

Requirements on a B2B E-contract Language

Samuil Angelov, Paul Grefen
Information Systems Group
Faculty of Technology Management
Eindhoven University of Technology
P.O. Box 513, 5600 MB, Eindhoven
The Netherlands

Abstract

Electronic contracts are the instrument to govern electronic trading relationships between business parties. A number of efforts exist in both the academic and industrial worlds to define an e-contract specification language. However, these efforts lack either universality or completeness. The main reasons for this are the complexity and diversity of business contracts. For the definition of a common, complete e-contract specification language, the identification of requirements on an e-contract language is an underlying prerequisite. In this paper, we present the requirements on an e-contract specification language. The identified requirements are based on the investigation of traditional business contracts from different business domains (business pull aspects), the technology aspects of e-contracts (technology push aspects), and the existing research efforts on this topic (research aspects). The results presented in this paper allow definition of e-contract specification languages that satisfy the requirements imposed by both business and technology.

1. Introduction

Business-to-business contracts are an indispensable part of trading business relations since many centuries. As defined by Reinecke [Rei89], “*A contract is a legally enforceable agreement, in which two or more parties commit to certain obligations in return for certain rights*”. With the advent of information technology, companies started using information technologies to support their trading relations. Consequently, in trading relations supported by modern information technology, traditional paper contracts become an inefficient and ineffective instrument to guarantee the rights and specify the obligations of the trading parties and electronic contracts become a necessity.

Electronic contracts can simply be digitized traditional business contracts. However, the benefits for a company from such electronic representation are usually low. We call this type of e-contracting “shallow e-contracting” [Ang04]. Electronic contracting that provides means for a high level of automation of the contract establishment, contract management, and enactment processes presents significantly more opportunities to the trading parties. We call this type of e-contracting “deep e-contracting”. For the implementation of deep e-contracting, many business, technological, and legal challenges have to be addressed. One of the fundamental requirements for the high level of automation of management and enactment of e-contracts is their digital representation in a format that can automatically be interpreted and reasoned about by an information system. A number of research efforts exist that aim at representing business contracts in an automatically interpretable electronic format. However, the requirements specifications on which these languages are based are either highly context dependent (e.g., [Koe00], [Dan98],

[Gri98]) or are of general nature (e.g., [Nea03]). These requirements specifications though not sufficient for the definition of a context independent e-contract language provide a valuable foundation that we use in our work.

In this paper, we identify the requirements on an automatically interpretable e-contract specification language. The identified requirements reflect both business drivers (business pull requirements) and technology drivers (technology push requirements). As a starting point, we examined the requirements identified in other related research projects. In this paper, we extend the existing set of requirements with the additional requirements that we have identified. Throughout the paper, next to each identified requirement, we make clear references to related, existing research findings on requirements on an e-contract language (when such exist). We use the previously defined 4W framework [Ang03] as basis for the requirements identification on an e-contract specification language. The high level of abstraction of the 4W framework allows only general requirements to be identified when using it. For this reason, in our next step, based on the identified set of general requirements, we define a set of more detailed requirements.

To illustrate the source of the identified business driven requirements, we provide two complete sample business contracts and a number of separate contract clauses. When an identified requirement is business driven, we make clear references to clauses in the sample contracts or to clauses from other contracts.

The collected requirements on an e-contract specification language presented in this paper provide a foundation for the definition of a context-independent e-contract specification language. A language based on these requirements will allow automation of e-contract verification, enactment, and management to be achieved.

To prove completeness and usability of the presented requirements, a number of e-contracts based on real-life business cases from different business domains must be defined in an e-contract language that implements the defined requirements. As a first step in this direction, in this paper, we provide a sample e-contract specification language that implements the requirements identified by us and that can be used to define e-contracts from various business domains.

This paper is organized as follows. In Section 2, we briefly explain the 4W framework and provide an example of a business contract that is positioned in the 4W framework. In Section 3, we examine the general technology and business requirements. In Section 4, we elaborate on the general set of requirements and define a set of more concrete requirements. In Section 5, we discuss the existing research efforts on e-contract specification languages. The paper ends with conclusions. A specification of a sample e-contract language is provided in Appendix A. A second business contract used in the paper as an illustration in the definition of business driven requirements is provided in Appendix B.

2. The 4W framework and its application

As a basis for structuring of our work on requirements identification on an e-contract language, we use the previously defined 4W framework. Next, we briefly present the 4W framework and its application on a sample business contract. A detailed description of the 4W framework can be found in [Ang03] and [Ang01].

2.1. The 4W framework

The central concept in the 4W e-contracting framework is the contract concept. At the highest level of abstraction, the contract concept is associated with four basic concepts, i.e., the **Who**, **Where**, **What**, and **HoW** groups (see Figure 1). The **Who** concepts models the *actors* that participate in the contract establishment and enactment. The **Where** concept models the *context* of the contract. The **What** concept models the *exchanged values* and the *exchange processes and rules*. The **HoW** concept relates to the *means* for contract establishment and enactment.

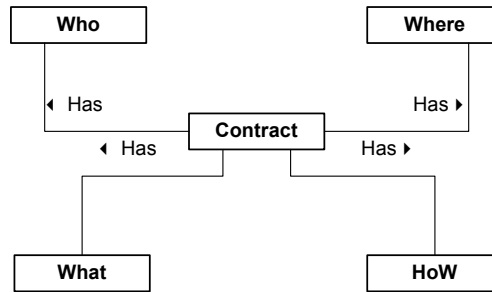


Figure 1 - The 4W e-contracting framework – top level

At the second level of detail, we specialize the four basic concepts as follows:

- The **Who** concept is specialized into the *parties*, *mediators* and *auxiliary implementors* concepts.
- The **Where** concept is specialized into *business*, *legal*, and *geographical*, context.
- The **What** concept is specialized into the *product*, *service*, *financial reward*, *exchange description* concepts.
- The **HoW** concept is specialized into the *contracting phases (information, pre-contracting, contracting, and enactment)*, the *contract representation, content, structure, communications, and standards* concept.

In order not to overburden Figure 1, the concepts from the second level of detail are not shown in the figure.

Further specialization of the concepts at a third level of detail reveals concepts like: *processes, messages, time*. This list is not complete and is provided only with an illustrative purpose.

2.2. Analysis of a traditional business contract

Business-to-business contracts are not bound to observe any legal or business requirements. Each company can use pre-existing contracts, elaborate its own contract templates, or come to a new contract after negotiations with its counterparty. Companies are advised to put in the contract certain clauses, e.g., the law that governs the contract (especially in international contracts), the jurisdiction responsible for handling any contract disputes. However, they are not obliged to do so (according to the freedom of contract [Gis00]). The size of a business contract can vary from 1 page (see example contract below) up to hundreds of pages (the contracts discussed in [Das97] reach 300 pages). This indicates the boundless variety of existing contracts. This variety of contracts is one of the obstacles to the formalization of contracts in automatically interpretable documents.

In our work, we have used sample business contracts from various companies trading in various business domains (e.g., advertising, publishing, and consulting). Next, we present one contract and position it in the 4W framework. The contract has been chosen for its brevity while containing clauses related to each of the four groups of concepts of the 4W framework. For privacy reasons, we have replaced the names of the companies with the names "Company A" and "Company B". We use this sample contract template and the contract presented in Appendix B to illustrate the requirements on an e-contract language set by the business (the business pull requirements).

Example 1:

Contract for HyperText Markup Language Formatting

1 This contract is to establish the conditions under which Company A, hereafter referred
2 to as The Contractor, will provide to Company B, hereafter referred to as The Author,
3 formatting services for electronic files listed below, referred to as The Product.
4

5 The Author will provide to The Contractor all necessary files for The Product in an
6 electronic form by a mutually agreed upon method, such as uploading through the
7 Internet to the Contractor's ftp directory. These file will be complete, free of viruses
8 and errors, to the best of the Author's ability to determine.
9

10 The Product will be in an electronic format mutually agreed upon in advance, and will
11 include a separate file with any special instructions on how to format The Product.
12

13 The Contractor will change the files included in The Product into HyperText Markup
14 Language, compliant with HTML 2.0 protocols, following the Author's instructions.
15 The Contractor will make no other changes of any kind to any of the files.
16

17 The payment for this formatting service will be a discount of (an amount agreed upon
18 in advance) on any sales of The Product by The Contractor for a period of one year.
19 This contract will also constitute an exclusive contract for distribution through the
20 Internet of The Product for one year, and supersedes any non-exclusive provisions of
21 any Distribution License between The Author and The Contractor relating to The
22 Product.
23

24 All services rendered under this contract are "work for hire" by a contractor. Company
25 A is not an employee of The Author, and has no claim to ownership or copyright of
26 any work produced.
27

28 This contract is not valid without a separate Distribution License.
29

30 Files to be included in The Product:
31

32	File Name	File Size	Description
33	_____	_____ bytes	_____
34	_____	_____ bytes	_____

	Buyer's signature	Seller's signature
36	Buyer:	Seller:
37	<i>Name</i>	*****
38	<i>dba (if applicable)</i>	*****
39	<i>address</i>	Anaheim, CA *****
40	<i>city, state, zip</i>	<i>Phone Number</i> *****
41	<i>email address:</i> _____	<i>email address:</i> *****

Next, we position the contract clauses in the 4W framework. The purpose of this is twofold. First, we show the application of the 4W framework for structuring the content of contracts (used in Section 3.2.3). Second, it allows us to provide examples for the source of business driven requirements in a more direct manner (as we derive the set of requirements on an e-contract language based on the 4W framework)

The relation between the sample contract and the 4W framework is as follows:

Who: Lines 1-2, 35-41 define the two contracting parties, their contact data, and their signatures.

Where: Lines 18, 19-28 define rules that apply on the contract context. Line 18 defines the duration of the contract (one year). Lines 19-22 state the overriding on any other obligations existing for the same product. Lines 24-28 define copyright and ownership provisions and a reference to an additionally required external document (a distribution license).

What: Lines 3, 5-18, and 30-34 (except lines 6-7) describe the exchanged values and the processes and rules for their exchange.

Exchanged values: Line 3 defines the main service to be delivered by Company B to Company A (formatting service). Lines, 7-8, 10-11, and 30-34 provide a specification of the agreed products for formatting (the files must be in an electronic format, free of errors and viruses, and a file with formatting instructions will accompany them). Lines 17-18 define the corresponding financial benefit for Company B.

Exchange processes: Lines 5 and 13-14 define the activities that must and may be performed by the two parties (Company A has to deliver the files, Company B has to format the files, and Company B can distribute them).

Exchange rules: Line 15 defines a rule that applies on the exchange of values (Company B is not allowed to change the files).

How: Lines 6-7 describe the standard to be used for the delivery of the product (by uploading the files through the ftp protocol).

The provided example briefly illustrates the intention of each of the groups of concepts of the 4W framework. In this example, the processes to be performed by the parties and the rules for the value exchange are trivial. In Appendix B, we provide a second sample contract that contains more complex process description and rules that apply on it.

Next, based on the e-contracting concepts from the first and second level of detail of the 4W framework, we identify a number of general requirements on an e-contract specification language.

3. General requirements on an e-contract language

In this section, first, we investigate the general requirements on an e-contract specification language from the point of view of the highest level of abstraction of the 4W framework. Next, we continue with a description of general requirements that result from the concepts defined in the second level of detail of the 4W framework. At the end of each sub-section, we provide a summary and a list of identified requirements in the sub-section. Next to each identified requirement, we give references to existing research work related to it and indicate whether the requirement is technology or business driven.

3.1. High level general requirements

To define the actors in an e-contract (defined in the **Who** group of concepts), an e-contract specification language must support definition of *data item constructs*. A data item is data associated with an identifier that is unique for the e-contract. For example, to define the address of a party, the address data can be associated with an identifier “ADDRESS” (see example contract in Section 2.2, lines 36-41). Coupling between values and identifiers allows the value to be referenced from other e-contract elements. The data item construct is required for data specification in the other groups of concepts as well. For example, in the **What** group of concepts, data item constructs are required to define the quantities and qualities of products/services that are exchanged.

The context (**Where**) of a contract is expressed through various contract clauses often referred to as contract provisions (see example contract in Section 2.2, lines 15, 18, 20-22 and 24-28). Contract provisions define constraints, deontic assignments, and prescribe procedures related to states and processes agreed between the contracting parties. Data item constructs cannot express complex rules that define constraints (or even interrelated constraints) valid under certain conditions, procedures (valid under certain conditions), etc. To define the contract provisions in an e-contract, *rule specification constructs* that capture the context of the exchange (**Where**) and the rules for the product exchange itself (**What**) are required.

The exchanged values (**What**) between contracting parties can be products or/and services. A product is characterized by its qualities (e.g., size, weight) and quantity. A service is characterized by its qualities, quantity, as well as the steps that must be undertaken for the service delivery (see example contract in Section 2.2, lines 5 and 13-15). To define the steps for the service delivery, *process specification constructs* are required (see Appendix B, lines 9-57 for an example of a more complex contracting process). Process specification constructs are required for the definition of the contract enactment phase (**HoW**) as well. In general, process specification constructs are required to allow the specification of the processes that are agreed to be performed by the parties.

Finally, we can conclude that at least three fundamental language constructs are required for the specification of e-contracts, viz. data constructs, rule constructs, and

process constructs. The three classes of constructs can be compared to constructs in existing programming languages. For example, data definition constructs are used in most programming languages (Pascal, Java, etc.). Rule constructs are used in rule-based languages (e.g. expert system languages), and process constructs can be found in any process specification oriented language (BPEL, Workflow Nets, etc.). In the sequel of the paper, where appropriate, we use the experience collected from other programming languages as a source of inspiration for the requirements identification on the e-contract language constructs.

SUMMARY:

E-contracts need three general language constructs for the definition of data, rules, and processes.

REQUIREMENTS:

- **Requirement 1:** An e-contract specification language must provide support for the definition of data constructs.
Business driven. Also in: [Cro99], [Nea03].
 - **Requirement 2:** An e-contract specification language must provide support for the definition of rule constructs.
Business driven. Also in: [Cro99], [Nea03], [Gro99], [Goo00].
 - **Requirement 3:** An e-contract specification language must provide support for the definition of process constructs.
Business driven. Also in: [Cro99], [Nea03].
-

Requirements 1, 2, and 3 provide the basis for further requirements identification. In Section 4, we discuss a more detailed list of requirements on each of the three basic e-contract constructs.

An e-contract element is specified through one of the three basic constructs. Similar to data item elements, rule and process elements must have a unique designation, through which they can be identified and referenced from inside and outside the e-contract content. We recommend the use of strings for identifiers, as this will improve human readability of the e-contract when involvement of humans is required. An identification string must be unique for the whole e-contract, as the scope of contract elements is the whole document. In this way, any contract element can be reached from any other contract element, regardless of its position in the contract. The reason to require global scope of e-contract elements is that the relations between the e-contract elements are numerous and an element have to be referenced by other elements that can be positioned at any place in the e-contract. However, often, existing, predefined snippets of e-contracts will directly be included in newly defined e-contracts (for example a set of data and rule elements defining the governing law can be reused in all e-contracts of a company). Snippets can contain data, rule, and process elements and allow reuse of e-contract parts. When a snippet is imported in an e-contract, it must be given its own unique (for the whole e-contract) designation. The e-contract elements in the snippet use their previously defined identification strings. The scope of uniqueness of identification strings of elements in a snippet is only local (for the snippet). When referenced from outside the snippet, a contract element is

referenced through the combination of the snippet identification string and the element identification string. This prevents the need of renaming of identification strings when a snippet is inserted.

The complete e-contract (with all its elements and included snippets) must have its own designation through which it can be identified. The combination of a contract ID and element ID allows an element to be referenced from outside the contract as well.

SUMMARY:

For their identification and referencing, e-contract elements must have a unique identification string. To allow reusability of contract parts, predefined e-contract snippets can be directly included in an e-contract. To avoid renaming of the identifiers of elements defined in a snippet, the scope of their identifiers must be local (only in the scope of the snippet). An element defined in a snippet can be referenced through the identifier of the snippet combined with the identifier of the element. The scope of “non-snippet” elements is global i.e., the whole e-contract. Contract elements can be referenced from outside the contract. They are identified through the combination of the identification string of the e-contract and their own identification string.

REQUIREMENTS:

- **Requirement 4:** An e-contract specification language must provide a mechanism for the definition of a unique identification string for each e-contract element and the e-contract itself.

Technology driven. Also in: [Cro99].

- **Requirement 5:** An e-contract specification language must provide mechanisms for the inclusion in an e-contract of predefined snippets with unique (for the e-contract) identification strings.

Technology driven. Also in: (none).

3.2. Second level general requirements

In this section, we discuss general requirements set by concepts from the second level of the 4W framework. As concepts from the **How** group are related to the means for e-contracting, they set technology driven requirements on an e-contract language. We have identified requirements set by the “communication”, “standards”, and “structure” concepts. There are no specific requirements set on an e-contract language by the e-contract “representation” concept. The “content” concept is not relevant for the requirements analysis as all requirements are on set by the e-contract content. The concepts from the **Who**, **Where**, and **What** groups at the second level of the 4W framework are a source only of specializations of the high level general requirements. As these specializations requirements are not specific for a single concept, the consecutive examination of each concept and the related requirements will bring unnecessary repetition. We call all the specialization requirements “concrete requirements” and address them in Section 4. In this section, we start with a discussion of the “**communication**” concept and the requirements that are related to this concept. Next, we continue with the “**standards**” concept, and finish with a discussion on the “**structure**” concept and the requirements related to it.

3.2.1. E-contracts: extended with communication information

During contract enactment, contracting parties communicate to each other information related to the contract enactment (“enactment information”). For example, they can inform the counterparty of a contract violation, request an update on the progress of the service delivery, request change of the delivery date, etc. In traditional paper contracts, some of the enactment information and communication processes are included in the contract, whilst others are omitted in the contract content and are only communicated among the parties when and if needed. In the examples given above, the specification of the event of contract violation will be included in contracts together with a number of clauses that handle contract violations, while the request for information on the progress of the service delivery will usually not be included in the contract though it will take place. The information on the progress update will be exchanged among humans in a person to person non-formal conversation. Due to the requirement for automatic e-contract enactment, parties must agree on the communicated enactment information prior to the e-contract establishment. Without the intuition, common sense knowledge, and context adaptability of humans, an e-contract enacting system requires a precise definition of the possible scenarios and activities to be performed, information to be exchanged, and rules to be observed during contract enactment. In order to guarantee legal protection for companies from repudiation of the agreed to be communicated enactment information, a specification of this information and the agreed activities for its exchange have to be included in the e-contract content as well. Thus, compared to paper contracts, e-contracts are extended with a specification of the information that is communicated during the contract enactment and a specification of the activities required for its communication.

SUMMARY:

An e-contract is extended with a specification of the exchanged information and performed communication activities during contract enactment.

REQUIREMENT:

- **Requirement 6:** An e-contract specification language must support the definition of the exchanged information and performed communication activities during contract enactment.

Technology driven. Also in: [Nea03], [Cro99].

A special class of communicated enactment information is the set of possible requests for an e-contract update [Ang05]. During the contract enactment, it happens that companies have to apply changes to the agreed contract. For example, a company might like to change the product delivery address stated in the contract. In traditional paper contracting, small changes might not require contract re-negotiation and can be applied manually on the paper contract itself, in a contract appendix, while other changes can lead to significant deviations from the agreed contract and will lead to contract re-negotiation. Similar to paper contracts, e-contracts can also be subject to

updates. In order to provide high automation of the e-contract enactment, e-contracts must contain clear instructions for the allowed updates.

In Table 1, we present two dimensions of contract updates. The first dimension is based on the predictability property of contract updates. We identify two classes of updates in this dimension. Anticipated updates are updates the possible occurrence of which was anticipated during contract establishment. Exception updates are updates that were not anticipated during contract establishment. The second dimension is based on the protocol required for performing an update. In this dimension, we distinguish four classes of updates. Free updates are those updates that can be applied on the contract content without obstructions and limitations from the counterparty. For example, a URL address stated in the contract for obtaining a negotiated resource (a web-service, a digital product) can be updated at any time by the party running the server. For the counterparty this update is of importance for the proper obtaining of the resource but it will not lead to any changes in its business processes. Another example is the provision: “*The Board may change its contact person at any time upon written notice to the Contractor*”. Rule governed updates are updates that can be applied or disallowed based on rules provided in the contract. The accompanying e-contract rules indicate at any point in time if the change is currently allowed. Acknowledgment updates are updates that require the explicit acknowledgement from the counterparty (can be seen as a one step negotiation updates). Re-negotiation updates require multi-step negotiations for a contract update.

Table 1. Contract update classes

<i>Update class</i>	Anticipated	Exception
<i>Free updates</i>	X	--
<i>Rule governed updates</i>	X	--
<i>Acknowledgement updates</i>	X	X
<i>Re-negotiation updates</i>	X	X

Any of the four contract update classes can be anticipated during contract establishment. Consequently, an e-contract specification language must be suited to support the definition of anticipated updatable elements, the update class to which they belong, and the possible conditions for their update. An e-contract specification language must allow the definition of general procedures for handling of exception updates as well.

SUMMARY:

An e-contract can change during its enactment. To support automatic changes in an e-contract, an e-contract specification language must provide mechanisms that allow defining whether a contract element can change and the conditions for its change. It must allow the definition of general procedures for handling of unanticipated changes as well.

REQUIREMENT:

- **Requirement 7:** An e-contract specification language must provide mechanisms for the definition of anticipated changes and of general procedures for handling of unanticipated changes.

Technology driven. Also in: (none)

3.2.2. E-contracts: standardized

Standards play a pivotal role in e-contract content representation. For the automated e-contract management and enactment, a commonly agreed semantics and syntax of the e-contract elements is required. Providing a joint terminology between members of a community is a topic addressed by the Ontology research community. The Web Ontology Language [OWL] is the currently recommended by the W3C language for specification of ontologies.

To achieve a commonly agreed semantics of e-contract elements, each e-contract element must have a standardized name. The name of an e-contract element must belong to an e-contract ontology agreed upon by the parties. The name of an element is not its unique identifier. While, an identifier identifies uniquely a specific e-contract element, the name is used to define its semantic meaning. There can be several elements with the same name and with different identifiers. For example, a company might provide two e-mail addresses for contacts. The first e-mail address is to be used for reporting information related to the exchange of values and the second e-mail address is to be used for reporting contract violations. In this case, two data elements are required to define the two e-mail addresses. The name of the two elements is the same (e.g. "EmailAddress"), indicating that both elements contain an e-mail address. Each "E-mailAddress" element has a unique (in the scope of the e-contract) identifier. The identifier of an "EmailAddress" element can be used by other e-contract elements for referencing to it.

SUMMARY:

For the automatic interpretation of e-contracts, the semantics of the e-contract elements must be agreed and standardized between the contracting parties.

REQUIREMENT:

- **Requirement 8:** An e-contract specification language must provide a mechanism for associating an e-contract element with its semantics that is agreed among the contracting parties.

Technology driven. Also in: Association of process e-contract elements with ontologies of business processes is investigated in the SweetDeal project [Gro04].

3.2.3. E-contracts: structured

Similar to a software programming language, where program structure allows easier program creation, compiling, and maintenance, an e-contract structure model

facilitates the e-contract creation, processing, and maintenance. An e-contract structure allows (again similar to complex software projects) various e-contract parts to be created or maintained by different persons from different departments. For example, people from the legal department of an organization will only be interested in viewing/manipulating the clauses related to the legal context of an e-contract as these are in their area of expertise. The process specification part will be in the area of expertise of the business processes designers, and they will be interested in viewing/manipulating only this part of the e-contract. In this section, in line with these considerations, we describe an e-contract content structure. In our approach, a contract is structured according to the 4W groups of concepts. The main task is to identify which concepts of the 4W framework are beneficial for the structuring of the e-contract content. In the following sections, we identify these concepts and describe the proposed e-contract content structure. The proposed structure is not mandatory. It can further on be elaborated or a different e-contract structures can be defined (e.g., as in [Cro99], [Gri98], [XCBL]). Consequently, the derived requirements on the e-contract structure are recommended but not mandatory for consideration (see [Wie02] for a more detailed discussion on desired and mandatory property specifications).

SUMMARY:

E-contracts must have a structure that facilitates the e-contract creation, processing, and maintenance.

REQUIREMENT:

- **Requirement 9:** An e-contract specification language must provide support for the structuring of the defined e-contracts.
Technology driven. Also in: [Cro99], [Gri98].
-
-

3.2.3.1. The Who section

In an e-contract, two types of *actors* can be defined, i.e., parties and mediators (see Figure 2). Each e-contract contains information about two or more parties and zero or more mediators. *Parties* are actors that have rights and obligations that are listed in the e-contract. *Mediators* are actors that participate in the e-contract enactment but their rights/obligations are not stated in the e-contract. Consequently, mediators do not have to sign the e-contract. If their relations with the parties must be defined as legally binding, they can become a party in the same or in a separate contract where their rights and obligations are stated. An exception of this principle is the e-notary mediator. Parties can use e-notaries as mediators that guarantee the proper e-contract signing (in e-contract establishment or e-contract updating). In this scenario, e-notaries have to sign the e-contract as well and their signature proves the proper signing of the e-contract by the contracting parties [Ang05]. As mentioned in Section 2, *auxiliary implementors* are also part of the actors in an e-contract enactment. However, auxiliary implementors are not specified in the e-contract content and consequently this concept from the 4W framework can be omitted in the e-contract structure.

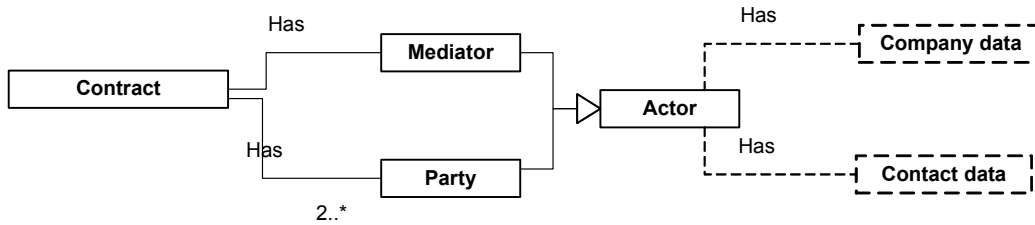


Figure 2 – Concepts in the Who section

As we have discussed in the introduction of Section 3.2.3, the proposed by us e-contract structure that is based on the 4W framework can be further on elaborated. We illustrate this by introducing two additional subsections for each e-contract actor, i.e., the *company data* and *contact data* subsections (denoted in Figure 2 by a dashed outline). The *Company data* concept contains data about an actor. Typically it will include company description, trade registration number, VAT registration number, address of registration, etc. The *Contact data* concept contains data for the possible ways an actor can be contacted. The company and contact data are expressed by means of data constructs. These two subsections of each actor provide a better structuring of the **actor** sections.

SUMMARY:

The *Who* section in an e-contract can be divided into subsections that describe the e-contracting **parties** and **mediators** (if any). An actor section can contain in separate subsections the actor’s **data** and **contact data**. Data constructs are used for the specification of each subsection.

REQUIREMENT:

- **Requirement 10:** An e-contract specification language can provide support for the definition of a “Who” section that contains subsections for each e-contracting party and mediator. Each subsection of an e-contracting actor can contain “data” and “contact data” subsections.
Technology driven. Also in: (none).

3.2.3.2. The Where section

The **Where** group of concepts contains provisions related to the context of the e-contract. We distinguish two basic aspects of the e-contract context that are defined in an e-contract, i.e., the business context and the legal context. Thus, we propose two subsections in the **Where** section. In addition, a third subsection can be defined to include other e-contract provisions that are not related to the legal and business context (see Figure 3). Data item, rule, and process constructs are used in the three subsections.

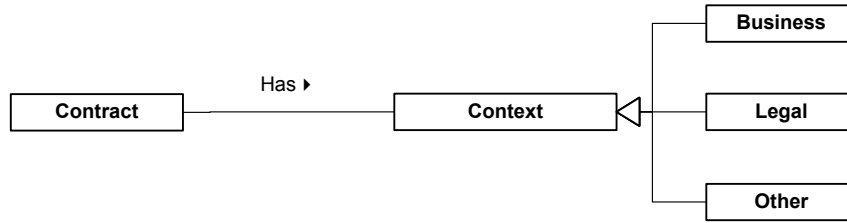


Figure 3 – Concepts in the Where section

SUMMARY:

The *Where* section in an e-contract can be divided into three subsections, i.e., **business**, **legal**, and **other** subsections. Data item, rule, and process constructs are used for the expression of the provisions in these subsections.

REQUIREMENT:

- **Requirement 11:** An e-contract specification language can provide support for the definition of a “Where” section that contains “business”, “legal”, and “other” subsections.

Technology driven. Also in: (none).

3.2.3.3. The What section

The What group of concepts contains concepts related to the exchanged values and the conditions for their exchange (see Figure 4).

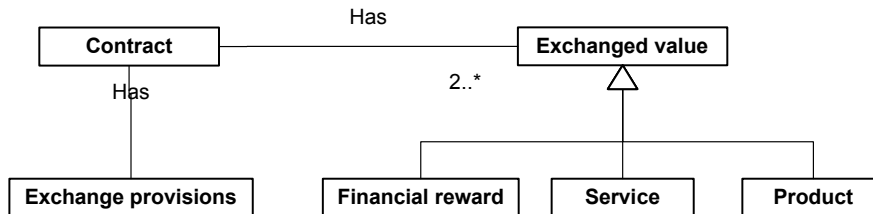


Figure 4 – Concepts in the What section

We distinguish two main sub-sections in the What section, i.e., the exchanged value description and the corresponding provisions for the value exchange. In separate subsections in the exchange value section, the values provided by each party are listed. In a case of product exchange, the product is described by means of data constructs. In a case of service exchange, the service is described through a combination of data and process constructs. The corresponding financial reward for the received value (in non-barter exchanges) uses the same constructs as a service description subsection. The value exchange provisions subsection requires the use of rule and process specification constructs.

SUMMARY:

The *What* section in an e-contract can be divided into two subsections, i.e., **exchanged value** and **exchange provisions**. Data, and process constructs are used for the expression of the exchanged values and rule and process constructs for the exchange provisions.

REQUIREMENT:

- **Requirement 12:** An e-contract specification language can provide support for the definition of a “What” section that contains “exchanged value” and “exchange provisions” subsections.

Technology driven. Also in: (none).

3.2.3.4. The HoW section

From the How group of concepts, only three concepts are represented in the e-contract content, i.e., the e-contracting process, standards, and communication (see Figure 5). The e-contracting process concept is optional. The subsection that corresponds to this concept describes the process that will be followed when the current agreement ends and a new one has to be signed or when the current agreement has to be renegotiated. In this subsection, data item, rule, and process constructs are used.

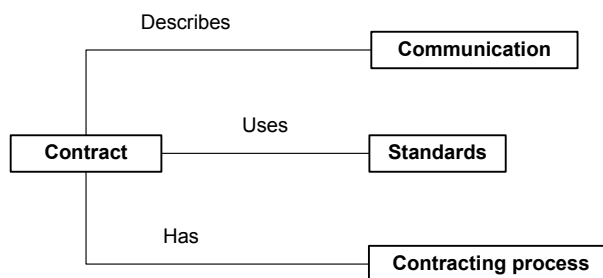


Figure 5 – Concepts in the HoW section

The standards subsection contains information related to the standards used in the e-contracting process and documents. For example, the communication protocol that will be used during contract enactment, the used ontology, the used contract template, etc. In the standards subsection, data and rule constructs are used. In the communication subsection, the enactment information and processes are defined (as discussed in Section 3.2.1). In this subsection, data, rule, and process constructs can be used.

SUMMARY:

The *How* section in an e-contract can be divided into three subsections, i.e., **communication**, **standards**, and **contracting process**. Data, rule, and process specification constructs are used for the expression of the communication related

elements and contracting processes, data and rule constructs for the standards elements.

REQUIREMENT:

- **Requirement 13:** An e-contract specification language can provide support for the definition of a “How” section that contains “enactment information”, “standards”, and “contracting process” subsections.
Technology driven. Also in: (none).
-
-

4. Concrete requirements on an e-contract specification language

In Section 3.1, we have shown that three basic language construct must be supported by an e-contract specification language. In Section 3.2, we have presented a number of general requirements that are not related to the types of constructs used in an e-contract language. In this section, we provide a list of concrete requirements that are specializations of one or set of the general requirements presented in Sections 3.1 and 3.2. The identified concrete requirements in this section are categorized on the basis of the three basic e-contract constructs required in an e-contract language. We start with concrete requirements on the data constructs, continue with requirements on rule constructs, and finish with requirements on process constructs.

4.1. Concrete requirements on data item constructs

As discussed in Section 3.1, a data construct is required for the definition of data items. Data items are data definitions in the e-contract that are to be used by the contract enactment system. Furthermore, data items can have the function of variables in programming languages, allowing an item to be referenced by other e-contract elements. As in many programming languages, we require that data items belong to a specific data type. This allows easier contract processing and sets basic constraints on the allowed values for a data element. We have identified two classes of required data types, i.e. standard data types and special data types. Next, we list the identified required data types. After each data type definition, we provide an example of a data element that was found in an existing business contract and is from the given data type.

4.1.1. Standard data types

Standard data types required in an e-contract language are:

String. The string type is required for the support of definition of string data items. A construct of the type “string (number)” can optionally be defined to limit the number of allowed character in a string.

Example string data: “Terrific Consulting” (a company name as defined in line 6 in Appendix B).

Number. The number data type is required for the support of definition of number data items. A construct of the type “number (number, number)” can optionally be

defined to limit the allowed number of digits before and after the decimal point. Differentiation of number types (e.g. integer, real) increases the language richness and allows improved basic data value verification.

Example number data: 8026586060 (for a phone number).

Boolean. The boolean data type is required for the support of definition of boolean data items. A boolean data item can be assigned a true/false value.

Example boolean data: True (for “Work is copyrighted”)

Set. The set data type is required for the support of definition of sets of data items. Sets can be used for example for grouping of contract elements sharing a common property, for defining a set of allowed values of a data item, etc. A number of predefined sets can be part of an e-contract specification language. For example, the **state** set constant type is required for the support of definition of the state of an e-contract element (see Section 4.2.3). The state of an element can be enabled or disabled. The **status** predefined set is required for the definition of the status manipulation of an e-contract process allowed to a party (see Section 4.3). The status of a process can be one of the values: started, suspended, resumed, aborted, or finished. Similarly, other predefined set constants (e.g., currency set, country set) can be provided in an e-contract language.

Example set data: “enabled, disabled” (a state predefined set).

List. The list data type is required for the definition of ordered sets. Lists can be used for example for the definition of the order of firing of processes or rules.

Example list data: “Rule1, Rule2, Rule3”.

Record. The record data type is required for the definition of bundles of data items. A record consists of several fields that are grouped together. Every field contains a data item that is from one of the predefined data types. This construct is beneficial in the context of more complex, data intensive contracts. It allows better structuring and manipulation of the data. Records can be used also for defining special data types (discussed in the next section).

Example record data: (see Section 4.1.2, money, event, ERR data types).

4.1.2. Special data types

In addition to the standard data types, we have identified the need for data types that are of importance for e-contracts but are not widely accepted in programming languages (we call them “special data types”). A special data types can be seen as a specialization of one of the standard data types. Next, we describe the required special data types.

Date/Time. The date and time data types are required for the support of definition of date/time data items.

Example date data: “Dec 31, 1999” (a deadline as defined in line 59 in Appendix B).

Example time data: “10-00-00” (for opening time).

Money. The money data type is required for the definition of amounts of money from a specific currency.

Example money data: “\$400 000” (a fee to be paid as defined in line 81 in Appendix B).

Event. The event data type is required to specify events that can occur during contract enactment. A data element of this type can have two values: occurred, not occurred. An event data item is required for an e-contract language to specify events that trigger firing of rules, or to specify events that can be sent during contract enactment. As the

time and date of occurring of an event are of importance and will be exchanged together with the event notification, a data item from the event data type must contain the date and time of occurrence as well.

Example event data: “occurred, 11-20-00, 10-02-2005” (for the event “product is delivered”).

External Resource Reference (ERR). The ERR data type is required for specification of references to external contract elements. An external reference is a “link” to an external resource (e.g., an umbrella contract, subsidiary agreement, external information, general provisions). The value of an external reference variable is an URI. The information that can be exchanged during e-contract enactment about a resource is whether the resource has changed its status (i.e., available, unavailable, or new content). For example, the content of the resource can change, the resource can be provided at a later stage of the contract enactment (or for a limited amount of time), etc. For every external resource defined in the e-contract, it must be stated if the resource is legally binding. In contracts, parties specify and sign the terms and provisions for their agreement. Thus, any external references to the contract will normally be not legally binding. However, in some situations, external references can be legally binding. For example, when they refer to previously signed documents (umbrella contracts), to general governing rules and conventions, etc.

Example ERR data: “http://www.interdigm.com/hosting/, available, non-binding” (for an online resource containing currently offered services).

SUMMARY:

In e-contracts, data items can be of various data types. Typing of data items allows easier contract processing and sets basic constraints on allowed values for a data item.

REQUIREMENTS:

- **Requirement 14:** An e-contract specification language must provide support for the definition of data items from string, number, boolean, set, list, and record data types (specialization of **Requirement 1**).
Business driven. Also in: [Cro99].
- **Requirement 15:** An e-contract specification language must provide support for the definition of data items from the special data types date/time, money, event, external resource reference (ERR) (specialization of **Requirement 1**; related to **Requirement 6**).
Business driven. Also in: (none)

4.2. Concrete requirements on rule constructs

As discussed in Section 3.1, a contract rule construct is required for the specification of contract provisions. Contract rules are business rules relevant to the contracting relations between the contracting parties. In the existing literature four basic types of business rules are identified, i.e., integrity rules, derivation rules, reaction rules, and deontic assignments [Wag02], [Wag03]. We have identified examples of each of these four rule classes in existing business contracts. This shows that constructs for each of them has to be provided in an e-contract specification language. In addition, a free-

text rule construct can be used for expression of any provisions targeted only for reading by humans. Every business rule can either be in an enabled or disabled state (see Section 4.2.3 for details). Two rules can conflict to each other when both apply at the same time. This requires prioritization of execution of rules, i.e., when two or more rules have conditions that are satisfied, prioritization can be used to suppress the execution of some of them. The idea of prioritizing contract rules was initially presented in [Gro99].

In the following subsections, we describe each of the rule classes and the requirements that must be satisfied by an e-contract specification language for the support of definition of the complete spectrum of business provisions.

SUMMARY:

Business rules can be integrity, derivation, reaction, and deontic rules. Rules can override other rules.

REQUIREMENT:

- **Requirement 16:** An e-contract specification language must provide mechanisms for prioritization of rules.
Business driven. Also in: [Gro99].

4.2.1. Integrity rules

Integrity rules represent assertions that must be satisfied in the evolving states and state transitions. Two sub-classes of the integrity rules exist, i.e., state constraints and process constraints (also referred to as dynamic constraints) [Wag02].

State constraints set constraints on states of objects at any point in time.

Example1: “*There shall be three arbitrators*”.

Example2: “*... a fee not to exceed \$1,405,000*” (see line75 in Appendix B).

State constraints define a condition that must be satisfied. The state constraint is presented as a boolean expression that must hold true. Operators like <, >, =, >=, <=, IS_IN must be part of the language specification. The IS_IN operator is used to check if a data element is part of a set of data elements. The complete set of required operators and their semantics still has to be defined.

Dynamic constraints set restrictions on transitions from one state to another.

Example: “*Any research report (...) shall be reviewed and approved prior to release*”.

Dynamic constraints define constraints on the changing of a state. In e-contracts, dynamic constraints are applied to define constraints on the transition from a completed activity(s) to the following activity(s), i.e. for changing the status of activities (see Section 4.3). Furthermore, dynamic constraints are applied to define constraints for changing the state of e-contract elements (from enabled to disabled and vice versa). Consequently, the construct for support of dynamic constraints on e-contract elements must define condition(s) for the allowance/disallowance of a new specific state of an e-contract element.

SUMMARY:

E-contract can contain integrity rules. Integrity rules represent assertions that must be satisfied in the evolving states and state transitions during contract enactment. Two types of integrity rules exist, i.e., state constraints and dynamic constraints.

REQUIREMENT:

- **Requirement 17:** An e-contract specification language must provide support for the definition of state and dynamic constraints (specialization of **Requirement 2**; related to **Requirement 20** and **Requirement 21**).

Business driven. Also in: limited support for dynamic constraints is provided in [Cro99].

4.2.2. Derivation rules

A derivation rule prescribes the way for obtaining the value of a data element. Two types of derivation rules can be distinguished, i.e., computational and linguistic rules. In **Computational rules**, the value of a data element is calculated through a mathematical expression.

Example: “Expenses to be billed include travel (\$.25 per mile for auto travel), lodging and meals while in Iceberg, long-distance phone calls, and any copying and mailing services” (see line 75 in Appendix B).

For the support of computational rules, an e-contract specification language must allow definition of mathematical expressions. Support for operators like +, -, *, /, √, and order execution definition syntax like (“,”) is required.

In **Linguistic rules**, information for a data element is derived from existing information (or from information that itself can be derived).

Example1: “The project will be considered complete when the written report described above is submitted” (see lines 83-84 in Appendix B).

Example2: “Any amendments of or extensions to this contract shall be considered valid and binding only if they follow the rules of evidence and formation set forth below...”

The examples of derivation rules that we have found in various contract clauses have one common pattern. In this pattern, a data item is associated with a specific value, if a number of conditions are satisfied. In Example1, the “project” data element obtains the value “complete” if a report is submitted. Thus, the construct required in an e-contract language must assign a value to a data item when certain conditions are met. When a data item can obtain several different values, the data item has to be of type set or list. This limited form of linguistic derivation rules can be supported by copier reaction rules as well (discussed in section 4.2.3).

SUMMARY:

An e-contract can contain mathematical and linguistic derivation rules. The linguistic derivation rules used in business contracts follow one basic pattern, which allows derivation rules to be expressed by copier reaction rules.

REQUIREMENT:

- **Requirement 18:** An e-contract specification language must support the definition of mathematical and linguistic derivation rules (specialization of **Requirement 2**; related to **Requirement 19**).

Business driven. Also in: (none).

4.2.3. Reaction Rules

Reaction rules are rules that lead to invocation of actions (execution of processes, changing of data values, etc.) in response to certain events, provided that certain state conditions are true. These rules are often referred to as Event Condition Action (ECA) rules. Condition Action (CA) rules can be seen as special cases of the ECA type of rule. The structure of an ECA rule is encoded in its name. An ECA rule construct must provide definition of the events and conditions for the firing of the rule, and the actions to be performed. In [Ros01], three classes of reaction rules are distinguished according to the type of action defined in the action sub-construct, i.e., enabler, executive, and copier reaction rules.

Enabler reaction rules create/delete or enable/disable e-contract elements.

Example: (none – enabler reaction rules are specific for e-contracts and there are no examples from existing business contracts)

In paper contracts, any changes to an established contract are made when companies observe the need for change and if they agree to the change. As we have discussed in Section 3.2.1, in e-contracts, the free and rule governed updates are agreed upon in advance. For this reason, e-contracts contain the text that has to be added or deleted in its content. Often complete data item, rule, or process specification (or a set of the) will be initially or during the contract enactment disabled. These clauses of the contract must be indicted as non-active parts of the e-contract. When an enabler reaction rule fires, a data item, a rule, or a process can change its status (to enabled or disabled). Thus, **enabler** ECA rules do not directly delete or create new content but only disable or enable previously defined contract elements. When a rule of this type evaluates to true, the contract management system has to cater for the change of the state of the relevant e-contract elements. Enabler rules require the action sub-construct to contain the new state and the identification string of the element whose state is to be changed. This limited functionality of enabler rules required in e-contracts can be delivered by copier reaction rules as well (copier reaction rules are discussed below). A detailed discussion for the approaches for handling of e-contract updates is provided in [Ang05].

Executive reaction rules cause a process to be triggered or a rule to fire.

Example1: “The payments will be made in three installments: \$400,000 at the signing of this contract; \$400,000 upon completion of the Iceberg meeting described above; and \$605,000 upon completion of this project” (see line 81-83 in Appendix B).

Example2: “If this work has been copyrighted, a photo-copy of the copyright registration should be submitted to the publisher.”

When an ECA rule is from the **executive type**, then the action sub-construct invokes the execution of a process defined in the e-contract (see Section 4.3), or the immediate firing of another rule. Thus, the action sub-construct has to contain a process or rule identification string (or a set of IDs) that must be triggered/fired. The e-contract

monitoring system evaluates the contract rules, and when an executive reaction rule evaluates to true notifies the contract management system which process to be started or a rule to be executed. A reaction executive rule construct must support definition of the sequence of firing of rules or processes when several rules/processes must be fired at the same time.

Copier reaction rules set the value of a data element.

Example1: "Publisher can (...) drop the price after a reasonable amount of time".

Example2: See examples in linguistic derivation rules.

The data element whose value is set/changed by the copier reaction rule can be specified either in the e-contract (see Example1), or in the e-contract environment (for example a data item indicating the current status of a process). Copier reaction rules can specify in advance the new value to be assigned when the rule fires. However, the value can be provided at a later stage during the e-contract enactment as well (see Example 1). When an ECA rule is from a **copier type**, the action sub-construct must provide the targeted data item and the new value to be assigned (if known).

SUMMARY:

An e-contract can contain enabler, executive, and copier reaction rules. Enabler rules provide means for activating or deactivating parts of the e-contract content. Executive rules provide means for firing of rules or processes. Copier rules provide means for changing the value of a data element.

REQUIREMENTS:

- **Requirement 19:** An e-contract specification language must provide means for the definition of enabler, executive, and copier reaction rules (specialization of **Requirement 2**).

Business and technology driven. Also in: executive reaction rules are discussed in [Chi03] and briefly in [Cro99].

- **Requirement 20:** An e-contract specification language must provide mechanisms for indication of the current state of a data item, rule, process/activity (related to **Requirement 7**).

Technology driven. Also in: (none).

4.2.4. Deontic assignments

Deontic assignments regulate the assignments of powers, duties, etc. In contracts deontic assignments indicate the rights, obligations, and prohibitions each party (or a person/role from a party) is given/imposed during the contract enactment. In e-contracting, deontic assignments are related to integrity constraints and reactive rules. Parties can have obligations, rights, and prohibitions to perform certain processes or single activities. This requires that the corresponding executive reactive rules must be related to the deontic assignments of parties.

Example1: "Interdigm may refuse to accept any order".

Example2: "Contractor will write..." (see line 52 in appendix B).

Parties can have rights or obligations to update the e-contract correspondingly (see Section 3.2.1), by enabling and disabling e-contract elements (see Section 4.2.3 enabler rules), or by providing new values for data elements (see Section 4.2.3 copier rules). This requires from an e-contract language to support the linking between enabler and copier reactive rules and the corresponding deontic assignments.

Parties can have obligations to observe certain constraints or rights to break constraints under certain conditions (see Section 4.2.1). In this case, an e-contract language must provide mechanisms for the linking of static integrity constraints with the corresponding deontic assignments.

Example3: "The contractor agrees to abide by the requirements of the following as applicable: Title VI and VII of the Civil Rights Act of 1964".

It must be noted that deontic assignments can be imposed not only to a party but to specific roles or persons from a party. This requires that deontic assignments in an e-contract language can be specialized per party to a role or even person level.

Example4: "Only the branch manager has the right to approve the start of contract re-negotiation".

SUMMARY:

An e-contract contains deontic assignments to parties. A party can have the right/obligation/prohibition to perform an action/process or an update of the e-contract content. A party can have an obligation to observe a constraint or right to break a constraint under certain conditions. Deontic assignments can be imposed on parties, roles, persons, etc.

REQUIREMENT:

- **Requirement 21:** An e-contract specification language must provide support for definition of deontic assignments to parties or members of a party and for linking these assignments to the corresponding rule and process specifications (specialization of **Requirement 2**; related to **Requirement 17** and **Requirement 19**).

Business driven. Also in: [Nea03].

4.2.5. Free-text rule construct

Formalization of certain contract provisions is not necessary as it does not deliver any value for an e-contracting system. For example, integrity constraints related to copyrights, property rights, etc. cannot be automatically monitored. As these rules can be monitored only by humans, their formalization is superfluous for an e-contracting system.

Example1: "The author does hereby warrant that he/she is the author and sole owner of the Work or has been assigned such rights; that it is original and that it contains no matter unlawful in its content, nor does it violate the rights of any third party" .

Example2: "Interdigm owns all right, title and interest in Interdigm's trade names, service marks, inventions, copyrights, trade secrets, patents, and know-how relating to the design, function, or operation of Plans and of the hardware and software

systems and resources necessary to provide the individual service elements of which they consist”.

Furthermore, as already discussed in Section 3.2.1, during the e-contract enactment, unanticipated events can occur. In contract law practice, these events are often referred to as “Force majeure” or “Act of God”. Though sometimes the handling of such events might be automated (e.g. by immediately terminating the contract), often parties will prefer a manual approach for handling of exception events, which cannot be formally specified in the e-contract

Example3: “Either party’s performance of any part of this Agreement shall be excused to the extent that it is hindered, delayed or otherwise made impractical by reason of flood, riot, fire, explosion, war, acts or omissions of the other party or any other cause, whether similar or dissimilar to those listed, beyond the reasonable control of the non-performing party. If any such event occurs, the non-performing party shall make reasonable efforts to notify the other party of the nature of such condition and the extent of the delay and shall make reasonable, good faith efforts to resume performance as soon as possible”.

A free-text rule construct is sufficient to represent these kinds of rules. A free-text rule construct contains the rule in a free text form and indicates that it contains a rule that is to be monitored and evaluated by humans. This allows its automatic extraction from the e-contract and representation to the e-contracting parties for manual evaluation.

SUMMARY:

In e-contracts, certain integrity rules cannot be automatically monitored and their formal representation is superfluous. Furthermore, for certain reaction rules the conditions for firing and the activities to be performed when fired cannot be formally specified.

REQUIREMENT:

- **Requirement 22:** An e-contract specification language should provide mechanisms for specification and indication of free-text rules (specialization of **Requirement 2** and **Requirement 7**).

Technology driven. Also in: [Cro99].

4.2.6. Definition of sub-constructs in e-contract rule constructs

As we have shown, an e-contract rule construct is built of various sub-constructs. For example, in the case of reactive rules, the rule construct has event, condition, and action sub-constructs. Identification of detailed requirements on the specification of conditions and mathematical/logical expressions is a challenging task and requires further attention. We illustrate the complexity of the problem with a short example on requirements on the specification of time points and time intervals which can be part of rule conditions and expressions. We use the discussion from [Nea03] on the requirements for specification of temporal expressions. Further work is required for the identification of the complete set of requirements for specification of rule conditions and expressions.

Specification of fixed time-points and time-intervals is straightforward and does not need special attention. Exception mechanisms for the time-point and time-interval are required (for example “on 28.02 except in a leap-year”). Specification of relative time points and intervals requires mechanisms that allow the definition of time points relative to an occurring event or a change of state. For example, “3 days after the receiving of the goods” is a relative time point, fixed by the event receiving of goods. An example for a relative time interval is “the period starting with the delivery of the first batch and ending with the delivery of the last batch”. In [Nea03], “sliding time windows” are paid explicit attention. Sliding time windows are time intervals with fixed duration that have infinite number of starting time points. An example for a sliding time window is “an event must occur in **any 3 day period**”. Sliding time windows are used in business contracts and consequently an e-contract specification language must provide mechanisms for their definition. Specification of the granularity of sliding windows shifts defines when the sliding window performs the next shift. For example, defining a granularity of one day in the sliding time window “any 3 day period” means that if an event occurs on Monday, the next event should occur not later than by the end of Thursday. However, if the granularity is one hour, and an event occurs on Monday 10.00 am the next event should occur not later than 11.00 am on Thursday.

SUMMARY:

E-contract expressions can contain specifications of fixed and relative time-points and time-intervals, as well as sliding time windows.

REQUIREMENT:

- **Requirement 23:** An e-contract specification language must provide mechanisms for the definition of fixed and relative time-points and time-intervals, as well as of sliding time windows with different levels of granularity. It must allow specification of exception time-points and time-intervals in the temporal definitions (related to **Requirement 2**).

Business driven. Also in: [Nea03].

4.3. Concrete requirements on process constructs

As discussed in Section 3.1, to specify the activities that are agreed to be performed during contract enactment, a process language construct is required.

Example: “Terrific Consulting agrees to provide the following products and services to the Iceberg County Art Center: A. Information gathering. (...) B. Contractor will design a one-page membership survey (...) C. Contractor will design a “community leader interview” format (...) D. Contractor will travel to Iceberg for a one-day stay to meet for half a day with Client’s board (...) F. Contractor will write (...)” (see lines 6-57 in Appendix B for the full text).

One possible approach to define in an e-contract the processes agreed to be performed by the parties is by using the classical definition that a process comprises one or more activities connected through transitions [WMC]. This approach has been chosen in [Cro99]. However, often companies would prefer to specify the agreed processes in a

less prescriptive and more flexible manner. In these cases, the agreed process is specified through activities that must be performed and rules on their performance [Ang02]. The rules can specify order of execution of certain activities (when such exists), mutual exclusion of activities, etc. A combination of the two approaches is also possible.

In both approaches, the starting of a process is defined by executive reaction rules (see Section 4.2.3). During contract enactment, parties can send information about the current status of the process, i.e., a process was started, suspended, resumed, aborted, or finished. The possible options for process manipulation (i.e., changing the status of the process) by a party must be defined in the e-contract content (see Section 3.2.1). Deontic assignments are used for the specification of the parties' rights and obligations on process manipulation and monitoring.

An activity is the atomic building element of processes. In the prescriptive process specification approach, an activity can be started when its preceding activity has been completed and the dynamic constraints imposed on it (if any) evaluate to true. In the rule based approach, an activity can be started when the imposed state and dynamic rules evaluate to true.

Similar to processes, the state of an activity specification can also be changed during contract enactment (disabled or enabled). An activity status can also be changed by the authorized parties (as in processes – started, suspended, resumed, aborted, or finished) and e-contracts must contain the corresponding information about the obligations of a party to perform or the rights to control an activity.

In the CrossFlow project [Cro99], a transactional model for support of transactional properties in the agreed processes is proposed. For the support of the model additional information in e-contracts like possibilities for rollback of activities, compensation activities, safe points for rollbacks, etc, is required. However, this model requires further attention and research work before requirements on transactional support can be defined.

SUMMARY:

E-contracts can contain complex process specifications built of activities. Processes can be specified in a prescriptive or in a less prescriptive, rule-based manner. A process is started through executive reaction rules and a transition to the next activity is governed by dynamic constraints. In rule based process specifications, static constraints can be used to govern the starting of an activity as well. Process and activities can be in an enabled or disabled state. Parties can exchange information about the current status of processes/activities. The rights and obligations of each party to perform manipulations on the status of each process/activity are defined in the e-contract through deontic assignments.

REQUIREMENTS:

- **Requirement 24:** An e-contract specification language must provide mechanisms for definition of detailed process specification comprised of activities (specialization of **Requirement 3**).
Business driven. Also in: [Cro99], [Gri98].
- **Requirement 25:** An e-contract specification language must provide mechanisms for the association of reaction rules with process specifications and of integrity rules with activity specifications.

5. Related work

In this section, we discuss existing work on specification of the separate e-contract constructs and on complete e-contract specification languages. We start with a discussion on the specification of business rules, continue with existing work on business process specification, and finish with an overview of several existing e-contract languages.

5.1. Related work on rule specification

Rules and business rules in particular have been paid significant attention in the recent years. An extensive discussion on business rules in practice can be found in [Wag02]. Next, we provide a brief overview of the existing work.

Prolog is a programming language based on the specification of facts and derivation rules. The solid theoretical foundation of Prolog motivated its use as a starting point in many of the approaches on specification of derivation business rules.

In [Gro99], an approach for specification of business rules in electronic contracts is presented. The approach is based on Ordinary Logic Programs (OLP) and is extended to Courteous Logic Programs (CLP), which allows prioritization of conflict handling. Furthermore, the authors represent the CLP rules in an XML syntax (called Business Rules Markup Language). However, one of the main deficiencies of the work described in [Gro99] is the lack of full support for all four classes of business rules. For example reactive rules cannot be expressed through CLP. The use of XML for representation of business rules, as observed in [Gro99], has several advantages. It allows easier processing of the document, integration with the internet research world, and easier language standardization. Other research projects on business rules came to similar conclusions and have used XML languages for representation of business rules as well.

RuleML [RuleML] is an effort that aims at defining an XML-based rule markup language. Derivation rules are currently the main objective in the RuleML initiative. RuleML has the ambitions to extend its specification in the future by including reaction rule semantics as well.

Reaction rules and more specifically ECA rules have been seen in several research works as a sufficient construct to express all possible rules in a business contract. This approach was taken for example in [Chi03], [Per03]. However, ECA rules are not sufficient to express the rules that exist in a business contract. As discussed in Section 4.2, specification of integrity constraints, derivation rules, and deontic assignments requires the usage of constructs that differ from the ECA rule construct.

Thus, none of the existing research efforts can provide a complete solution for the specification of e-contract rules. Existing results can only be used as a foundation and inspiration for the construction of the e-contract rules in an e-contract specification language.

5.2. Related work on process specifications

Many standardization efforts in the field of process specification exist. Results and experience from these efforts can be used for the process specification construct in an e-contract specification language. The WorkFlow Management Coalition provides standards for workflow process specifications [WMC]. In the ebXML standardization effort, a Business Process Specification Schema is provided for process specification [ebX01]. BPEL4WS [BPEL] targets the specification of business processes in the context of Web Services. XRL [Aal03] is an XML process specification language based semantically on Petri Nets. This list of standardization efforts can be significantly extended. However, as none of these languages targets explicitly specification of processes in an e-contract, they all need certain extensions to satisfy the requirements presented in this paper. These extensions are due to the relations of processes and e-contract rules and the positioning of the process specification in the e-contract (identification strings, names, etc.). Thus, no significant changes in the underlying process specification language model are required. As all of the listed standardization efforts have an XML representation, each of them is a good candidate to be adapted for use in a contract specification language. The choice on the process specification language can be influenced by its underlying philosophy and the context of an e-contract. For example, constructs based on BPEL4WS would be more beneficial in the context of contracting of web services.

5.3. Existing e-contract languages

A number of efforts on definition of e-contract specification languages exist. Next, we discuss three of these efforts. An outstanding characteristic of these three projects is that they aim at defining an e-contract language that provides support for high level of automation of the interpretation of e-contracts.

In the CrossFlow project [Cross], an XML based e-contract specification language is defined. The CrossFlow project concentrates on the automatic support for cross-organizational workflow management in virtual enterprises. To serve the project goals best, the language concentrates on the data and process e-contract specification elements [Cro99], [Koe00]. A limited support of reaction rules is envisioned. However, integrity and derivation rules and deontic assignment are not paid attention to. This is due to the fact that in CrossFlow contracts serve the role of specification documents used by the enactment systems for the proper performance of the agreed processes, and not as legally protective documents. As a result, definition of rule constructs required by the e-contract legal protective functions (prohibitions, rights, state constraints, etc.) is not supported. Provisions related to the context of the e-contract are defined in human readable form and cannot be automatically interpreted. This is the major deficiency of the CrossFlow e-contracting language. The advanced e-contract process specification mechanisms provided by the CrossFlow e-contracting language allowing highly automated management of cross-organizational process is its outstanding feature. In CrossFlow, the proposed e-contract structure is based on the constructs provided in the CrossFlow e-contract language rather than on the e-contract business logic. A CrossFlow contract has a parameter definition section, process specification section, etc. This structure does not allow easy e-contract creation and maintenance of the business related aspects of e-contracts. In our approach, the e-contract structure is based on the business logic of e-contracts (see Section 3.2.3).

The SweetDeal project [Gro04] aims at building a rule-based system for representing e-contracts in a digital form. The primary goal of the SweetDeal project is to provide a foundation for representing e-contracts between Semantic Web Services (SWS) that can automatically be established and enacted. The approach uses the existing RuleML and DAML+OIL/OWL standardization efforts. The research work concentrates on representation of exception handling rules in e-contracts. As e-contract rule specifications are based on Situated Courteous Logic Programs and encoded in RuleML, the SweetDeal approach does not support the complete range of rule types discussed in this report. Another shortcoming of the approach is the lack of support of advanced e-contract process specifications. Processes can only be invoked through business rules, but cannot be specified in the e-contract content. The main contribution of this project is the linkage of the e-contract elements with their semantics defined outside of the e-contract. Another virtue of the project is the formal definition and representation of certain e-contract rules, which allows their automated processing, interpretation, and reasoning about. In the SweetDeal project, no e-contract structure is defined.

The Business Contract Language (BCL) is a result of combined efforts of two previously defined e-contracting frameworks, i.e., the Business Contract Architecture (BCA) and the Enterprise Contract Language (ECL) developed in DSTC and the University of Kent respectively. A description and comparison of the two frameworks is provided in [Nea03]. Furthermore, in [Nea03], three requirements on an e-contract language are identified. First, the authors discuss the need of specification of behavioral patterns in e-contracts. Behavior patterns specify the actions that will be performed by the parties. The requirement on behavior patterns is equivalent to **Requirement 3** in our work (see Sections 3.1). The second requirement on e-contract languages defined by the authors is the need of an authorization and accountability model. The authorization and accountability model specifies the permissions and obligations of the parties (authorization) and the actors responsible for the performance of the defined activities (accountability). This requirement is equivalent to the requirement on deontic assignments discussed in Section 4.2.4. The third requirement is on the need for support in an e-contract language of complex temporal expressions. We have directly referred to and included this requirement in Section 4.2.6. In [Lin04], implementation details of BCL are discussed. BCL incorporates deontic assignments, a limited support for integrity rules, and event based process specification. From the existing publications on BCL, we cannot evaluate the semantic potential of the used e-contract constructs and their complete mapping to the requirements identified in this report. The empirical approach undertaken for the definition of BCL through the consideration of a large number of test cases makes it the most related to the existing contracting practices academic work. However, this approach lacks structure and completeness of the identified e-contract language requirements. Similar to the already discussed e-contract languages, BCL is encoded in an XML syntax.

6. Conclusions

For the support of electronic contracting, an e-contract specification language that can automatically be interpreted and reasoned about is required. In this paper, we present a list of requirements on an e-contract specification language. The list of requirements represents both the business and technology driven requirements on an e-contract

specification language. Clear references to existing research work in the domain of e-contract specification and representation are made. Due to domain specifics, we expect that certain requirements will be added to the presented list of requirements. For example, the need of an advanced transactional support in process intensive domains will add new requirements to the process specification constructs. Transactional support in electronic contracting is currently being researched in the XTraConServe project [XTC].

As we have already discussed in Section 5.1, an XML based language can offer many advantages as an e-contract specification language. In Appendix A, we provide a sample, XML based, e-contract specification language that satisfies the requirements identified in this report.

In previous work [Ang04], we have identified five main paradigms of e-contracting, i.e., micro-, just-in-time-, precision-, enactment-, and management-contracting. Each paradigm is a specific new business opportunity introduced by e-contracting. The requirements on an e-contract language that we have identified in this report provide the starting point for the design of e-contract languages that can be automatically interpreted. Thus, these requirements set the foundations for the support of precision-, enactment-, and management-contracting paradigms, as each of them requires automatically interpretable e-contracts. However, for the complete support of the three paradigms, additional requirements for each paradigm must be identified. For example, in the management-contracting paradigm, for the automated monitoring of the value of a contract rule, an e-contract management language must provide a mechanism for the definition of the frequency of evaluation of the rule by the monitoring system. Similarly, in the enactment-contracting paradigm, an e-contract enactment language must provide mapping mechanisms between the internal business processes and the agreed in the e-contract processes. As the additional information that has to be specified in the precision-, management-, and enactment-paradigms is not part of an e-contract, we do not address the additional requirements in our report. However, it is important that any additional requirement specifications are aligned with the defined e-contract specification requirements. A starting point on requirements on a language for the management-contract paradigm is provided in [Lin04].

Another issue only briefly addressed in this paper (see Section 3.2.2) but deserving significant attention is the semantic interpretation of the used contract elements. The use of general business and domain-specific ontologies is the current approach to address this problem. The elaboration of concrete, domain-specific ontologies that can be used by e-contracting parties is still in its initial stages. An example for an industry specific standardization effort in this direction is [RN].

References

[Aal03] Aalst, W. M. P. van der, Kumar, A. (2003). "XML Based Schema Definition for Support of Inter-organizational Workflow." *Information Systems Research*, Vol. 14, No.1, pp. 23-46.

[Ang01] Angelov, S., Grefen, P. (2001). "A Framework for the Analysis of B2B Electronic Contracting Support." In *Multidisciplinary Perspectives on Electronic*

Commerce - Proceedings of the 4th Edispuut Conference, Amsterdam, The Netherlands, pp. 6-20.

[Ang02] Angelov, S., Grefen, P. (2002). "Support for B2B E-contracting – The Process Perspective." In Knowledge and Technology Integration in Production and Services – Proceedings of the 5th International Conference on Information Technology for Balanced Automation Systems in Manufacturing and Services, Cancun, Mexico. pp. 87-96.

[Ang03] Angelov, S., Grefen, P. (2003). "The 4W framework for B2B e-contracting." Int. J. Networking and Virtual Organisation, Vol. 2, No.1, pp.78-97.

[Ang04] Angelov, S., Grefen, P. (2004). "The Business Case for B2B E-Contracting." Sixth International Conference on Electronic Commerce ICEC04: Towards a New Services Landscape; Delft, The Netherlands. pp. 31-40.

[Ang05] Angelov, S., Till, S., Grefen, P. (2005). "Dynamic and Secure B2B E-contract Update Management." Proceedings of the 6th ACM Conference on Electronic Commerce (EC'05), Vancouver, Canada.

[BPEL] The Business Process Execution Language for Web Services Specification. (2003). Version 1.1.

[Chi03] Chiu, D., K., W., Cheung, S., C., Till, S. (2003). "A Three-Layer Architecture for E-Contract Enforcement in an E-Service Environment." Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03), Big Island, Hawaii.

[Cro99] CrossFlow Consortium. (1999). "Contract Model." CrossFlow deliverable: D4.b, La Gaude.

[Cross] CrossFlow project <http://www.crossflow.org> .

[Dan98] Dan, A., Dias, D., Nguyen, T., Sachs, M., Shaikh, H., King, R., Duri, S. (1998). "The Coyote Project: Framework for Multi-party E-Commerce." Proceedings of the 7th Delos Workshop on Electronic Commerce, Crete, Greece.

[Das97] Daskalopulu, A., Sergot, M. (1997). "The Representation of Legal Contracts." AI and Society, 11 (1 & 2), pp. 6 - 17.

[ebX01] ebXML. (2001). "ebXML Business process specification schema, v1.01 6." ebXML.

[Gis00] Gisler, M., Stanoevska-Slabeva, K., Greunz, G. (2000). "Legal Aspects of Electronic Contracts." Infrastructures for Dynamic Business-to-Business Service Outsourcing (IDSO'00), Stockholm, 2000.

[Goo00] Goodchild, A., Herring, C., Milosevic, Z. (2000). "Business contracts for B2B", Proceedings of the CAISE00 Workshop on Infrastructure for Dynamic

Business-to-Business Service Outsourcing, Stockholm, Sweden.

[Gri98] Griffel, F., Boger, M., Weinreich, H., Lamersdorf, W., Merz, M. (1998). "Electronic Contracting with COSMOS - How to Establish, Negotiate and Execute Electronic Contracts on the Internet." Proceedings of the 2nd Int. Enterprise Distributed Object Computing Workshop (EDOC '98), San Diego, CA, USA.

[Gro99] Grosz, B., Labrou, Y., Chan, H. (1999). "A Declarative Approach to Business Rules in Contracts: Courteous Logic Programs in XML." Proceedings of the 1st ACM Conference on Electronic Commerce (EC'99). Denver, CO, USA. ACM Press.

[Gro04] Grosz, B., Poon, T. (2004). "SweetDeal: Representing Agent Contracts with Exceptions Using Semantic Web Rules, Ontologies, and Process Descriptions". International Journal of Electronic Commerce, Volume 8, Number 4, Summer 2004, pp. 61.

[Koe00] Koetsier, M., Grefen, P., Vonk, J. (2000). "Contracts for Cross-Organizational Workflow Management." Proceedings of the 1st International Conference on Electronic Commerce and Web Technologies, London, UK, pp. 110-121.

[Lin04] Linington, P., Milosevic, Z., Cole, J., Gibson, S., Kulkarni, S., Neal, S. (2004). "A unified behavioural model and a contract language for extended enterprise". Special issue of Data & Knowledge Engineering: Contract-driven coordination and collaboration in the internet context. Volume 51, Issue 1. pp. 5 – 29.

[Nea03] Neal, S., Cole, J., Linington, P., Milosevic, Z., Gibson, S., Kulkarni, S. (2003). "Identifying requirements for Business Contract Language: a Monitoring Perspective". Proceedings of the 7th International Enterprise Distributed Object Computing Conference (EDOC'03), Brisbane, Queensland, Australia, pp. 50.

[OWL] OWL Web Ontology Language <http://www.w3.org/TR/owl-features/>.

[Per03] Perrin, O., Godart, C. (2003). "A Contract Model to Deploy and Control Cooperative Processes." Proceedings of the 4th International Workshop Technologies for E-Services (TES 2003), Berlin, Germany, Lecture Notes in Computer Science 2819 Springer 2003, pp. 78-90.

[Rei89] Reinecke, J., Dessler, G., Schoell, W. (1989). Introduction to Business – A Contemporary View, Allyn and Bacon.

[Ros01] Ross, R., Lam, G. (2001). "RuleSpeak Sentence Templates: Developing Rules Statements Using Sentence Patterns." Business Rule Solutions, available at www.BRCommunity.com.

[RN] RosettaNet standards. <http://www.rosettanet.org/>

[RuleML] The Rule Markup Language Initiative. <http://www.ruleml.org/>

[Wag02] Wagner, G. (2002). "How to Design a General Rule Markup Language?". In Proceedings of the Workshop XML Technologies for the Semantic Web (XSW 2002), HU Berlin, Institut für Informatik, Lecture Notes in Informatics, Gesellschaft f. Informatik.

[Wag03] Wagner, G., Tabet, S., Boley, H. (2003) "MOF-RuleML: The Abstract Syntax of RuleML as a MOF Model" discussion paper, OMG meeting October 2003, Boston.

[Wie02] Wieringa, R. (2002). Design Methods for Reactive Systems: Yourdon, StateMate, and the UML, Elsevier Science & Technology Books.

[WMC] WMC. (1999). "Workflow Management Coalition Interface 1: process definition interchange process model v1.1" The Workflow Management Coalition.

[XCBL] XML Common Business Library (xCBL), <http://www.xcbl.org> .

[XTC] Execution of Transactional Contracted Electronic Services (XTraConServe), NWO project No: 612.063.305.

Appendix A

E-Contracting Markup Language (ECML)

Version: 0.02

Comments: ECML has been designed with an illustrative to this report purpose. ECML is an XML based language that supports the requirements on e-contract languages listed in the report. In this appendix, we provide the XML schema of ECML. Throughout the schema, in textual commentaries, we indicate the e-contract language requirements that a certain section satisfies. The ECML schema provided next can directly be used for definition of e-contracts. All textual comments are enclosed in “<!--“ ... “--> “ and do not influence XML parsers (these comments can be deleted, resulting in the pure ECML schema).

The schema is divided into four parts. The first three parts correspond to the three basic classes of e-contract language constructs (as discussed in Section 3.1). The fourth part defines the structure of e-contracts provided by ECML (as discussed in Section 3.2.3).

ECML schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.4 U by Samuil Angelov (Technische Universiteit Eindhoven) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
<!-- PART 1 -->
<!-- Next, the definition of the data types required in an e-contract language (see Section 4.1, Requirement 1,
Requirement 14 and Requirement 15) are provided. We use the built in the XML schema data types and
extend them with the required additional attributes for a data item. The common attributes for all data items
are listed in the group of attributes "Common_var_attributes". We start with the definition of the standard data
types (see Section 4.1.1) and continue with the definition of the special data types (see Section 4.1.2). At the
end of this part of the ECML schema, we list three predefined set types (i.e., "State_type", "Constraint_type",
"Currency_type"). These constant sets are used as the data type of attributes of contract elements that are
defined later on in the language schema. -->
<!-- definition of standard Variable TYPES -->
  <xs:attributeGroup name="Common_var_attributes">
    <xs:attribute name="Tag_name" type="xs:string" use="required"/>
    <xs:attribute name="Var_ID" type="xs:ID" use="required"/>
    <xs:attribute name="Owner" type="xs:IDREFS" use="optional"/>
    <xs:attribute name="Changeable" type="xs:boolean" use="required"/>
    <xs:attribute name="Properties" type="xs:IDREFS" use="optional"/>
    <xs:attribute name="Enabled" type="State_type" use="required"/>
    <xs:attribute name="Rules_for_change" type="xs:IDREFS" use="optional"/>
  </xs:attributeGroup>
<!-- "Tag name" attribute is used to satisfy Requirement 8. The value of this attribute is a name from an
agreed ontology.
  "Var_ID" attribute is used to satisfy Requirement 4. It contains the unique identifier of the data item.
  "Owner" attribute is used to satisfy Requirement 21 and is related to Requirement 7 (see free updates). It
contains a reference to an owner of the data item who is allowed to update the value of the data item.
  "Changeable" attribute is used to satisfy Requirement 7. It indicates if the value of the data element can
be changed.
  "Properties" attribute is used because of Requirement 18. It indicates the properties a data element has.
  "Enabled" attribute indicates if the data item is currently enabled. The value of this attribute is from the
defined State_type (used to satisfy Requirement 20).
  "Rules_for_change" attribute is used to indicate the rules in the contract that must evaluate to true in
order a change to take place. It is optional and is used solely for improving the efficiency of the monitoring
systems. -->
  <xs:complexType name="String_type">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attributeGroup ref="Common_var_attributes"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
  <xs:complexType name="Real_type">
    <xs:simpleContent>
      <xs:extension base="xs:decimal">
```

```

        <xs:attributeGroup ref="Common_var_attributes"/>
    </xs:extension>
</xs:simpleContent>
</xs:complexType>
<xs:complexType name="Integer_type">
    <xs:simpleContent>
        <xs:extension base="xs:integer">
            <xs:attributeGroup ref="Common_var_attributes"/>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<xs:complexType name="Boolean_type">
    <xs:simpleContent>
        <xs:extension base="xs:boolean">
            <xs:attributeGroup ref="Common_var_attributes"/>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<!-- Sample definition of data items from the List data type. Additional list types can be defined by designers in
the schema or in the e-contract itself. -->
<xs:simpleType name="List_of_strings_type">
    <xs:list itemType="xs:string"/>
</xs:simpleType>
<xs:complexType name="List_of_events_type">
    <xs:sequence>
        <xs:element name="Event" type="Event_type" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType >
<!-- For the definition of data items from the Record data type, contract designers can use the build in XML
Schema Complex Type element. -->
<!-- Definition of special data types -->
<xs:complexType name="Date_type">
    <xs:simpleContent>
        <xs:extension base="xs:date">
            <xs:attributeGroup ref="Common_var_attributes"/>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<xs:complexType name="Time_type">
    <xs:simpleContent>
        <xs:extension base="xs:time">
            <xs:attributeGroup ref="Common_var_attributes"/>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<xs:complexType name="Money_type">
    <xs:simpleContent>
        <xs:extension base="xs:decimal">
            <xs:attributeGroup ref="Common_var_attributes"/>
            <xs:attribute name="Currency" type="Currency_type" use="required"/>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<xs:complexType name="Event_type">
    <xs:simpleContent>
        <xs:extension base="xs:boolean">
            <xs:attributeGroup ref="Common_var_attributes"/>
            <xs:attribute name="Time_of_occurrence" type="xs:time" use="optional"/>
            <xs:attribute name="Date_of_occurrence" type="xs:date" use="optional"/>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<xs:complexType name="External_resource_reference_type">
    <xs:simpleContent>
        <xs:extension base="xs:anyURI">
            <xs:attributeGroup ref="Common_var_attributes"/>
            <xs:attribute name="Resource_state" type="List_of_resource_types" use="optional"/>
            <xs:attribute name="Is_legally_binding" type="xs:boolean" use="required"/>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>

```

```

    </xs:simpleContent>
  </xs:complexType>
<!-- Snippets allow inclusion of predefined contract parts (see Requirement 5) -->
<xs:complexType name="Snippet_type">
  <xs:sequence>
    <xs:any minOccurs="1" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="Snippet_ID" type="xs:ID" use="required"/>
</xs:complexType>
<!-- Definition of rule/process state type -->
<xs:simpleType name="State_type">
  <xs:restriction base="xs:string">
    <xs:pattern value="enabled | disabled"/>
  </xs:restriction>
</xs:simpleType>
<!-- Definition of deontic operators -->
<xs:simpleType name="Constraint_type">
  <xs:restriction base="xs:string">
    <xs:enumeration value="must"/>
    <xs:enumeration value="may"/>
    <xs:enumeration value="may_not"/>
  </xs:restriction>
</xs:simpleType>
<!-- Definition of static constraints deontic operators -->
<xs:simpleType name="Static_constraint_type">
  <xs:restriction base="xs:string">
    <xs:enumeration value="must_observe"/>
    <xs:enumeration value="may_break"/>
  </xs:restriction>
</xs:simpleType>
<!-- Definition of currency types -->
<xs:simpleType name="Currency_type">
  <xs:restriction base="xs:string">
    <xs:enumeration value="USD"/>
    <xs:enumeration value="EUR"/>
    <xs:enumeration value="GBP"/>
    <xs:enumeration value="CAD"/>
    <xs:enumeration value="BGN"/>
  </xs:restriction>
</xs:simpleType>
<!-- Definition of resource state types -->
<xs:simpleType name="List_of_resource_types">
  <xs:list itemType="Resource_state_type"/>
</xs:simpleType>
<xs:simpleType name="Resource_state_type">
  <xs:restriction base="xs:string">
    <xs:enumeration value="available"/>
    <xs:enumeration value="unavailable"/>
    <xs:enumeration value="changed"/>
  </xs:restriction>
</xs:simpleType>
<!-- PART 2 -->
<!-- In the second part of the ECML schema, we provide the constructs required for the definition of the
integrity, derivation, and reaction contract rules (see Section 4.2, Requirement 17, Requirement 18, and
Requirement 19). The common for all three rule types attributes are extracted in the attribute group
"Common_rule_attributes". A sample set of operators that can be used in expressions is provided with an
illustrative purpose (e.g., the "Unary_operator" type). The complete set of operators must additionally be
defined (for example the defined Requirement 23). -->
<!-- Definition of common rule attributes. In addition to the attributes "Tag_name", "Rule_ID", "Changeable",
Enabled" explained in the data item definition section, the following attributes are defined:
The "Overrides" attribute indicates which other rule are overridden by this rule if they fire together (used to
satisfy Requirement 16). -->
<xs:attributeGroup name="Common_rule_attributes">
  <xs:attribute name="Tag_name" type="xs:string" use="required"/>
  <xs:attribute name="Rule_ID" type="xs:ID" use="required"/>
  <xs:attribute name="Enabled" type="State_type" use="required"/>
  <xs:attribute name="Changeable" type="xs:boolean" use="required"/>
  <xs:attribute name="Overrides" type="xs:IDREFS" use="optional"/>
</xs:attributeGroup>

```

```

<!-- Definition of some operators used in expressions in rules. -->
<xs:simpleType name="Unary_operator">
  <xs:restriction base="xs:string">
    <xs:enumeration value="plus"/>
    <xs:enumeration value="minus"/>
    <xs:enumeration value="sqrt"/>
    <xs:enumeration value="percent"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="Unary_boolean_operator">
  <xs:restriction base="xs:string">
    <xs:enumeration value="not"/>
  </xs:restriction>
</xs:simpleType>
<!-- Next, we start with the definition of Integrity rules (distinguishing the two classes of integrity rules, namely
the "State_constraint_rule_type" and the "Dynamic_constraint_rule_type" as discussed in Section 4.2.1). The
State_constraint_rule_type construct is a simple Boolean expression. The "Assigned_to" attribute indicates the
party (if any) to which the constraint is assigned. The "Type" attribute indicates if the rule obligates the
assigned party to observe the rule or allows it to break the rule (related to Requirement 21). The
Dynamic_constraint_rule_type construct consists of four elements. If the boolean expression evaluates to true
the rule is in force and the referenced element "Element_ref" (or attribute of the element
"Element_attribute_ref") may/may not (depending on the value of the value of the type attribute) receive a new
value indicating its new state. The "Restricted-nonrestricted_values" element contains a set of
allowed/disallowed values. -->
<!-- Definition of expressions in rule types. Expressions in rules are presented as strings that are to be parsed
by the contract interpretation software in order to obtain an expression tree. The expression strings can be
mixed with references to earlier defined variables. -->
<xs:complexType name="Expression" mixed="true">
  <xs:sequence>
    <xs:element name="Expression_variable_ref" type="xs:IDREF" minOccurs="0"
maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<!-- Definition of the possible rule types and their sub-constructs -->
<xs:complexType name="State_constraint_rule_type">
  <xs:sequence maxOccurs="1">
    <xs:element name="Rule_conditions" type="Expression"/>
  </xs:sequence>
  <xs:attributeGroup ref="Common_rule_attributes"/>
  <xs:attribute name="Assigned_to" type="xs:IDREF" use="optional"/>
  <xs:attribute name="Type" type="Static_constraint_type" use="optional"/>
</xs:complexType>
<xs:complexType name="Dynamic_constraint_rule_type">
  <xs:sequence>
    <xs:element name="Rule_conditions" type="Expression"/>
    <xs:element name="Element_ref" type="xs:IDREF"/>
    <xs:element name="Element_attribute_ref" type="xs:IDREF" minOccurs="0"/>
    <xs:element name="Restricted-nonrestricted_states" type="List_of_strings_type"/>
  </xs:sequence>
  <xs:attribute name="Type" type="Constraint_type"/>
  <xs:attributeGroup ref="Common_rule_attributes"/>
</xs:complexType>
<!-- Next, we define the Derivation rule constructs (distinguishing the two types of derivation rules, namely
"Computational_derivation_rule_type" and "Linguistic_derivation_rule_type" as discussed in Section 4.2.2).
The "Derived_variable_ref" attribute contains a reference to the data items the value of which is derived with
this rule. "Derivation_expression" contains a Boolean expression that when evaluating to true fires the rule.
In "Linguistic_derivation_rule_type", the "Property_extended_variable_ref" contains a reference to the data
item that is given a property when the rule fires. "Property_to_be_added_ref" contains a reference to a data
item that contains the property to be added. In this approach, we require the properties to be defined as data
items in the e-contract, as in this way they can be referenced by many components and will be from a clearly
defined data type. However, as already indicated in Section 4.2.2, it is possible linguistic derivation rules to be
completely replaced by copier reaction rules that simply state the new value to be given to the data item (in a
string format). -->
<xs:complexType name="Computational_derivation_rule_type">
  <xs:sequence>
    <xs:element name="Derived_variable_ref" type="xs:IDREFS"/>
    <xs:element name="Derivation_expression" type="Expression"/>
  </xs:sequence>
  <xs:attributeGroup ref="Common_rule_attributes"/>

```

```

</xs:complexType>
<xs:complexType name="Linguistic_derivation_rule_type">
  <xs:sequence>
    <xs:element name="Derivation_expression" type="Expression"/>
    <xs:element name="Property_extended_variable_ref" type="xs:IDREF"/>
    <xs:element name="Property_to_be_added_ref" type="xs:IDREF"/>
  </xs:sequence>
  <xs:attributeGroup ref="Common_rule_attributes"/>
</xs:complexType>
<!-- Next, the ECML constructs for the support of Reaction rules follow. As discussed in Section 4.2.3, a
reaction rule can be from one of the three subtypes, i.e., enabler, executive, and copiers. To support each of
the three types, we provide three sub-constructs for each of them. Each reaction rule fires when its
"Rule_conditions" evaluates to true.
The "Enabler_action_type" has two elements. The "Target_element" contains a reference to the element the
state of which will be changed. "New_state" indicates the new state that must be assigned to the element
(enabled or disabled). The "Type" attribute indicates the deontic assignment to a party (defined in the
"Assigned_to" attribute), i.e., if a party that is owner of the rule is obliged/allowed/forbidden to perform the
update of the status of the element (related to Requirement 21).
The "Executive_action_type" sub-construct indicates through the "Targets" element the execution of which
process specifications must be/may be/cannot be started when the rule fires (related to Requirement 25). The
"Type" attribute indicates the deontic assignment to a party, i.e., if a party (defined in the "Assigned_to"
attribute) is obliged/allowed/forbidden to start the execution of the process (related to Requirement 21).
The "Copier_action_type" sub-construct indicates which element (the "Target_element" element) or its attribute
(the "Target_attribute" attribute) is given a new value (the "New_value" attribute). Again the "Type" attribute
indicates the deontic assignment on the owner of the rule. -->
  <xs:complexType name="Enabler_action_type">
    <xs:sequence>
      <xs:element name="Target_element" type="xs:IDREF" minOccurs="1"/>
      <xs:element name="New_state" type="State_type" minOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="Type" type="Constraint_type"/>
    <xs:attribute name="Assigned_to" type="xs:IDREF" use="optional"/>
  </xs:complexType>
  <xs:complexType name="Executive_action_type">
    <xs:sequence>
      <xs:element name="Targets" type="xs:IDREFS"/>
    </xs:sequence>
    <xs:attribute name="Type" type="Constraint_type"/>
    <xs:attribute name="Assigned_to" type="xs:IDREF" use="optional"/>
  </xs:complexType>
  <xs:complexType name="Copier_action_type">
    <xs:sequence>
      <xs:element name="Target_element" type="xs:IDREF"/>
      <xs:element name="Target_attribute" type="xs:string" minOccurs="0"/>
      <xs:element name="New_value" type="List_of_strings_type" minOccurs="0"/> <!-- besides the
assignment of one value, a party can be allowed to set one of a set of values (for example in the control of a
proces status -->
    </xs:sequence>
    <xs:attribute name="Type" type="Constraint_type"/>
    <xs:attribute name="Assigned_to" type="xs:IDREF" use="optional"/>
  </xs:complexType>
  <xs:complexType name="Reaction_rule_type">
    <xs:sequence>
      <xs:element name="Rule_conditions" type="Expression"/>
      <xs:choice>
        <xs:element name="Enabler_action" type="Enabler_action_type"/>
        <xs:element name="Executive_action" type="Executive_action_type"/>
        <xs:element name="Copier_action" type="Copier_action_type"/>
      </xs:choice>
    </xs:sequence>
    <xs:attributeGroup ref="Common_rule_attributes"/>
  </xs:complexType>
<!-- To satisfy Requirement 22, next we provide the construct for free-text rules (see Section 4.2.5 -->
  <xs:complexType name="Free_text_rule_type">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attributeGroup ref="Common_rule_attributes"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

```

```

    </xs:complexType>
<!-- PART 3 -->
<!-- For the definition of process specification constructs in ECML we use as a foundation XRL
(eXchangeable Routing Language). XRL is an instance-based workflow language that uses XML for the
representation of process definitions and Petri nets for its semantics. A few modifications have been applied
on XRL in order to satisfy the additional requirements as discussed in Section 4.3. These extensions are
marked with commentaries. XRL is a prescriptive language which defines explicitly the control flow between
activities.
XRL was chosen as process specification language for several reasons. First, its XML representation and
short schema definition allow easy integration and extension to suit the ECML goals. Second, the underlying
Petri Net semantics in XRL allows the Petri net representation of an XRL process specification to be analyzed
using state-of-the-art analysis techniques and tools. Finally, the experience and background of using XRL in
the IS group at the Technical University of Eindhoven allowed easier implementation and maintenance in
ECML. More information about XRL and its original DTD/schema can be found at:
http://tmitwww.tn.tue.nl/staff/anorta/XRL/xrlHome.html -->
<xs:group name="Common_elements">
  <xs:choice>
    <xs:element name="task" type="taskType"/>
    <xs:element name="sequence" type="sequenceType"/>
    <xs:element name="any_sequence" type="any_sequenceType"/>
    <xs:element name="choice" type="choiceType"/>
    <xs:element name="condition" type="conditionType"/>
    <xs:element name="parallel_sync" type="parallel_syncType"/>
    <xs:element name="parallel_no_sync" type="parallel_no_syncType"/>
    <xs:element name="parallel_part_sync" type="parallel_part_syncType"/>
    <xs:element name="parallel_part_sync_cancel" type="parallel_part_sync_cancelType"/>
    <xs:element name="restricted_parallel_sync" type="restricted_parallel_syncType"/>
    <xs:element name="wait_all" type="wait_allType"/>
    <xs:element name="wait_any" type="wait_anyType"/>
    <xs:element name="while_do" type="while_doType"/>
    <xs:element name="terminate" type="terminateType"/>
  </xs:choice>
</xs:group>
<xs:complexType name="any_sequenceType">
  <xs:sequence>
    <xs:group ref="Common_elements" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="choiceType">
  <xs:sequence>
    <xs:group ref="Common_elements" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="conditionType">
  <xs:choice minOccurs="0" maxOccurs="unbounded">
    <xs:element name="true_branch" type="true_branchType"/>
    <xs:element name="false_branch" type="false_branchType"/>
  </xs:choice>
  <xs:attribute name="condition" type="xs:string" use="required"/>
  <xs:attribute name="description" type="xs:string"/>
</xs:complexType>
<xs:complexType name="false_branchType">
  <xs:sequence>
    <xs:group ref="Common_elements"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="parallel_no_syncType">
  <xs:sequence>
    <xs:group ref="Common_elements" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="parallel_part_syncType">
  <xs:sequence>
    <xs:group ref="Common_elements" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="number" type="xs:NMTOKEN" use="required"/>
</xs:complexType>
<xs:complexType name="parallel_part_sync_cancelType">
  <xs:sequence>

```

```

        <xs:group ref="Common_elements" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="number" type="xs:NMTOKEN" use="required"/>
</xs:complexType>
<xs:complexType name="parallel_syncType">
    <xs:sequence>
        <xs:group ref="Common_elements" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="restricted_parallel_syncType">
    <xs:sequence>
        <xs:group ref="Common_elements" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="sequenceType">
    <xs:sequence>
        <xs:group ref="Common_elements" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="taskType"> <!-- as required in Requirement 24 -->
    <xs:sequence>
        <xs:element name="event" type="Event_type" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="name" type="xs:ID" use="required"/>
    <xs:attribute name="Tag_name" type="xs:string" use="required"/> <!-- extension -->
    <xs:attribute name="address" type="xs:string" use="required"/>
    <xs:attribute name="Executor" type="xs:IDREF"/> <!-- extension; required to indicate which party has
to perform the activity. It is required as in the executive reaction rule that triggers the process execution one
party can be assigned the responsibility to start the process execution. However, a specific activity can be
agreed to be executed by another party. -->
    <xs:attribute name="doc_read" type="xs:NMTOKENS"/>
    <xs:attribute name="doc_update" type="xs:NMTOKENS"/>
    <xs:attribute name="doc_create" type="xs:NMTOKENS"/>
    <xs:attribute name="result" type="xs:string"/>
    <xs:attribute name="start_time" type="xs:NMTOKENS"/>
    <xs:attribute name="end_time" type="xs:NMTOKENS"/>
    <xs:attribute name="notify" type="xs:string"/>
    <xs:attribute name="Enabled" type="State_type"/> <!-- extension -->
    <xs:attribute name="Applied_rules" type="xs:IDREFS"/> <!-- extension; required to indicate the
applied static and dynamic constraints on the starting of execution of the activity (Requirement 25) -->
</xs:complexType>
<xs:complexType name="terminateType"/>
<xs:complexType name="timeoutType">
    <xs:sequence>
        <xs:group ref="Common_elements" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="time" type="xs:string" use="required"/>
    <xs:attribute name="type" default="absolute">
        <xs:simpleType>
            <xs:restriction base="xs:NMTOKEN">
                <xs:enumeration value="relative"/>
                <xs:enumeration value="s_relative"/>
                <xs:enumeration value="absolute"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:attribute>
</xs:complexType>
<xs:complexType name="true_branchType">
    <xs:sequence>
        <xs:group ref="Common_elements"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="wait_allType">
    <xs:choice maxOccurs="unbounded">
        <xs:element name="event_ref" type="xs:IDREF"/>
        <xs:element name="timeout" type="timeoutType"/>
    </xs:choice>
</xs:complexType>
<xs:complexType name="wait_anyType">

```

```

    <xs:choice maxOccurs="unbounded">
      <xs:element name="event_ref" type="xs:IDREF"/>
      <xs:element name="timeout" type="timeoutType"/>
    </xs:choice>
  </xs:complexType>
  <xs:complexType name="while_doType">
    <xs:sequence>
      <xs:group ref="Common_elements"/>
    </xs:sequence>
    <xs:attribute name="condition" type="xs:string" use="required"/>
  </xs:complexType>
  <xs:complexType name="Route"> <!-- Control over the process and monitoring rights and obligations are
specified through deontic assignments and reaction rules (executive for monitoring and copier reaction rules
for control rights) -->
    <xs:sequence>
      <xs:group ref="Common_elements"/>
    </xs:sequence>
    <xs:attribute name="Tag_name" type="xs:string" use="required"/>
    <xs:attribute name="Process_ID" type="xs:ID" use="required"/> <!-- extension -->
    <xs:attribute name="Enabled" type="State_type"/> <!-- extension -->
    <xs:attribute name="Created_by" type="xs:string"/>
    <xs:attribute name="Date" type="xs:string"/>
  </xs:complexType>
<!-- PART 4      Syntax      -->
<!-- In this part, we provide the syntax definitions of ECML, i.e., the structure of ECML (see Section 3.2.3,
Requirement 9). First, we provide three basic structures, i.e., the Variable definition section, Rule definition
section, and Process definition section. In each of these sections can respectively be defined data constructs,
rule constructs, and process constructs. -->
  <xs:complexType name="Variables_def_section">
    <xs:sequence minOccurs="1" maxOccurs="unbounded">
      <xs:choice>
        <xs:element name="String_var" type="String_type"/>
        <xs:element name="Real_var" type="Real_type"/>
        <xs:element name="Integer_var" type="Integer_type"/>
        <xs:element name="Boolean_var" type="Boolean_type"/>
        <xs:element name="Date_var" type="Date_type"/>
        <xs:element name="Time_var" type="Time_type"/>
        <xs:element name="Event_var" type="Event_type"/>
        <xs:element name="Money_var" type="Money_type"/>
        <xs:element name="External_resource_reference_var"
type="External_resource_reference_type"/>
        <xs:element name="List_of_events_var" type="List_of_events_type"/>
        <xs:element name="List_of_strings_var" type="List_of_strings_type"/>
        <xs:any/> <!-- Required to allow the definition of user-defined complex types and lists -->
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="Rule_def_section">
    <xs:sequence minOccurs="1" maxOccurs="unbounded">
      <xs:choice>
        <xs:element name="State_constraint_Rule" type="State_constraint_rule_type" minOccurs="0"
maxOccurs="unbounded"/>
        <xs:element name="Dynamic_constraint_Rule" type="Dynamic_constraint_rule_type"
minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="Computational_derivation_Rule"
type="Computational_derivation_rule_type" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="Linguistic_derivation_Rule" type="Linguistic_derivation_rule_type"
minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="Reaction_rule" type="Reaction_rule_type" minOccurs="0"
maxOccurs="unbounded"/>
        <xs:element name="Free_text_rule" type="Free_text_rule_type" minOccurs="0"
maxOccurs="unbounded"/>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="Process_def_section">
    <xs:sequence>
      <xs:element name="Process" type="Route" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>

```

```

</xs:complexType>
<!-- Next, we define the possible combinations of data constructs, rule constructs and process constructs that
can be used in an e-contract sub-structure as discussed in Section 3.2.3. -->
<xs:complexType name="Only_vars_section">
  <xs:sequence>
    <xs:element name="Var_section" type="Variables_def_section"/>
    <xs:element name="Snippet_section" type="Snippet_type" minOccurs="0"/> <!-- required to
support the inclusion of externally defined data items (see Requirement 5) -->
  </xs:sequence>
</xs:complexType>
<xs:complexType name="Vars_and_rules_section">
  <xs:sequence>
    <xs:element name="Var_section" type="Variables_def_section" minOccurs="0"/>
    <xs:element name="Rule_section" type="Rule_def_section" minOccurs="0"/>
    <xs:element name="Snippet_section" type="Snippet_type" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="Vars_and_processes_section">
  <xs:sequence>
    <xs:element name="Var_section" type="Variables_def_section" minOccurs="0"/>
    <xs:element name="Process_section" type="Process_def_section" minOccurs="0"/>
    <xs:element name="Snippet_section" type="Snippet_type" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="All_section">
  <xs:sequence>
    <xs:element name="Var_section" type="Variables_def_section" minOccurs="0"/>
    <xs:element name="Rule_section" type="Rule_def_section" minOccurs="0"/>
    <xs:element name="Process_section" type="Process_def_section" minOccurs="0"/>
    <xs:element name="Snippet_section" type="Snippet_type" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<!-- definition of contract sub-structures of the 4W structure-->
<xs:complexType name="Company_Info">
  <xs:sequence>
    <xs:element name="Company_data" type="Only_vars_section"/>
    <xs:element name="Company_contact_data" type="Only_vars_section"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="Value_types">
  <xs:sequence>
    <xs:element name="Product" type="Only_vars_section"/>
    <xs:element name="Service" type="Vars_and_processes_section"/>
    <xs:element name="Fincial_reward" type="Vars_and_processes_section"/>
  </xs:sequence>
</xs:complexType>
<!-- Finally, we define the root element of the schema and the complete contract structure -->
<xs:element name="Contract">
  <xs:complexType>
    <xs:sequence>
      <!--WHO Section contains the description of the parties and the possible mediators (see
Requirement 10) -->
      <xs:element name="Party" type="Company_Info" minOccurs="2" maxOccurs="unbounded"/>
      <xs:element name="Mediator" type="Company_Info" minOccurs="0"
maxOccurs="unbounded"/>
      <!--WHERE Section contains the description of the context of the agreement (see
Requirement 11) -->
      <xs:element name="Business_context_provisions" type="All_section" minOccurs="0"
maxOccurs="unbounded"/>
      <xs:element name="Legal_context_provisions" type="All_section" minOccurs="0"
maxOccurs="unbounded"/>
      <xs:element name="Other_context_provisions" type="All_section" minOccurs="0"
maxOccurs="unbounded"/>
      <!--WHAT Section contains the description of the exchanged goods/services and the
conditions for the exchange (see Requirement 12) -->
      <xs:element name="Exchanged_value" type="Value_types" minOccurs="2"
maxOccurs="unbounded"/>
      <xs:element name="Exchange_provisions" type="All_section" minOccurs="2"
maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

```

```

        <!--HOW Section contains the description of the means for the contract enactment (see
Requirement 13) -->
        <xs:element name="Contracting_process" type="All_section" minOccurs="0"
maxOccurs="unbounded"/>
        <xs:element name="Standards" type="Vars_and_rules_section" minOccurs="0"
maxOccurs="unbounded"/><!-- in this section the <xs:element name="Agreed_ontology"
type="External_resource_reference_type" minOccurs="1" maxOccurs="unbounded"/> can be used to satisfy
Requirement 8 and to define the used ontologies -->
        <xs:element name="Communication" type="All_section" minOccurs="0"
maxOccurs="unbounded"/> <!-- to satisfy Requirement 6 -->
        </xs:sequence>
        <xs:attribute name="Contract_ID" type="xs:string"/>
    </xs:complexType>
</xs:element>
</xs:schema>

```

Appendix B

SAMPLE CONTRACT ON CONSULTING SERVICES

(used with permission from Barbara Davis)

1 This sample Agreement for Services, or Contract, is for a project that began with a
2 Request for Proposal where the Great Scott Arts Association, a new organization in
3 Mosquito Falls, MN, was seeking a consultant or consultants to assist in its initial set-
4 up and in the analysis of possible administrative and performing space.

5

6 1. Terrific Consulting (hereafter called Contractor) agrees to provide the following
7 products and services to the Iceberg County Art Center (hereafter called Client):

8

9 A. Information gathering: Contractor will review the following information compiled
10 by Client:

11

1) information on attendance at past events

12

2) samples of past promotional pieces and any recent press coverage

13

3) samples of past membership flyers

14

4) financial information from the past three or four years

15

16 Contractor will also inquire into the availability of any recent market studies on the
17 Iceberg area done by the city government or other groups working on the Iceberg's
18 economic development.

19

20 Contractor will also confer by phone with the part-time coordinator and two or three
21 Client board members to get their view of the issues facing Client.

22

23 B. Member survey: Contractor will design a one-page membership survey. Client will
24 be responsible for duplicating the survey, sending it out to the membership and
25 tallying the responses.

26

27 C. Contractor will design a "community leader interview" format. Client's board
28 members will call on community leaders and conduct interviews. Contractor will
29 summarize the finding and merge them with the survey responses, giving Client a
30 written report on how it is perceived by members and community, and what people
31 think it should be doing.

32

33 D. Contractor will travel to Iceberg for a one-day stay to meet for half a day with
34 Client's board to discuss:

35

1). the survey findings, Client's audiences or "publics", Client's "position" in the
36 community, the "messages" that Client wants to communicate to people, past
37 promotional efforts and possible changes to make in the future.

38

2). the management training needs of Client's staff, board and committee chairs.

39

Contractor will present a format to use in writing job descriptions and teach Client
40 how to use it. The group will also develop an organizational chart for Client,
41 identifying the various committees, board officers, committee chairs and others with
42 particular management responsibilities.

43

3). Client's fundraising needs and opportunities. Contractor will explore how much
44 Client wants to raise, and what Client is willing to do to raise it, including Client's

45 membership program, grant opportunities and the possibility of more individual
46 fundraising.

47

48 E. Before leaving, Contractor will give some "homework" assignments to Client.
49 They would likely include preparation of some job descriptions and a clear annual
50 fundraising goal.

51

52 F. Contractor will write:

53 1) a regular promotional schedule based on the goals identified at the meeting above

54 2) a review of Client's job descriptions with suggested revisions

55 3) recommended management training opportunities

56 4) if feasible, an annual calendar of management tasks

57 5) recommendations for raising more contributions

58

59 This work will be completed no later than December 31, 1999, and will be conducted
60 by Contractors' agent, Edmund E. Expert.

61

62 It is understood that circumstances arising during the consulting project may require
63 the activities described above to be replaced with other activities of an equivalent
64 value. Such changes will be based on mutual agreement of both parties, which may be
65 recorded as an addendum to this agreement, or as a letter from one party to the other.

66

67 2. Client agrees to:

68 A. participate as requested in consulting activities. This includes calling meetings,
69 providing meeting sites and amenities, and providing information requested by
70 Contractor. Client's entire board will be involved in this process, not just the
71 coordinator. Client will also be asked to duplicate, distribute and tally the results of a
72 membership survey, to conduct communicate leader interviews and to complete the
73 homework assignments.

74

75 B. pay Contractor a fee not to exceed \$1,405,000 plus expenses. Expenses to be billed
76 include travel (\$.25 per mile for auto travel), lodging and meals while in Iceberg,
77 long-distance phone calls, and any copying and mailing services, outside of normal
78 communication with Client. Lodging and meal expenses will be documented with
79 receipts.

80

81 The payments will be made in three installments: \$400,000 at the signing of this
82 contract; \$400,000 upon completion of the Iceberg meeting described above; and
83 \$605,000 upon completion of this project. The project will be considered complete
84 when the written report described above is submitted.

85

86 C. authorize Amy Administrator to approve Contractor's work and any expenses
87 Contractor wishes to incur on behalf of Client.

88

89 3. Either Party may terminate this agreement with thirty days' written notice. If the
90 agreement is terminated, Contractor will present Client with a statement of account
91 showing all fees paid to that time, and itemizing work performed. If work performed
92 exceeds fees paid to date, Client will pay Contractor for such work at the rate of
93 \$40,000 an hour. If fees paid exceed work performed to date, Contractor will return

94 unearned fees to Client.

95

96 Signed:

97 for Iceberg County Art Center:

98

99 _____ Date _____

100

101 for Terrific Consulting:

102

103 _____ Date _____

104