract document and the e-contract elements such as parties, activities and clauses are extracted to model an e-contract. Workflows will be identified to execute the activities carried out by different parties. However, changes in design-time and run-time environment during e-contract enactment need modification at conceptual level as well as at logical level.

1.2 Motivation (Scenarios for e-contract evolution)

While an e-contract is being enacted, the parties involved, the clauses, the ECA rules, etc. may change. Further, in the changed scenario, an e-contract may require one or more subcontracts for its enactment, which are not envisaged at the beginning of the e-contract execution. This further requires translation of e-contract instances to deployable workflows. Hence, e-contract deployment necessitates an appropriate model management at multiple levels. Below we discuss three possible scenarios to deal with conceptual modeling for evolving e-contracts.

Scenario 1: A model can be instantiated and necessary modifications can be made on it depending on the revised scenario.

Such an approach is suitable when there is a change in the number of parties involved, the clauses, and the activities. Here, there is no major structural change in the model. Consider an example of a contract between a

borrower and a bank related to housing loans. According to the contract, the monthly repayment for the loan is to be made by the borrower. In case the borrower defaults for more than a specified period, then the bank contacts the guarantor to repay the installments. Here, the role of the guarantor has changed to borrower.

Scenario 2: An e-contract may require one or more additional model elements (for example, subcontracts), which are to be incorporated in the model.
In the event of a change, one can move from an instance of model to an instance of another model, which can further drive the e-contract enactment. Suppose that every loan is associated with insurance in the borrower-bank contract. In the event of death or any disability of the borrower, the original contract will take a different shape in the form of a new sub-contract between the bank and insurance company. This sub-contract may require a different set of elements such as parties, activities and clauses. Further, it may have few sub-sub-contracts to meet the requirements such as police verification and medical certificate. Under the changed circumstances, the model has to be replaced or augmented with an instance of another model.

Scenario 3: The change could evolve the model itself.
By this we mean that one can always use a standard model and instantiate a model when an e-contract is enacted, but as time progresses, to cope up with the changing scenario, evolve the model by adding or modifying the schema concepts. However, this approach is difficult to implement, as it requires structural changes within a model. Consider a situation wherein a house built with the loan amount has to be dismantled due to expansion of a road. In this case, the government has to compensate a prescribed amount to the borrower for the portion of the house that was damaged. Here, not only adding a party (i.e., government) takes place, but also the policies that govern different activities changes. This necessitates generation of new set of models and possibly requires adding new concepts (for example, 'dependent event' concept) to the ER model.

The above scenarios are not exhaustive, but in the real world, more scenarios may arise which leads to more complex cases that determine appropriate evolution of the e-contracts.

1.3 Contributions and organization of the paper

In order to handle e-contracts in a dynamic environment, the ER^{EC} model has to be instantiated by invoking the most appropriate model (if possible) that best meets the requirements of the changes. A template driven two-way active behaviour for supporting the changing applications is presented by Krishna and Karlapalem (2006). In this paper, we have addressed the intricacies involved in supporting evolving e-contracts through an active meta modeling framework. This framework captures the active behaviour (due to run-time and design-time environments changes) of e-contracts during its enactment and models it through meta-ECA rules. The changes required in the meta-model are handled through a set of operations and meta-events that need to be propogated to logical level by specifiying the evolution polices and behaviour for these changes.

The rest of the paper is organized as follows: Section 2 describes our active meta-modeling approach to support structural validation of modeling changing e-contracts. In section 3, we present the mechanism for supporting behavioural validation of modeling changing e-contracts and implementation issues, and section 4 presents related work. Section 5 concludes the paper.

2 Active Meta Modeling for E-contracts

Usually, the parties agree on terms and conditions (clauses) of a contract before starting execution. As mentioned by Iwahihara, Jiang and Kambayashi (2004), a contract process does not describe a concrete work procedure but provides abstract representations of rules and clauses. In the real world, a contract may evolve over a period of time due to changes in the design-time environment. This necessitates design of generic models that abstracts the details of the supporting infrastructure.

Figure 1 shows our approach to deal with active meta-modeling for enactment of evolving e-contracts and is described in subsequent sections. The main engine for supporting execution of e-contract is the context sensitive instance of the meta-model shown as data model in figure 1. The two validations, namely, structural and behavioural ensure that the run time environment is consistent with the design specifications. If the run time environment is not consistent or design-time environment changes - the meta-ECA rules kick-in and select appropriate data model or in the worst case modify the meta model to meet the new requirements.

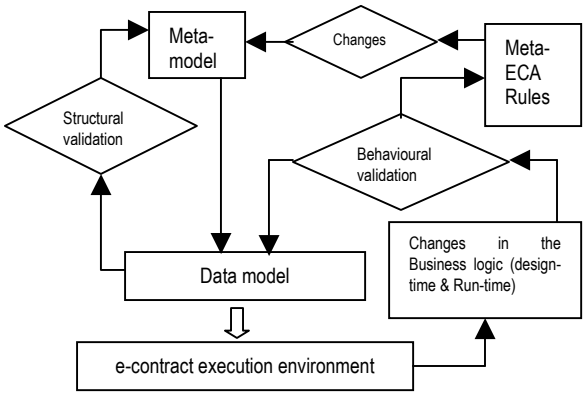

Figure 1: Active meta-modeling of e-contracts

2.1 Support for evolving e-contracts

Deployment of e-contracts poses challenges at conceptual level, logical level and implementation level (like traditional three-schema architecture). Also, the changes in the factors influencing the contract execution require changes at one or more levels. The factors that are to be handled during e-contract evolution includes the following:

At conceptual level:

- Certain clauses are modified during the contract execution of the contract either by the will of the parties involved or by force of law

- Unpredicted events render certain clauses practically or legally vacuous or contradicting clauses
- Exceptional situations arise, for which the contract explicitly defines what should be done. For example, in a housing-loan contract, if the debtor fails to pay, then the guarantor is called to do the payment.

At logical level:

- Changes in the database schema and workflow schema
- Version maintenance of schemas
- Conflicts between different schemas
- Generating/modifying ECA rules when unpredicted events occur
- Handling of exceptions

At implementation level:

- Generating new workflows based on the changes in the clauses and occurring of new events
- Handling of exceptions
- Relating workflow activities dynamically to the contract clauses that are modified during enactment.

In this paper, our focus is on modeling the active behaviour of e-contracts through evolution operations, meta-ECA rules, evolution policies and behaviour validation at conceptual and logical levels. We propose the active meta-model approach which handle/propagate the changes to lower levels. That is, we considered one more level, namely meta-model (or meta-schema), in addition to the three-level schema architecture that is often used in the design of database applications. The conceptual level changes are handled through sequence of operations and meta-ECA rules as described in this section, where as logical level changes are handled through the evolution primitives and behavioural semantics as described in section 4. The system for e-contract evolution possess three properties namely correctness, completeness and consistency.

- Correctness refers to the provision of modifying schema without affecting its semantics. That is, providing this property ensures that the schema is syntactically correct after modification.
- Completeness refers to the possibility of transforming a generic conceptual schema/model into another generic conceptual schema. This property enables the modeling of e-contracts so that it can handle unexpected exceptions. Completeness also implies that unaffected aspects of e-contract are carried forward after.
- Consistency refers to the achievement of modifying schema without causing run-time errors.

The consistency of e-contract schema can be structural consistency and behavioural consistency.

- The structural consistency implies that after any sequence of modifications applied to a schema that is in legal state, the resulting schema remains legal.
- The behavioural consistency implies that application of set of primitives to a legal e-contract schema results in another e-contract schema that is in legal state.

2.2 Taxonomy of operations to affect e-contracts evolution

The taxonomy of operations to evolve e-contracts that takes place in conceptual modeling is presented below:

Adapt: *The model is allowed to adapt based on the new requirements.* It includes cases of run-time changes and exceptions, where the structure of model does not change, but some constructs have to be treated differently because of some exceptional and unforeseen requirements.

Migrate: *The change affects the current model instance and hence a new model has to be instantiated.* A meta-model can be used to generate new model. The new e-contract has to be completed with respect to the original contract.

Merge: *A new model is instantiated and merged with the current model.* This is mostly required when more and more sub-contracts arise in an e-contract enactment.

Build: *The change cannot be handled with current model and also a new model cannot be instantiated.* That is, the meta-model may not generate a model that supports changed scenario. In this case, new concepts have to evolve and refine/augment the meta-model and instantiate a new model. It will consist of generating a new meta-model from existing meta-model and instantiating an instance of the newly generated meta-model. This is required when an unforeseen event occurs and cannot be incorporated with the existing models.

Additive: *The change needs to have a new model while continuing the current model.* This results in running multiple models over a period of time. This is especially required when the changes affect the activities carried out after a specified time while continuing the old processes. The currently executing instance of a model will continue and a new model is enacted for the changed specifications and at a later time some of the concurrently executing models can be terminated as per the termination semantics of the e-contract.

A brief description is provided below for various changes that can occur during contract evolution and how they can be conceptualized. When additions are made to the text in the contract document, it follows the operation *adapt* in which the current instance is augmented with some more entities/elements. Similarly, when certain clauses have been modified or added, new ECA rules have to be generated and accordingly, the model can be modified. The operations for the three scenarios described in the section 1.2 are given below.

For scenario 1, the changes can be made in the model instance itself as per the revised scenario. As there is no structural change, it requires addition or modification of some elements in the model. In scenario 2, additional set of model elements has to be incorporated into the model. This requires *migrate* and/or *merge* operations depending on the change. For example, if a sub-contract has to be incorporated, it requires both migrate and merge operations. In this case, entities and relationships, which have participation in both the instances, are to be modeled appropriately. Scenario 3 needs the operation *build*. The changes in this scenario cannot be modeled directly using the meta-model. In the case of the example described in scenario 3, it requires *adapt* and *build* operations. In the case of adding a new party, the *adapt* operation facilitates augmentation of one more party elements into the model. The *build* operation facilitates

instantiation of a new model (not from the meta-model) with the new construct, which was not envisaged during constructing meta-model. This feature requires that the meta-model itself has to be adaptable or a new meta-model has to be evolved. In the conceptual modeling of evolving e-contracts, one or more operations have to be performed to carry out the changes. In addition to the cases described here, more complex cases can exist for different scenarios in the real world. Moreover, evolving conceptual model based on the operations result in appropriate changes at logical and implementation levels.

During requirement analysis of an e-contract application, specific meta-events have to be identified. Some of the meta-events in the housing loan example described above are: default rate, death/disablement and road expansion. For instance, if the number of non-repayments crosses a specified limit, then the borrower becomes a defaulter. We consider non-repayment by a specified date in a month as an event, whereas, defaulting is a meta-event. Here, a customer can default due to many reasons such as non-repayment, check bounced and customer did some fraud in one or multiple banks. Therefore, we treated defaulting as a meta-event.

Below we explain the adaptability of ER^{EC} model for the examples given for three scenarios described in section 1.2. Here, the meta-events are default rate, death/disablement and road expansion. In each of these cases, a new model is instantiated from the ER^{EC} model repository. When a borrower is granted loan by a bank, the contract is initiated between the said parties which we refer as the "housing-loan" contract.

The contract is enacted as per a standard ER^{EC} model where the parties are, the bank, borrower, guarantor, insurance company. However, at the beginning, the bank and the borrower are only the active parties and the guarantor, and insurance company does not have any active role to play in the contract.

Case 1: **(Run-time change) - Borrower defaults**

As per the contract, the borrower is supposed to repay the installment by a due date. If the borrower fails to do that, then an event raised and the system produces the responsive action in the form of an alert. If the borrower fails to pay for 3 months (condition), then a meta-event "defaulting" occurs. In such a scenario, the bank contacts the guarantor to pay for the balance amount for the loan. This meta-event triggers the following procedure:

```
On event <defaulting>
  begin
    Activate party <guarantor>
    Assign  party <guarantor>  role <borrower>
    Deactivate party <borrower>
  end
```

Case 2: **(Run-time change) – Borrower's death/disablement**

There are two possibilities of this meta-event; (a) the borrower expired in which case the insurance company pays the balance amount and the house is allotted to the nominee; (b) the borrower has disablement due to accident or illness; the insurance company releases a compensation amount. Thus, when this meta-event leads to a subcontract, "insurance-claim" thereby instantiating a new instance of ER^{EC} model. Note that there was no subcontract entity in the original model. This meta-event triggers the following procedure:

```
On event <death-disable>
  begin
    Instantiate model with sub-contract < insurance-claim>
    copy   party <insurance> of contract <housing-loan>   to
           contract <insurance-claim>
    if <death> then
       begin
          add party <nominee>
          assign  party <insurance>  role <borrower>
          deactivate  party < borrower >
          raise event (full payment )
                    /* <payment> by party <insurance>) */
          raise event (allotment)
                    /* <allotment> to party <nominee>) */
       end
     if <disablement>
       raise event (compensation )  /*provide compensation to
          borrower */
  end.
```

Case 3: **(Design-time change) - road expansion**

This meta-event might not have been conceived during requirement analysis. It is a change in the design-time environment, which may happen rarely, however, this change has to be adapted and modeled appropriately. This would require human intervention for creating suitable model by augmenting new concepts and/or modifying existing model elements. The new concepts could be virtual entities – society, human rights, besides adding a party like government. The process is explained as follows:

```
On event <road-expansion>
  begin
     Activate party <government>
     Add virtual-entity <society, human rights…>
     Add virtual-role to virtual-entity <society, human rights…>
              /* estimate compensation amount for the damage */
     raise event (estimate damage)
                    /* provide compensation to house owner */
     raise event (compensation )
  end
```

Here, the virtual-entity and virtual-role are new concepts and have to be modeled appropriately in both meta model and conceptual model.

From the above cases, it is known when a meta-event occurs, the focus of running e-contract shifts, thus triggering selection of appropriate model instance. It is not difficult to perceive specific e-contract domains, a set of ER^{EC} models defined and put in use.

3 Mechanisms to support Active Behaviour of E-contracts

Figure 2 shows the high-level description of ER^{EC} model for e-contract enactment. ER^{EC} model instance for a specific contract can be instantiated from the ER^{EC} meta-model (meta-schema). In this work, the changes occurred during evolution of an e-contract are treated as exceptions. During evolution of an e-contract, the ER^{EC} model is able to incorporate changes and sometimes it may require addition or modification of a concept. The exceptions (or changes) are easily handled by modifying the instance values without affecting the ER^{EC} instance for the same e-contract. However, some of the changes

that adhere to the meta-schema require instance level evolution and some changes require modifications in the meta-schema itself. Thus, there are two kinds of e-contact evolution of ER^{EC} model:

(i) meta-level evolution: in this case, the old (i^{th}) ER^{EC} meta-schema modifies into a new (j^{th}) meta-schema. That is, $ER_i^{EC} \rightarrow ER_j^{EC}$

(ii) instance level evolution: in this case, the old (i^{th}) ER^{EC} instance is modified into a new (j^{th}) ER^{EC} instance. That is, $ER^{EC}{}_i \rightarrow ER^{EC}{}_j$

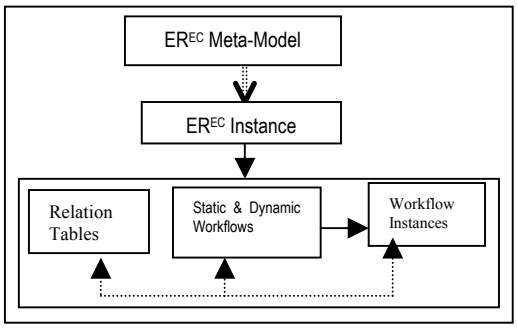

Figure 2: A High Level ER^{EC} Model for e-contract Enactment

We propose a set of primitives that allow generic modification of e-contract schema, while preserving syntactical correctness of e-contract schema. At the same time, the running instances will need to respect the new rules as well. During contract evolution, the contract parameters or contract entities (such as parties, clauses, activities and sub-contracts) may change. These parameters can be added or removed in the e-contract schema. We introduce primitives such as AddParameter, RemoveParameter, AddEntity and RemoveEntity to handle these changes in the schema. Handling of some of the unexpected exceptions may cause changes in semantics of the e-contract schema, that is, addition of new entities and relationships. Thus, the two primitives AppendEntityType and AppendRelationshipType help in supporting such kind of behavioural changes in the schema.

Behavioural relations between roles in a contract express the ordering of their activities carried out by the parties in the contract. Moreover, business rules apply to the roles involved, specifying refinement of their behaviour based on the different clauses specified in a contract. Formally, a contract *C* is described by, $C = \{P, A, CL\}$, where $P = <P_1, P_2, ...P_n>$ is a set of parties, $n >= 2$; A is the set of activities to be performed by different parties; and *CL* is the set of clauses. Specification of *meta-model* for an e-contract will be of the form:

```
<Party>    ::=  PartyID ':' Party_name [PartyID ':' Party_name]*
<Role >    ::=  Role_ID ':' Role_name ':' Activity
<Activity> ::=Activity_ID ':' Activity_name [Activity_name]*
<Clause>  ::=Clause_ID ':' Clause_sentence
<Evolution Policies>   ::= Policy_sentence
<Behaviour>   ::= Behaviour_sentence
```

Here, *behaviour_sentence* typically specifies the active behaviour in the ER^{EC} model that needs to be satisfied in order for a clause or evolution policy is to be fulfilled and is defined as follows:

```
Behaviour_sentence ::= WHEN meta_event FIRE rule'.'
  Meta_event  ::= meta_event_ID ':' meta_event_type
  Rule  ::=  rule_ID ['(' description')'] ':' [IF condition THEN]
                  action_list
```

The *action_list* is a sequence of actions related to contract application, database operations or the operations (or sequence of operations) on meta-model. Here, the execution of actions in one rule may give rise to new meta-events that may fire other rules. A rule is defined as follows:

```
CREATE  RULE  rule_ID  [description]  {BEFORE  |  AFTER}
                  meta_event
EXECUTE PROCEDURE procedure_name
                  (procedure_parameters);
```

Here, *rule_ID* maps directly into *rule_name*, description maps into a string of characters for documentation purpose and event maps into the corresponding language construct. The meta-events could be contract event, internal event (for example, database event) or external event. Each object referred in the *meta_event*, condition or *action_list* part of the behaviour sentence is mapped into a database table. Further, each value referred in the condition or *action_list* is mapped into a value in the corresponding domain. The mapping process translates the behaviour definition into the specification of procedures and rules in the DBMS under consideration. Our idea here is the active capabilities of meta level modeling fill the gap of providing functionalities that exist at the logical level that do not have corresponding modeling constructs at the conceptual level. The mapping process is intended to use by a translation tool that automatically generates the executable DBMS language statements corresponding to the active behaviour specified in the ER^{EC} model. *In this work, we have not considered specification of constraints in the modeling of active behaviour of e-contracts.* As an example, consider the scenario 2 case of *meta_event* 'death'. The following procedure modifies the meta-model of Housing-loan contract.

```
CREATE PROCEDURE Proc_Borrowers_DD (Contract
          Housing_Loan, Contract Insurance_claim, Number
          repayment_balance)
AS DECLARE message VARCHAR NOT NULL;
BEGIN
  IF (repayment_balance > 0) THEN
    BEGIN
      ACTIVATE insurance_claim CONTRACT;
      CREATE RELATIONSHIP_TYPE Linked_To;
      ADD LINK BETWEEN Housing_Loan AND insurance_claim
          USING Linked_To;
      CREATE RELATIONSHIP_TYPE Have;
       ADD LINK BETWEEN Housing_Loan.Parities AND
          Insurance_claim.Parties AND Housing_Loan.Roles WITH
          Have;
       ASSIGN ACTIVITIES of Borrower to Insurance_claim.Parties.
          Insurance_company;
       MARK DELETE Housing_Loan.Parites.Borrower;
       MARK DELETE Housing_Loan.Parties.Insurance_Company;
       RAISE EVENT Re_Pay_from_Insurance_company;
       Message: "All references to BORROWER being deleted
    END
   ELSE Message : 'No Balance for Repayment'
   MESSAGE: "The Account is cleared. Allot house to nominee'
   PERFORM ACTIVITY Allotment_house_to_nominee;
END
```

The above procedure also raises other events and triggers corresponding activities to be carried out. This will enable to update the necessary table in the metadata and initiate a new workflow instances as required by the contract. We are currently architecting the complete

functionality of Meta-Model-Manager to carry out these tasks.

The operations defined in section 2 facilitates structural validation of meta-level and the specifications defined above facilitate the behavioural validation at meta-level. Thus, these operations and specifications at meta-level provides execution driven, consistency driven and flexibility driven for e-contract execution environment. The structural and behavioural consistencies in e-contract modeling are guaranteed for the application of a single primitive or a sequence of primitives, thus validating the modification applied to the e-contract schema.

A high-level architecture for developing e-contract system is described by Krishna and Karlapalem (2006). In this work, we developed mechanisms to carry out the implementation of the system. We note that the evolution operations determine how the meta-model changes. Hence, more precise syntax and semantics have to be made with an appropriate language.

4 Related Work

Considerable work has been carried out on the legal aspects and negotiation stages of e-contracts, but, to the best of our knowledge, e-contract evolution has not been paid sufficient attention by researchers in this field. Lei and Singh (1997) detailed a comparison of various workflow meta-models, but there has been no work on conceptually modeling e-contracts. The CrossFlow project by Grefen et al. (2000) introduces dynamic contracting and configuration for service enactment. Griffel et al (1998) described the COSMOS (Common Open Service Market for SMEs) project, which provides a set of services that facilitate the use of e-contracts on the Internet. Chiu et al. (2002) developed a framework for workflow view based e-Contracts for e-services. Xu and Jeusfeld (2003) proposed a framework for monitoring e-contracts during the contract execution. Angelov, Till and Grefen (2005) proposed architectures to support updates in dynamic, communication-intensive contract enactment environment. These architectures are mainly providing security aspects in a dynamic environment. Iwaihara, Jiang and Kambayashi (2004) presented a Workflow-Contract-Solution model to support execution of e-contract business processes. Linington (2005) presented model-driven approach based on meta-models to support contract-based business processes.

The ER^{EC} model was advocated to conceptualize and enactment of e-contracts in the works by Karlapalem, Dani and Krishna (2001), Krishna, Karlapalem and Chiu (2004) and Krishna, Karlapalem and Dani (2005). In this work, we describe an ER^{EC} model based on active meta-modeling to support enactment of evolving e-contracts.

5 Conclusions

E-contract evolves over a period of time due to the changes in the run-time as well as design-time environments. In this paper, we presented a pragmatic approach to support evolving e-contracts. We illustrated scenarios that determine the need of evolving e-contracts and developed an active meta modeling approach by introducing the taxonomy of evolution operations. Meta-ECA rules are also introduced to model the active behavior and drive the e-contract evolution. The evolution operations and meta-events facilitate the structural and behavioural conformance of e-contracts due to evolution. In this paper, the notion of supporting *evolving e-contracts* is driven by ER^{EC} meta-model.

6 References

Angelov, S., Till, S. and Grefen, P. (2005) Dynamic and Secure B2B E-contract Update Management, *Proc. ACM Conference on Electronic Commerce,* Canada, 19-28.

Griffel, F., Boger, M., Weinreich, H. , Lamersdorf W. and Mertz M. (1998): Electronic Contracting with COSMOS – How to establish, negotiate and execute electronic contracts on the Internet, *Proc. 2nd Int. Enterprise Distributed Object Computing Workshop*, 46-55.

Iwahihara, M., Jiang H. and Kambayashi, Y. (2004): An Integrated Model of workflows, e-Contracts and Solution Implementation, *ACM Symposium on Applied Computing*, 1390 – 1395.

Karlapalem, K., Dani, A. R. and Krishna, P. R. (2001): A Frame Work for Modeling Electronic Contracts, *ER-2001*, LNCS, vol. 2224, 193-207.

Lei, Y. and Singh, M. P. (1997): A comparison of workflow metamodels, Proc. the ER'97 Workshop on Behavioral Models and Design Transformations: Issues and Opportunities in Conceptual Modeling, California, USA.

Krishna P. R., and Karlapalem, K. (2006): Actively Evolving Conceptual Models for Mini-world and Run-time Environment, Proc. the ER'06 Workshop on Active Conceptual Modeling of Learning , Tuscon, AZ, USA.

Xu L. and Jeusfeld, M. A. (2003): Pro-active monitoring of electronic contracts, CAiSE'03.

Linington P. F. (2005): Automatic Support for e-Business Contracts, *Int. J. of Cooperative Information Systems*, **14** (2-3): 77-98.

Chiu, D. K. W., Karlapalem, K., Li, Q. and Kafeza, E. (2002): Eleanna, Workflow View Based E-Contracts in a Cross-Organizational E-Services Environment, *Distributed and Parallel Databases* **12(**2/3): 193-216.

Chiu, D. K. W., Li, Q. and Karlapalem, K. (2001): Web interface driven cooperative exception handling in ADOME workflow management system, *Information Systems,* **26**(2): 93-120.

Grefen, P., Aberer, K., Hoffner Y. and Ludwig, H. (2000): CrossFlow: Cross-Organizational workflow management in Dynamic virtual enterprises, *Int. J. of Computer Systems Science and Engineering*, **15**(5): 277-290.

Krishna, P. R., Karlapalem K. and Chiu, D. K. W. (2004): An ER^{EC} Framework for E-Contract Modeling, Enactment and Monitoring, *Data and Knowledge Engineering*, **51(1):** 31-58.

Krishna, P. R., Karlapalem, K. and Dani, A. R.(2005): From Contracts to E-Contracts: Modeling and Enactment, *Information Technology and Management,***4(**1):363–387.