TAPAS

IST-2001-34069

Trusted and QoS-Aware Provision of Application Services

D5 - TAPAS Architecture: Concepts and Protocols

Report Version: D5

Report Delivery Date: June 2003

Classification: Public Circulation

Contract Start Date: 1 April 2002 **Duration:** 36m

Project Co-ordinator: Newcastle University

Partners: Adesso, Dortmund – Germany; University College London – UK; University of Bologna – Italy; University of Cambridge – UK

Project funded by the European Community under the "Information Society Technology" Programme (1998-2002)

TAPAS Architecture: concepts and protocols

Carlos Molina-Jimenez
John Warne

School of Computing Science
University of Newcastle upon Tyne

(F. Panzieri, University of Bologna and S. K. Shrivastava, University of Newcastle upon Tyne, Eds.)

Table of Contents

TAPAS Architecture: concepts and protocols 1

1 Introduction 4

2 Regulating interactions 6

3 The virtual enterprise model 8

4 Monitoring and enforcement of contracts 12

5 Trust and trust-related models 21

6 Trust enforcement 25

7 Related work 35

References 37

Appendix 39

1 Carlos Molina-Jimenez, Santosh Shrivastava, Ellis Solaiman and John Warne, “Contract Representation for Run-time Monitoring and Enforcement”

2 Paul D Ezhilchelvan and Santosh K Shrivastava, “Systematic Development of a Family of Fair Exchange Protocols”

3 Nicola Mezzetti, “Towards a Model for Trust Relationships in Virtual Enterprises”

4 N. Cook, S.K. Shrivastava, and S.M. Wheater, "Dependable Information Sharing between Organisations Using B2B Objects"

Abstract: The aim of report D5 is to describe the interim TAPAS architecture for application hosting. To meet this requirement this document discusses the core concepts and algorithms needed to reason about and to build the TAPAS architecture. It develops the concept of virtual enterprise and describes how interactions between members of such an enterprise can be regulated by means of electronic contracts.

Keywords: Virtual enterprises, virtual objects, contracts, executable contracts, finite state machine, contract representation, contract monitoring, contract enforcement, contract implementation, finite state machines, non-repudiable evidences, events, trust, RBAC, digital evidence

1 Introduction

In the TAPAS project, we are particularly interested in developing solutions to the problem faced Application Service Providers (ASPs) when called upon to host distributed applications that make use of a wide variety of Internet services provided by different organisations. This naturally leads to the ASP acting as an intermediary for interactions for information sharing that cross organisational boundaries. However, despite the requirement to share information and services, autonomy and privacy requirements of organisations must not be compromised. Organisation will therefore require their interactions with other organisations to be strictly controlled and policed. This creates two major challenges. Firstly, contractual relationships between multiple organisations for information access and sharing will need to be governed by *service level agreements* (SLAs), which will need to be defined and agreed between the organisations and then enforced and monitored by the ASP. Secondly, the ASP will have to establish appropriate *trust relationships* with the organisations and implement corresponding security policies before organisations will permit the ASP to act as an intermediary for inter-organisational service invocations. Unfortunately, ASPs currently lack tools and techniques for offering hosting facilities for such distributed applications.

The lack of satisfactory solutions to the issues mentioned in the previous paragraph was one of the main motivations for the creation of the TAPAS project. It does not take long to realise that the issues discussed above are not constrained to the interaction between an ASP and a client. These issues arise in all business collaborations performed through networked computers. To justify our generalisation of the problem we will discuss here the example of the creation of a market place mentioned in [1].

Fig. 1 shows a market place that is built with the collaboration of several business partners: the owner of the market place, a TTP (Trusted Third Party), *n* vendors and a credit rating agency. Naturally, the owner of the market place has business interaction with buyers. What is relevant in the figure is that the owner of the market place relies of an ASP for computer related services. Likewise, the ASP relies on a ISP (Internet Service Provider) and on a SSP (Storage Service Provider). Similarly, the credit rating agency depends on the services provided by *n* retail banks.

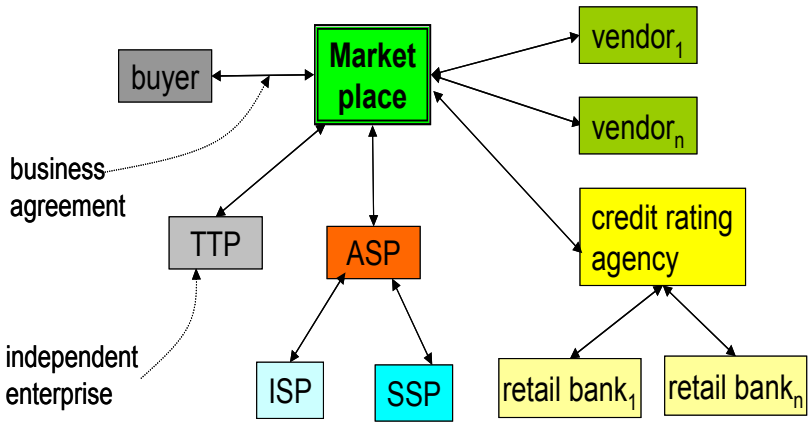

Fig. 1: A market place hosted by an application service provider.

The common pattern in this figure is the double-arrowed lines that join each pair of independent enterprises together. These arrows represent business agreements and imply business interactions. Business interaction is present between each pair of independent enterprises regardless of the nature of the service (network connection, disk storage, application service provision, credit rating information, etc.) being delivered.

This suggests that business interaction between the APS and the owner of the market place is just a special case of a general problem, namely that of building general purpose business partnerships.

It is conceivable that an enterprise might wish to take part in more than one business partnership at the same time. For this to be possible, enterprises need to keep their independence after joining a business partnership. The result of this is that a business partnership can be regarded logically as a new and independent (from its creators) enterprise. We are interested in implementing business partnerships electronically and call the new enterprise that results from the implementation *a virtual enterprise.*

The concept of virtual enterprise is central to the TAPAS project, however, to have space for providing more background about our understanding of business processes and business partnership, we will delay the discussion of virtual enterprises until section 3.

We do not make assumptions about the nature of the business interaction. Nor do we make assumptions about the goal of business partnership. The concepts we discuss in this work hold regardless of whether a business partnership is created for offering services to external users or for offering services only to users belonging to the enterprises that compose the partnerships. These two possibilities are illustrated in Fig. 2. It is worth mentioning that, although it is not shown in the picture, the services offered by a business partnership are normally accessible through an interface. In Fig. 2-a, this interface is visible only to internal users, whereas, in Fig. 2-b, this interface is made public, so that interested users can see and use it.

The reason for explicitly mentioning this matter is that we want to emphasise that our concepts of business partnership and virtual enterprise capture the business interaction of Fig. 1 where the ISP provides a service to the ASP as well as the business interactions where the TTP, ASP, credit rating agency and the *n* vendors provide a service to the buyer.

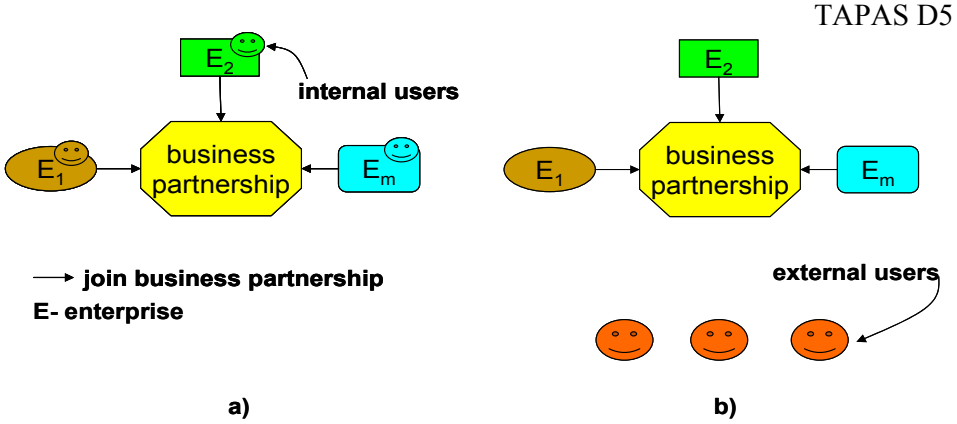

Fig. 2: Creation of business partnerships for internal and external use.

Regardless of the purpose of the partnership (either a) or b) in Fig. 2) the business interaction between the business partners is complex because the business partners have to share a great deal of information and because it involves cross-organisational processes. A *cross-organisational process* is a process whose execution involves components that belong to different independent enterprises and are located within the boundaries of their owners.

In the remaining of this report we will discuss how business interaction between two or more business partners can be regulated. In Section 2 we explain how business partnerships where members interact through the Internet can be build. To help reasoning about the business interaction between the members of business partnership we introduce a model for building virtual enterprises, Section 3. In Section 4 we discuss how executable contracts can be used for monitoring and enforcing at run-time the right and obligation of each business partner. In Sections 5 and 6 we introduce a trust model for guaranteeing that only entities that are legally entitled to gain access to the resources of the business partnership do. In Section 7 we discuss some works that are relevant to the TAPAS project. The Appendix contains papers and work in progress written by members of the TAPAS projects where some of the concepts mentioned in this report are discussed at large.

2 Regulating interactions

The interaction between *m* enterprises that participate in a business partnership can be abstracted as shown in Fig. 3. This figure shows all the main components that a business partnership involves.

Let us assume that in Fig. 3. enterprises E_1 and E_2 are trading partner and the builders of the business partnership shown in the figure. The purpose of the business partnership is the execution of a set processes, namely, *business process*$_1$, *business process*$_2$, ... , *business process*$_n$. The execution a *business process*$_i$ involves resources belonging to E_1 and E_2; for this reason these processes are called *cross-organisational*. The execution of a process starts when an operation to create an instance of such process is successfully invoked. Since we are aiming at general purpose business, we assume that at a given time, *n* instances of each business process might be in execution. Naturally, *instance*$_i$ and *instance*$_j$ of *business process*$_k$ do not necessarily follow the same path when they run; their execution paths are determined by the events that occur during their execution.

As one can see in the figure, each enterprise has a set of people with different skills and responsibilities (managers, engineers, etc.). These people are called *role players* and discussed in-depth in Sections 4.7 and 6.1.2, for now it suffices to say that role players are the entities that invoke operations on instances of business processes. It is important to mention that with role players and process instances we have a many to many relation; this mean that a given role player can invoke operations on more than one process instance and that a given process instance can receive the invocations of operations from more than one role player.

The picture shows what happens when $engineer_1$ from enterprise E_1 creates an instance (label (1)) of $business\ process_j$. $Instance_1$ of $business\ process_j$ runs (2), and at some point it requires the intervention (the invocation of operations) of several role players, this is shown by labels (3), (4), (5), (6) and (7).

Fig. 3 also shows how $engineer_n$ creates $instance_n$ of $business\ process_j$ (1') while $instance_1$ of $business\ process_j$ is till running. Notice that the new instance of business $process_j$ is created by $engineer_n$ and requires the intervention only of the manager of enterprise E_1 and the accountant of enterprise E_2 (labels (2') and (3') respectively). Obviously, $instance_n$ of business $process_j$ could have been created by $engineer_1$ as we do not restrict the number of instances a role player can interact with. Likewise, $engineer_n$ could have created an instance of $business\ process_1$ or of any other business process from the set of business processes available within the business partnership.

To relate the abstract example of Fig. 3 to a practical example we can imagine that enterprises E_1 and E_2 work together to design airplanes. Thus, $engineer_1$ creates an instance of a process to request a copy of the specifications of $part\ number_1$, whereas $engineer_n$ is requesting a copy of the specification of $part\ number_n$.

From this discussion, it should be apparent that business partnerships could involve rather complex interactions between the business partners. There are several crucial questions that need to be addressed:

- What resources and information is each participating enterprise prepared to share with its business partners in order to run the business partnership?
- What operations is each participating enterprise allow to execute or expected to execute on a given instance of a business process?
- When and in what order are operations executed on instances of business processes?
- Who, Alice the manager, Bob the engineer, etc., can legitimately create instances of business processes and who can legitimately execute operations on these business process instances?
- Who keeps digital non-repudiable records about the execution of operations on the process instances so that disputes can be fairly resolved?

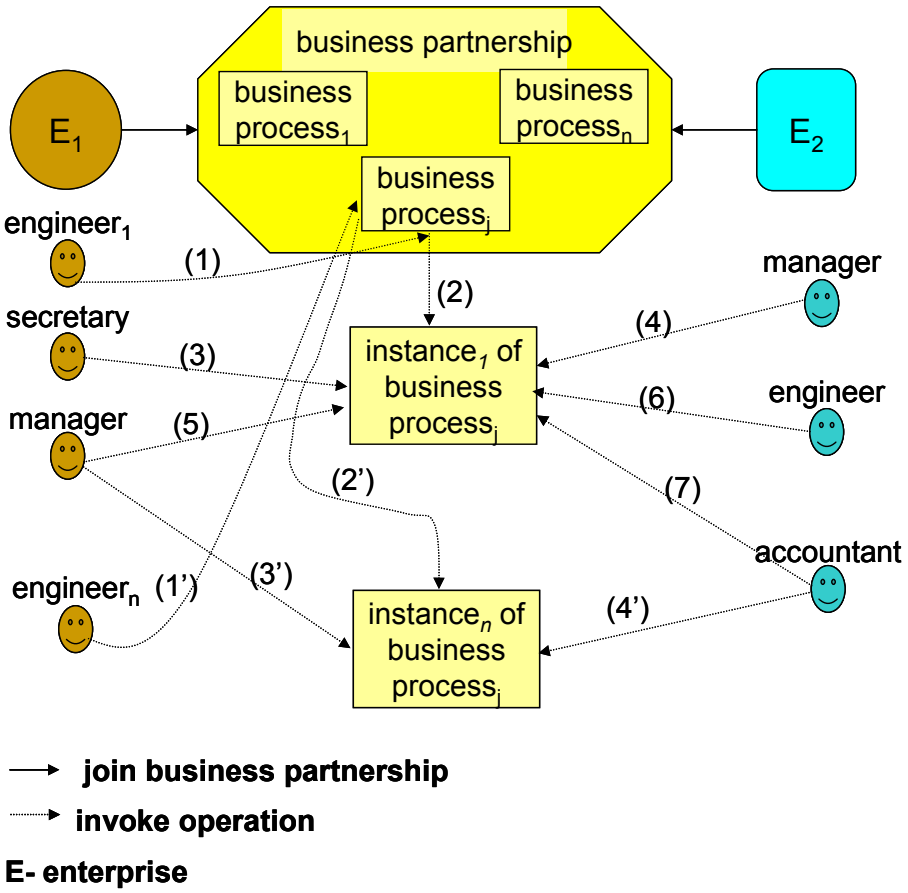

Fig. 3: General view of business interaction.

3 The virtual enterprise model

We believe that the complexity of the business interaction expected to take place can be modelled by means of the virtual enterprise model.

3.1. Definition of a virtual enterprise

A *Virtual Enterprise* (VE) is an enterprise composed out of *n* independently existing and possibly mutually suspicious enterprises ($E_1, E_2, \ldots, E_n$) that wish to establish a close business relationship for an agreed upon period without loosing their independence.

In this definition, the sentence *close business relationship* means that the enterprises engaged in a virtual enterprise expect to **access each other services** frequently and several times as opposite to one off trading. This means that during the period of the business relationship a significant amount of **information is shared** between the participating enterprises. The length of the business period is specified in a legal, business contract and is normally months or years. Needless to say, it is assumed that the participating enterprises are liked together by a communication network like the Internet.

Regardless of the number of enterprises that team together to build a virtual enterprise, a virtual enterprise looks, to its users, like a single enterprise, that is, its users do not see the

complexity of the interaction between the composing enterprises because the virtual enterprise conceals it.

3.2 Private and shared objects

In our model, an *object* is any resource or service that can be named. Examples of objects are documents, files, databases, computers, disks, printers, network connections, etc. It is assumed that before joining a virtual enterprise, each participating enterprise owns *m* objects.

The need to share services and information without compromising each other's independence, urges each participating enterprise to organise its objects as shown in Fig. 4.

Before joining a virtual enterprise a participating enterprise is expected to separate its objects into two sets, namely, into a set of ***private objects*** and a set of ***shared objects.***

In each participating enterprise the set of private objects contains all the objects that the enterprise wishes to conceal from its partners. Conversely, objects that the enterprise wishes to expose to its partners are grouped into the set of shared objects.

How the separation of the two sets is performed internally is a matter and responsibility of each enterprise.

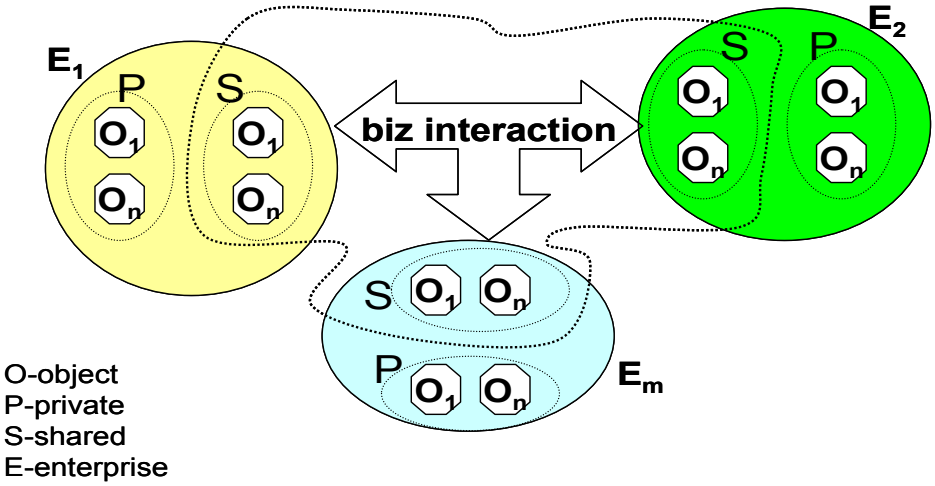

Fig. 4: Private and shared objects.

In Fig. 4, a dotted line is used to group together all the sets of shared objects. Later on we will discuss how the objects within the boundaries of the dotted line can used to build a virtual enterprise.

3.3. Realisation of a virtual enterprise

A virtual enterprise is called *virtual* because its main components are virtual objects. These virtual objects are provided by the enterprises that join the virtual enterprise and are realised as pointers to their shared objects. The realisation of a virtual enterprise composed out of two enterprises is shown in Fig. 5.

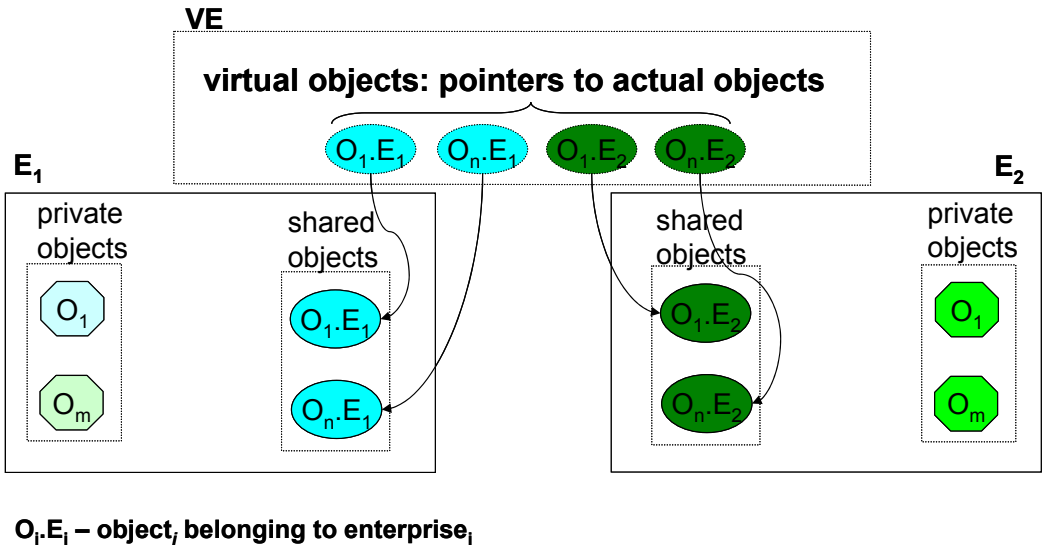

Fig. 5: Virtual objects of a virtual enterprise.

For simplicity, the virtual enterprise shown in Fig. 5 is composed out of two enterprises, that is, in this example, the number of participating enterprises is $m = 2$, however our model is general and is valid for any $m \geq 1$. This is illustrated in Fig. 6.

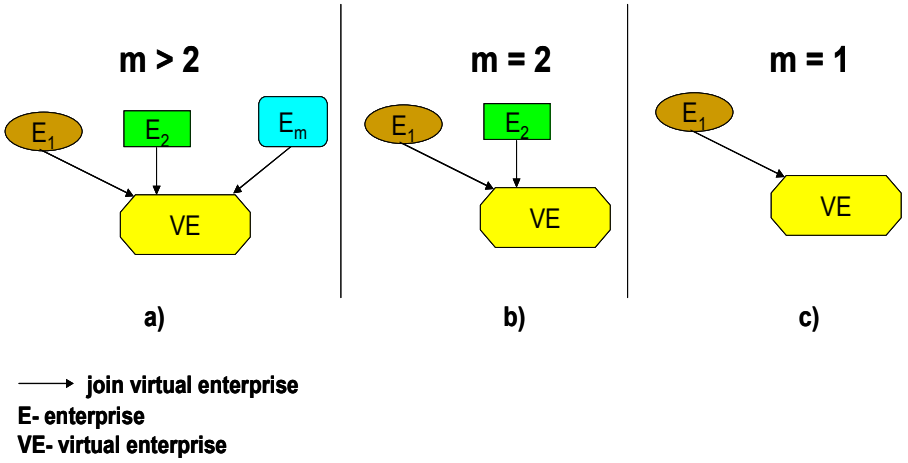

Fig. 6: Virtual enterprises with different number of composing enterprises.

Though our model makes no assumptions about the nature of the business between the composing enterprises it is worth mentioning that the case where $m > 2$ is typical of business partnerships where more than two enterprises gather together to compose a service that is presented as a single service to external users. The case where $m = 2$ is typical of applications where a supplier and a provider decide to build a virtual enterprise to make the provision of the service more efficient. The last case, where $m = 1$ is mentioned here for generality and because it is conceivable that the owner of an enterprise might think ahead and design its enterprise as a virtual enterprise and with the intention of attracting business partners to join it.

3.4. Extendibility and recursivity of the model

A question that inevitably arises at this point of this discussion is how the objects to which the virtual objects point to in Fig. 5 are realised. In other words, where are the real objects physically located?

To answer the question about where the physical objects accessed from within a virtual enterprise are located, we can say that our model of a virtual enterprise is general, extendible and recursive. We do not impose restrictions about the location of the actual objects shown in Fig. 5. Nor do we impose restrictions on the realisation or nature of E_1 and E_2. In our model, an enterprise that joins a virtual enterprise can be either real or virtual. This flexibility of our model is in line with new trends in the electronic business world where users are offered services that are composed out of existing services offered by different companies. With these arguments in mind, we can argue that the picture of a virtual enterprise presented in Fig. 5 can be generalised as shown in Fig. 7.

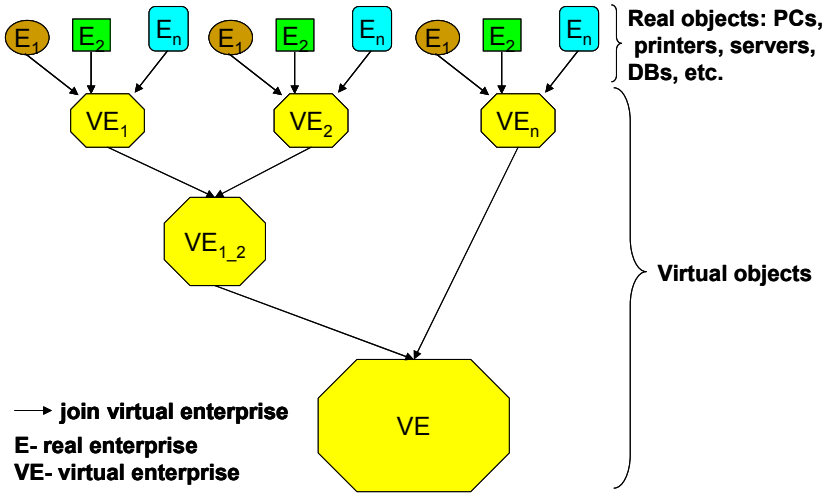

Fig. 7: Virtual enterprises composed of virtual enterprises.

As it can be appreciated from the figure, the actual objects to which the virtual objects point to, from within a virtual enterprise, are located somewhere in leaves of the tree of composing enterprises. This means that an operation invoked on a virtual object might travel down the tree crossing the boundaries of several enterprises till the operation is executed on the real object. Naturally, the invoker of the operation does not need to know about the complexity behind the interface he or she sees.

3.5. Virtual objects with interfaces

The creation of virtual object is only first step towards the creation of a virtual enterprise. Once the virtual objects are available we need to build a means for regulating the access to these objects. We need to regulate how, when and who can execute operations on the virtual objects. The key to answer these questions is the provision of well-defined interfaces to the virtual objects.

As shown in Fig. 8, in our model of virtual enterprise, each virtual object is provided with *n* interfaces ($I_1, I_2, \ldots, I_n$) and each interface contains *m* operations such as *R, W, Del, Update,*

Send, Rcvd, *Accept, Reject*, etc. The idea is that different interfaces are assigned to different role players. A role player to whom a given interface has been assigned has the privilege of executing all the operation specified in the interface. An operation on a virtual object is allowed only if it is a legal operation accordingly with the business contract and only if it is invoked by a legitimate role player. This is enforced at run time by the electronic "x-contract" shown in the figure. Notice that for simplicity, the figure shows only the set of shared object of each participating enterprise. How the roles are assigned to entities is discussed in Section 6. How a contract can be represented by a set of computaional objects, referred to as an x-contract is discussed in Section 4.

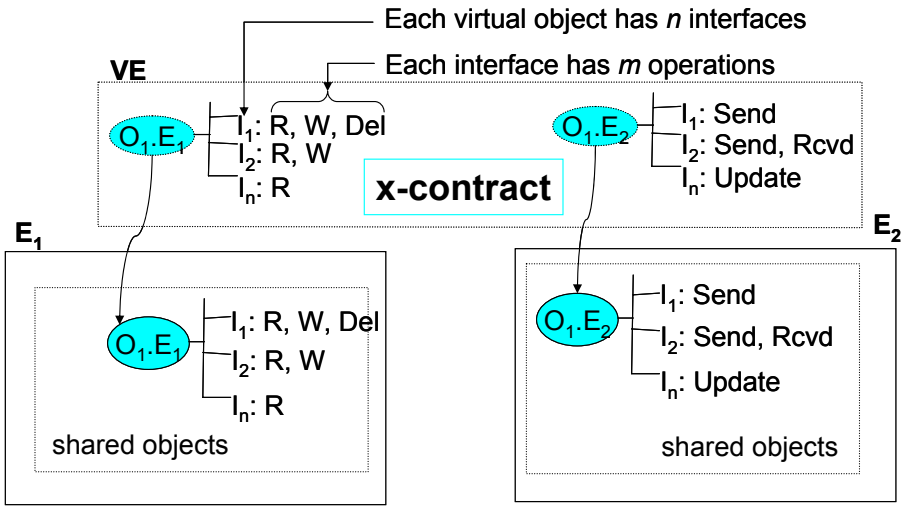

Fig. 8: Virtual objects with interfaces.

4 Monitoring and enforcement of contracts

In the conventional world, a *contract* can be defined as a legal document, written in a human language (for example, in English) that contains a set of clauses that stipulate the rights and obligations that two or more signing parties agree to honour for a period of time stipulated in the document.

In the business world, a contract is a document that regulates the interaction between two or more parties willing to conduct some business. Business partnerships like that discussed in Section 2 are inconceivable without a business contract.

No business partnership can run successfully unless the rights and obligations stipulated in its contract are monitored and enforced. In the conventional world, monitoring and enforcement of contracts is done manually. In this section we will show that it is possible to implement an electronic version of a conventional contract that monitors and enforces what the original contract dictates.

4.1. Service level agreements, rights and obligations

To understand how a conventional contract can be converted into an electronic equivalent it helps to think of a conventional contract as a document that contains a set of Service Level Agreements (SLAs), namely, $\{SLA_1, SLA_2, ..., SLA_n\}$, where n is an integer and should be $n \geq 1$ since a contract with no clauses is of no interests for this discussion.

We can think of each SLA_i as a contractual clause that describes a specific technical aspect of the global contract. It describes how a specific service should be provided and the metrics for measuring the delivery of the service. For example, is a contract related to the delivery of storage service, SLA_1 would describe the amount of megabytes to be make accessible to the customer, the response time, the number of simultaneous sessions, etc, whereas SLA_2 would describe security aspects such as guarantees of data protection, cryptographic algorithms, length of cryptographic keys, and so on. Finally, SLA_3 would describe how the payment for the service should be provided, discounts and fines for failing to pay by due dates, and so on.

To convert a human language contract into its electronic equivalent we should be able to abstract the contain of each SLA_i as two sets: a set of Rights (R) and a set of Obligations (O) that the contracting parties agree to honour.

An example of a contract represented as a set of SLAs is shown in Fig. 9. The contract shown in this figure involves only two contractual parties, namely, enterprises E_1 and E_2. The contract has been signed by the managers of these enterprises. Notice that for the sake of clarity, each right and each obligation has been superscripted with the name of the party responsible for it. For example, R_1^{ME1} is a right expected to be honoured by the manager of enterprise E_1, whereas R_1^{ME2} is a right expected to be observed by the manager of enterprise E_2. Notice that in each SLA_i the manager of enterprise E_1 has agreed to honour m rights and p obligations. Similarly, the manager of enterprise E_2 has agreed to honour n rights and q obligations in each SLA_i. It is probably worth mentioning that m, n, p and q are integers and equal or greater than zero. Obviously, for a contract to make sense it should have at least one right or one obligation.

Note that at this stage we are primarily concened with “horizontal” SLAs as defined in [2]. Such an SLA is between peer to peer entities. A “vertical” SLA typically specifies a quality of service contract for resource usage between an application and the underlying middleware services. Such SLAs will be used in QoS enabled application servers as discussed in [3].

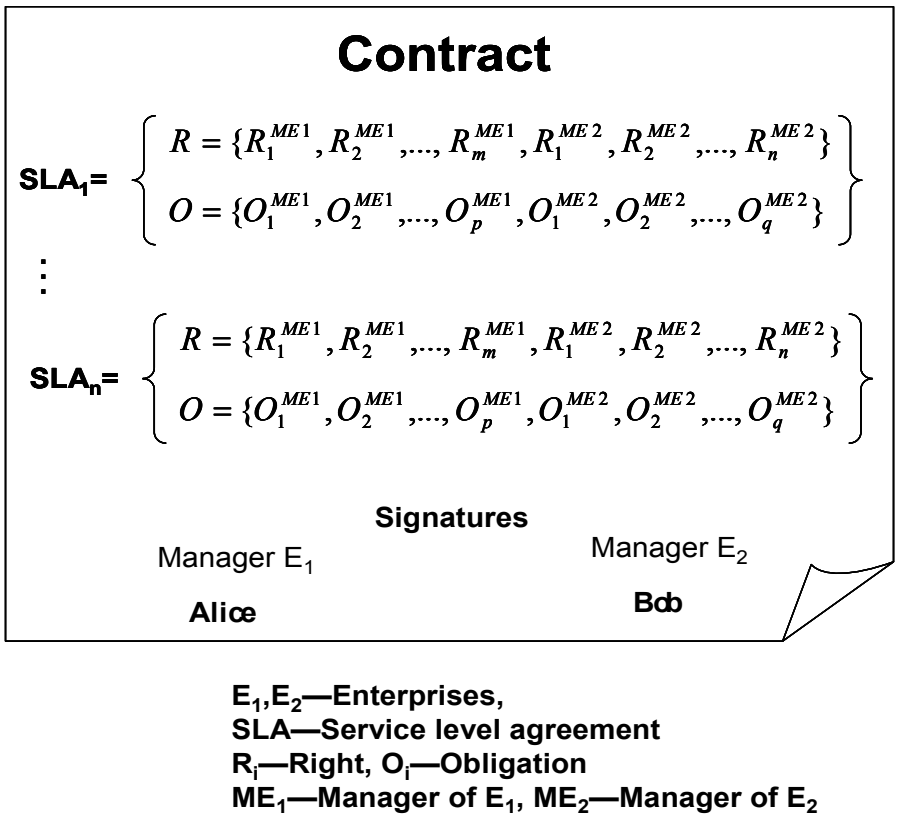

Fig. 9: Service level agreements, rights and obligations in a contract.

4.2. Conversion of conventional into executable contracts

One of the main aims of the TAPAS project is to regulate the interaction between trading partners with little or no human intervention. Since contracts are the means for regulating business interactions, this is the same as saying that the TAPAS architecture should support monitoring and enforcement of contracts automatically. In other words, we are interested in contracts that can be executed, that is, animated. We call executable contracts **x-contracts.**

An *x-contract* can be defined as a piece of software that monitors and enforces at run time the set of rights and obligations stipulated in a conventional contract written in a human language.

An x-contract contains parts meant to be understood by humans and parts that are meant to be understood by computers. This means that it contains computer-executable and human-readable files. An x-contract built with current technology would contain XML and executable java files, Word and ascii documents, graphics, pictures and whatever is necessary to ease the execution of the x-contract.

Conceptually, it helps to think that the files that compose an x-contract are located between the trading partners. However, the actual location of these file depends on the implementation.

We have learnt from our own experience that the most challenging task is the description of what the original contract stipulates in its clauses into a formal mathematical notation that can be unambiguously coded in a computer language.

The difficulties with the conversion are caused by the ambiguities that the original document is likely to have. The existence of ambiguities is understandable since contracts are traditionally written by a non-technical person (for example, a lawyer) and expected to be read by other humans that would normally tolerate and make sense of obvious ambiguities. The technical person in charge for implementing the x-contract has to be sure that no ambiguities are left in the contract before coding it in a computer language.

For medium and large sized contracts it might be advisable to code the original contract into an abstract validation model to formally validate the correctness of the contract. Then when an acceptable degree of confidence about the correctness of the contract is reached, the technical person simply translates the validation model into the implementation language to produce the x-contract. This process is shown in Fig. 10.

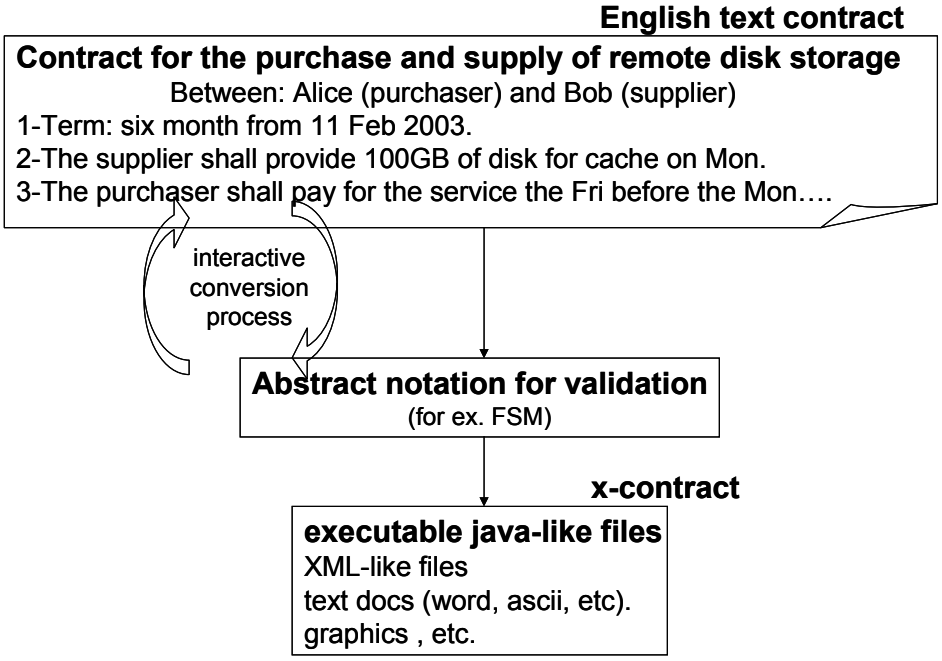

Fig. 10: Creation of an x-contract.

What mathematical notation is used for describing a contract and what computer language is used for programming this description is a matter of choice.

The selection of the programming language is of secondary importance as it does not affect the design at this stage; in fact, there is no need to make any decision about it at this stage. However, the selection of the mathematical notation at this stage might have a significant impact on the development of the design.

As a reference, we can say that the mathematical notation used for describing the contract should meet two requirements:

- It should have a semantic powerful enough to describe the rights and obligations stipulated in the contract and the events that trigger them.

- It should have software tools for verifying that the contract is free of ambiguities before attempting to code it into the programming language chosen for the implementation.

We have found out that a mathematical notation that meets these requirements is the Finite State Machines (FSM). Thus, we have decided to use them to describe our business contracts [4]. Another argument that justifies our choice is that FSMs are supported by a sound and well studied theory.

4.3. Use of FSMs for describing contracts

Formally, a finite state machine M is defined as the quintuple $[S, I, Z, \delta, \lambda]$, where $S = \{s_1, s_2, ..., s_m\}$, $I = \{i_1, i_2, ..., i_n\}$ and $Z = \{z_1, z_2, ..., z_p\}$ are finite nonempty sets of states, input symbols and output symbols, respectively. $\delta : S \times I \to S$ is the transition function and $\lambda : S \times I \to Z$ is the output function.

Informally, M describes an abstract system that stays in a given state until it receives an external stimulus. When such stimulus is received, the system reacts by doing something (for example, sending an output signal) and then moves to a different state. Note that *do something* might mean do nothing in some circumstances and that the new state is not necessarily different from the previous. The behaviour of this abstract system is deterministic. The quintuple $[S, I, Z, \delta, \lambda]$ unambiguously defines what to do and where to go next.

Because of their high level of abstraction, FSMs are used to describe and model a great variety of systems. In particular, the computer science community has gained a great deal of experience in the use of FSMs for describing communication protocols, and built several tools for validating such protocols. For example, Spin [5] is a well known protocol validator.

We have introduced communication protocols into the discussion about x-contracts for a valid reason: we strongly argue that from the point of view of the interaction and synchronisation between the parties involved, x-contracts are equivalent to communication protocols. We claim that x-contracts, as communication protocols are, can be precisely abstracted by FSMs. The advantage of looking at contracts as FSMs is that we can put into practice all the existing machinery that was originally developed for studying communication protocols. For instance, we can resort to Spin to validate an x-contract before converting it into the actual computer program that will enact it. The goal of a validation process is to analyze what is known as the correctness properties of the system. In other words, the essence of the validation is to discover, at an early stage, whether the execution of an x-contract takes the contracting parties into unacceptable situations. Among other things, validating the FSM model of an x-contract should reveal the existence of states (conditions in the x-contract) that are not reachable, that is, states for which there is no path from the initial state. If one of these unreachable states represents the receipt of the goods the situation would be unacceptable and the x-contract would need to be re-written. In the same order, the validator should show that at some point, the two contracting parties reach a final state (for example, end of contract) instead of being left in a transient state for ever. To mention another example, the validation should reveal whether the x-contract allows purchasers to receive goods before paying for them.

A question that naturally arises at this point is how the rights and obligations stipulated in a contract can be represented in a FSM.

4.4. Mapping conditions, rights, obligations and events into a FSM

At the level of rights and obligations an x-contract is often more easily understood as a set of FSMs, one for each contracting party. So, a contract between a purchaser and a supplier is represented as two finite state machines: one FSM for the purchaser and one FSM for the supplier, FSM_P, FSM_S respectively.

The physical location of each FSM is irrelevant to the functionality of the contract and is decided at the time of implementation. For the moment let us assume that FSM_P is located within the purchaser's enterprise and FSM_S is located within the supplier's enterprise. To enact the x-contract these two FSMs must share a common communication channel to interact with each other, that is, the output of FSM_P is somehow connected to the input of FSM_S and vice versa. We will now discuss how the rights and obligations stipulated in a contract can be mapped into the FSMs.

To reason about the rights and obligations stipulated in a contract it helps to think of them as operations. An arbitrary segment of a contract can be described by the following general syntax:

if $event_1$ & $condition_q = true$

perform $operation_1$ *and switch to* $state_1$

else if $event_2$ & $condition_q = true$

perform $operation_2$ *and switch to* $state_2$

... ...

else if $event_m$ & $condition_q = true$

perform $operation_m$ *and switch to* $state_m$

This syntax expresses the idea that, at some point an contract can be at any of *n* possible conditions ($condition_1, condition_2, ..., condition_n$). If the x-contract is in a given $condition_q$ (for example, *WatingForOffer*), there is a finite and well defined set of events ($event_1, event_2, ..., event_m$) that can affect the future behaviour of the x-contract. The occurrence of $event_i$ determines what objects (variables, files, database, etc.) within the system change their values, that is, the event determines to which new condition the systems switches. Similarly, there is a finite and well defined set of operations ($operation_1, operation_2, ..., operation_m$) that can be executed when the system is in $condition_q$. The $event_i$ determines the operation to be executed.

Bearing in mind the definition of a FSM presented in Section 4.3, we argue that the set of conditions of the general syntax presented above can be mapped into the set *S* of states of a FSM. Similarly, the set of events can be mapped into the set *I* of input symbols of the FSM. In the same order, the set of operations can be mapped into the set Z of output symbols of the FSM. Finally, we can map the set of switches to the next condition into the transitional

function δ. It is important to bear in mind that the operation *donothing* is a valid operation. In this discussion we represent it with the symbol ε.

Thus, in terms of FSMs, we can express the above syntax as shown in Fig. 11.

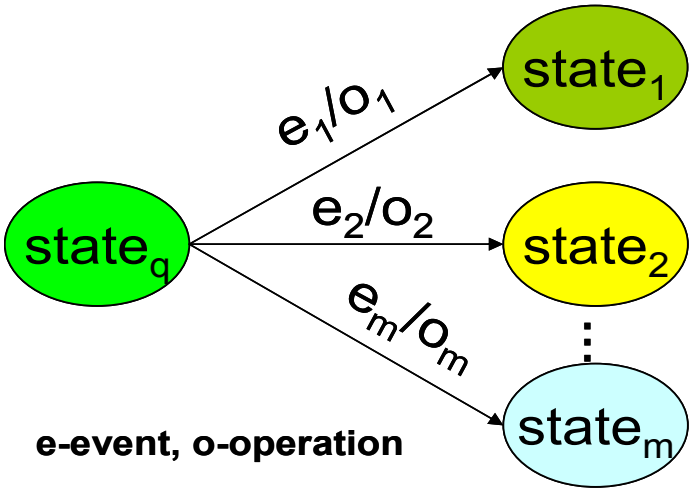

Fig. 11: Mapping of conditions, events and operations into a FSM.

4.5. Invocation of rights and obligations

To illustrate how the rights and obligations are triggered we will examine Fig. 12. This figure shows a snapshot of the two FSMs that describe the x-contract between two contracting enterprises, for the sake of the discussion let us say that these two enterprises are the purchaser and the supplier of some goods.

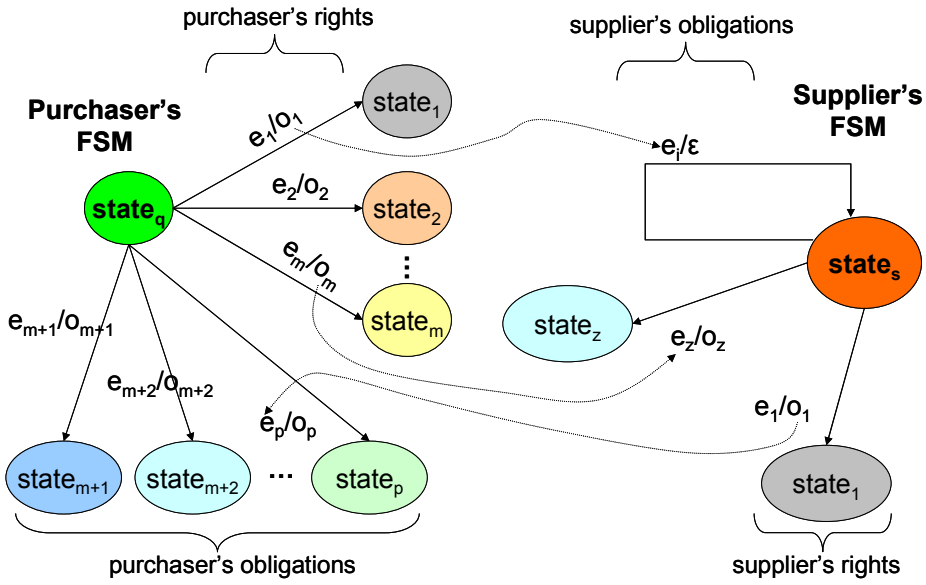

Fig. 12: Execution of an x-contract between a purchaser and a supplier..

To reason about how the contractual rights and obligations can be monitored and enforced by FSMs, it is useful to look at the rights and obligations a contracting

enterprise has at a given state of the execution of the x-contract. In terms of FSMs, this is equivalent to looking at the set of operations that can be executed when the FSM of contractual party is in a given state $state_q$. It is useful to classify this set into two subsets: the subset of operations the owner of the FSM has the right to perform and the subset of operations that person has the obligation to perform, $\{o_1, o_2, ..., o_m\}$ and $\{o_{m+1}, o_{m+2}, ..., o_p\}$, respectively.

In the snapshot of the execution of the contract shown in Fig. 12 the purchaser's FSM is in $state_q$, whereas the supplier's is in $state_p$. As it can be appreciated from the figure, the rights and obligations the purchaser has when his FSM is in $state_q$ can be mapped into the sets $\{o_1, o_2, ..., o_m\}$ and $\{o_{m+1}, o_{m+2}, ..., o_p\}$, respectively.

Executing an operation from the subset $\{o_1, o_2, ..., o_m\}$ means exercising a right given by the x-contract. Since each operation o_i is paired to an *event* e_i, the operation o_i can be executed only after the occurrence of e_i. How does event e_i occur?

Event e_i can be triggered internally within the purchaser's enterprise or externally, say for example, within the supplier's enterprise and then notified somehow to the purchaser's FSM. Since we are talking about rights, the execution of operation o_i is optional; because of this, event e_i might be deliberately triggered by the purchaser (for example, when the purchaser wishes to send a purchase order). Also, it can be the result of an unavoidable situation within the purchaser's enterprise (for example, a notification that the mainframe computer is non-functional) or it can be triggered by a notification received from the supplier (for example, when the supplier wishes to offer a new item to the purchaser).

Executing an operation o_{m+i} from subset $\{o_{m+1}, o_{m+2}, ..., o_p\}$ means complying with the contractual obligations the purchaser has when his FSM is in $state_q$. As with the rightful operations, the obligatory operations are paired to events which are triggered internally, or externally.

It is important to understand that exercising a right at one side of the contract might or might not have an effect at the other side. This depends on what the text of the original contract stipulates. The execution of operation o_i at the purchaser's side might trigger a right, an obligation, or nothing at the supplier's side. By nothing we mean that the supplier's FSM is no notified about the execution of the operation o_i at the purchaser's side. Similarly, the execution of an obligatory operation o_{m+i} from the subset $\{o_{m+1}, o_{m+2}, ..., o_p\}$ might trigger a right, an obligation or nothing at the supplier's side.

The dashed line pointing from the pair e_1 / o_1 at the supplier's side to the pair e_p / o_p at the purchaser's side implies that in $state_s$ the supplier has the right to execute the operation o_1. Obviously, we are assuming that text of the original contract stipulates that the purchaser (being in $state_q$) has the obligation to execute operation o_p upon receiving a notification of the execution of operation o_p at the supplier's side when the supplier is in $state_s$. Similarly, the dashed line pointing from e_m / o_m to e_z / o_z shows that in $state_q$ the purchaser has the right to execute the operation o_m. As a response to this operation, the supplier has the obligation

to execute the operation o_z. The dashed line from e_1 / o_1 to e_1 / ε shows that the purchaser's has the right to execute the operation o_1. However, the execution of such operation demands nothing at the supplier's side.

4.6. Interception of operations on virtual objects

A question that we have not addressed explicitly yet is how the x-contract regulates the execution of operations on the virtual objects. In our model of virtual enterprise, an x-contract works as an interceptor of operations. This is shown in Fig. 13.

The figure shows some steps of the execution of an x-contract between two arbitrary enterprises, namely, *E1* and *E2*. It is assumed that *E1* is a supplier and *E2* a purchaser.

The figure shows the behaviour of the contract when a role player from the purchaser's enterprise decides to place a payment. When this role player decides to place a payment (presumably by issuing a command from his keyboard) a notification of this event is sent to the purchaser's FSM (1). Receiving this notification causes the operation *pay* to be invoked on one of the interfaces of the *payment.E2* virtual object (2). This operation is forwarded (3) to the actual object *payment.E2* located within the enterprise *E2*. Once the payment is placed in the actual object *payment.E2*, an event is sent to the supplier's FSM. At the supplier's FSM this event is received as *PaymentRcvd* (4). This event triggers the invocation of the operation *collect* (5) on one of the interfaces of the virtual object *payment.E2*. This invocation is forwarded (6) to the actual object *payment.E2* located within the enterprise *E2*. Upon collecting the payment, the supplier's FSM makes a transition from state *Waiting for payment* into state *Shipping item*. In this state the supplier's FSM waits until a role player from the supplier's enterprise issues a command (let us say from her keyboard) to indicate that the item is ready for delivery. This event is received by the FSM as *ItemReady* (7). Receiving this event causes the operation *send* to be invoked (8) on one of the interfaces of the virtual object *item.E1*. This operation is forwarded (9) to the actual object *item.E1* located within the enterprise *E1*.

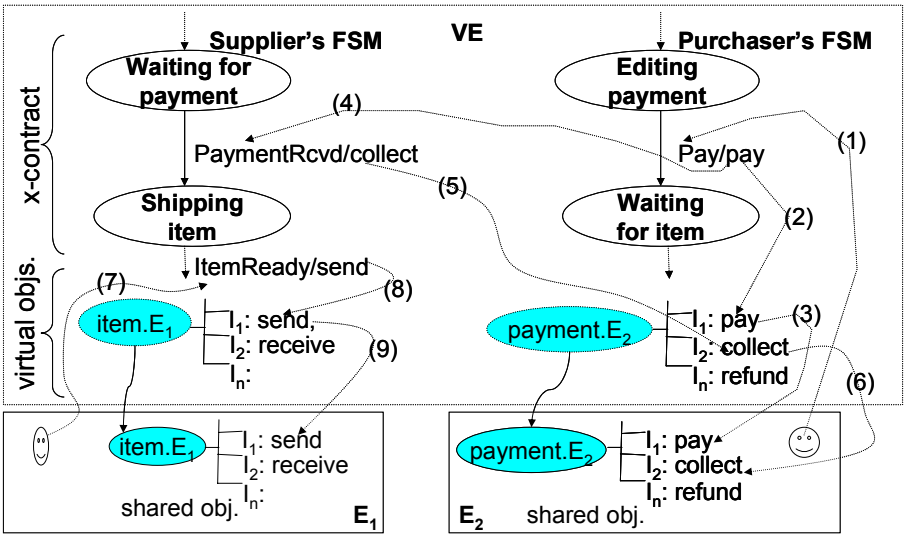

Fig. 13: An x-contract as an interceptor of operations.

4.7. Role players and x-contracts

Notice that in Fig. 13, all the operations (*pay, collect* and *send*) invoked on the virtual objects executed successfully, that it, they were intercepted by the x-contract and approved. This means that in that discussion we assumed that all the operations were invoked by legitimate role players. This is an exception. It is conceivable that in practice illegitimate role players will try to execute operations they are not entitled to. Invocations of illegitimate operations will be intercepted by the x-contract, disapproved and rejected. How the x-contract decides who is a legitimate role player and who is not is discussed in Section 6.1.3.

It is important to realise that the pair of FSMs shown in Fig. 13 monitors and enforces only one instance of a business process between the purchaser and the supplier. In our view over business interaction, we consider than more than one instance of different business processes can be active at the same time (see Section 2). If this is true, we would need one pair of FSMs for each instance of business process. As it was discussed in Section 2, two instances of a business process do not necessarily follow the same path. Consequently, different pairs of FSMs, involve different role players.

The set of role players associated with a given pair of FSMs is called a *role set*. This is discussed in-depth in Section 6.1.2.

Naturally, different instances of business processes might interact with each other. How to control the interaction between the pairs of FSMs that represent the business process instances is still an open question. We believe that this can be done with the help of another FSM, a kind of parent FSM to look after the pair of FSMs that represent the business process instances.

5 Trust and trust-related models

Of fundamental importance to TAPAS are the issues of *trust* and *trust management*, and the interrelationships between the notions of *behavioural trust*, *dependability* or *trustworthiness*, *dependence*, *confidence*, and *failure*. A firm understanding of these notions becomes crucial when we assume that the individual organisations (enterprises) within a TAPAS-compliant Virtual Enterprise (VE) might be mutually suspicious of one another, and as such might require their trust relationships to be monitored, and wherever possible enforced, so as to be seen to be trustworthy.

In this context, we define **behavioural trust** as *the mutual judgement of and dependence on the expected behaviours of one another's organisations in specifically agreed inter-organisational interactions, such that there is a mutual feeling of relative confidence in these interactions, even though negative consequences are possible*. Such negative consequences will appear as violations of trusted interactions, and thus be interpreted as failures of agreed trust relationships.

5.1. Trust and related concepts

It is instructive to examine the notion of *trust* and its related concepts of *dependability*, *dependence* and *failure*.

We begin by examining the concept of ***failure*** using the normally accepted dictionary definitions of the terms *system* and *judgement*:[1]

> "A given system, operating in some particular environment (a wider system) may fail in the sense that some other system makes, or could in principle have made, a judgement that the activity or inactivity of that given system constitutes failure.
>
> The second system, the judgemental system, may be an automated system, a human being, a relevant judicial authority, or whatever. (It may or may not have a documented system specification to guide it.) Different judgemental systems might, of course, come to different decisions regarding the given system. Moreover, such a judgemental system might itself fail – in the eyes of some other judgemental system – a possibility that is well understood by the legal system, with its hierarchy of courts. So, we can have a (recursive) notion of *failure* which is defined merely in terms of what are taken to be as the fundamental, dictionary-defined, concepts of *system* and *judgement*, and which clearly is a relative rather than an absolute notion."

This definition of *failure* leads naturally to a related definition of ***dependability***, which includes the system attributes of *reliability*, *availability*, *security* and *safety*, a property such that reliance can be placed upon the services a system is expected to deliver:

> "Given the above definition of failure, the concept of ***dependabilit***y can be simply defined as *the quality or characteristic of being dependable*, where the adjective *dependable* is attributed to systems whose failures are judged to be sufficiently rare or insignificant.
>
> Dependability measures can be assessments of past behaviour or predictions of future behaviour, relating to instantaneous events demanding response, or durations of system service. It is convenient to frame a totally dependable system as one that has a numerical measure of 1, and a totally undependable system as one with a measure of 0. In reality such measures will neither be expressed as single numbers nor single probability distributions, but rather as sets of measures corresponding to sets of types of the dependability of concern. Moreover, there may be even greater difficulty in establishing actual numerical values, for example, in the measures of security aspects of dependability."

These notions of *failure* and *dependability* lead to the concept of ***dependence***:

---

[1] The definitions of the terms *failure*, *dependability*, *dependence*, and the discussion on *trust* given herein in inset text, quoted essentially literatum, are taken from a private communication written by Brian Randell, University of Newcastle upon Tyne. For background concepts, see [6, 7].

> "It is commonplace to say that the dependability of a system should suffice for the dependence being placed upon that system. What is meant by the term *dependence of A on B* is a measure of *B's undependability impact on A's dependability*. Clearly, this measure can vary from a value of 1 (***total dependence*** – in which case, any failure of B will cause A to fail) to a value of 0 (***total independence*** – in which A will not fail when B fails)."

This notion of *dependence* leads in turn to the concept of ***trust***:

> "It is recalled that the NSA definition of a trusted component is (roughly) "one whose failure could cause your security policy to fail"
>
> It seems that the ***trust*** *A* has in *B* could be described as *acceptable dependence*; that is, the dependence of *A* on *B* allied to a judgement that this level of dependence is commensurate with *A*'s acceptance of *B*'s *dependability*.
>
> This judgement (made by or on behalf of *A* about *B*) may be explicit, and even laid down in a contract between *A* and *B*; but might also be implicit, or *dispositional* to the extent that *A* has a consistent tendency to trust *B* across spectrums of situations, or may even be unthinking. It may also be *situational* in that *A* has no option but to put its implicit trust in *B*, irrespective of the beliefs of the attributes of *B* in a given situation[2].
>
> Thus to the extent which *A* trusts *B*, need not assume responsibility for providing the means of tolerating *B*'s failures. (The question of whether *A* is capable of doing so is another matter.) Indeed, turning things around, the extent to which *A* fails to provide means of tolerating *B*'s failures is a measure of *A*'s (perhaps unthinking or unwilling) trust in *B*.
>
> Any system which provides evidence that can be used to justify *A*'s trust in *B*, and therefore provide confidence to *A*, can itself of course fail. One such failure of a confidence-building system (which might be system *A* itself) producing an underestimate of *A*'s dependence on *B*, which could lead to a decision to avoid using *B*, even though *B* is adequately dependable. What is normally a more serious type of failure of a confidence-building system puts *A* at unacceptable risk due to a failure of *B*, the case where system *B* proves to be *untrustworthy*.
>
> A distinction between trust and confidence is that the former leads to the act of becoming dependent, whilst the latter concerns how much one might feel about this act."

---

[2] The authors have added these comments to Randell's text in order to clarify the nature of dispositional and situational judgements.

5.1.1. Trust Propositions

The above notions of failure, dependability, dependence, and trust, particularly with respect to behavioural trust, correspond to the concept of *trust propositions* defined in [8]. These are of the general form:

A *trusts* **B** *on matters of* **X** *at epoch* **T**

Here, *A* and *B* may be people, computers and their specific resources and services, or even small or large enterprises that admit to ***trust relationships***. In the proposition, *A* is placing a trust relationship (***dependence***) on *B* with respect to matters of *X*. Such matters constitute the set of ***rights and obligations*** of *A* with respect to *B*, such that *B* permits access to specific resources (services) provided by *B* to *A* provided that *A* fulfils specific *obligations* (*conditions*) laid down by *B*. Epoch *T* represents the period during which both *A* and *B* observe the well being of the their trust relationship without incidence of failure.

Trust propositions of this type underpin the semantics of the executable contracts (x-contracts) described earlier in Section 4.2. They tie together the issue of ***trust***, the dependences that inter-organisational parties place on the expected principles and behaviour of one another; the issue of ***security***, the assurances that both the integrity and (optionally) the privacy of inter-organisational interactions are maintained; and the issue of ***contractual bindings***, the explicitly certified agreements that inter-organisational parties make about their respective expectations of trust and security with respect to one another.

5.2. Goal and objectives

It is a principal goal of TAPAS to develop an architecture that enables the specification and implementation of dependable x-contracts, where the underlying objectives are two-fold:

5.2.1. Negotiable multi-party trust agreements

Such agreements are typically based on the respective reputations of the parties involved, where each reputation is a measure of previously observed trusted behaviour. In real life, initial trust by one party of another typically begins by recommendation of a trusted third party, or by blind faith. Examples of such trust include, *the purchase of a software product from a Web site* (will the company representing the Web site sell my private purchase information to other companies?), *the downloading of a software product to a personal computer* (will the software do what I expect it to do?), and, *engagement with an Internet Service Provider* (will the ISP automatically log my private traffic information?). Each of these situations, and many others, requires specific agreements on specific matters of trust by specific parties operating in specific roles. And, as such, these trust arrangements must be fairly negotiated in order to guarantee unambiguous and unanimous decisions.

TAPAS is accordingly investigating techniques for fairness in negotiable multi-party trust agreements, with specific reference to the requirements of role-based security policies for authentication, integrity and authorisation. Our early work in the area of fair exchange protocols is presented in the Appendix [9]. Moreover, this appendix also includes our initial ideas on formalising trust relations [10].

5.2.2. Certified dependable X- contracts

It is conjectured that the aforementioned negotiable agreements will establish the foundations for creating system enforceable contracts which commit all parties involved to respect certain obligations in return for certain rights; which, in this case, will be the specification of the security policies agreed and the corresponding encoded implementations of these policies to be enforced at run-time. Each such contract will become certified when all party members have countersigned it with their respective digital signatures, together with endorsements from the trusted third parties responsible for the provisioning of those signatures.

TAPAS is therefore also investigating *processes for reliably creating and safeguarding certified x-contracts* and for realising an effective *contract monitoring service* for auditing detected run-time deviations of contractual obligations and rights, and for reporting them to responsible system adjudicator services for suitable action.

Whilst the general architecture for VEs with x-contracts has been described, the following presents the specific ***Role-based Access Control*** (*RBAC*) ***Architectural Model*** for controlling inter-organisational interactions within VEs, mediated by x-contracts; since x-contracts are themselves trusted components, provision must be made to ensure that their implementations are dependable and thus trustworthy.

6 Trust enforcement

It is clearly not possible to prevent mutually suspicious enterprises within a VE from misbehaving and attempting to cheat on their agreed trust relationships. It is simply not possible to force an individual enterprise to behave respectably, simply because its internal management and resource behaviours are outside the control of any associated x-contracts!

The best that can be achieved is to ensure that all contractual interactions between such enterprises are funnelled through their respective x-contracts, and that all other non-contractual interactions are disallowed.

This can only be secured if all interfaces for interactions within a VE are via x-contracts. In this way, x-contracts serve as security firewalls: firstly, to protect unwanted interactions from outsiders not privy to the contract; secondly, to protect each legitimate enterprise in its required security actions for authentication, integrity, authorisation, and availability; and finally, to ensure that all contractual interactions are monitored and audited so that they may be inspected by all parties involved for integrity checks, and thereby allow the possibility for each party to refer disputes to appropriate VE adjudicators for resolution.

The TAPAS RBAC architectural model with x-contracts is intended to achieve this level of security for all inter-organisation interactions within a VE.

6.1. Secure RBAC with x-contracts

The intent of an access control system is the protection of a system and its resources against unauthorised access, disclosure, modification or destruction of its services and its information. This can only be accomplished by ensuring, among other things, that the ***identification*** and ***authentication of the legitimate users*** of system services and their

encapsulated resources are securely verified. The decision of which access controls to implement is not only based on organisation (enterprise) policies but also on two generally accepted standards of practice: ***separation of duties*** and the ***principle of least privilege***, where the former assigns ***roles to duties*** and ***entities to role-players***, whether such entities are human beings or machines; whilst the latter assigns ***only those privileges to role-players necessary to achieve their respective duties***. For such controls to be accepted and, therefore, used effectively, they must not be disruptive to normal work flow activities nor place too many burdens on the administrators, auditors, or authorised users of the system, especially within a VE. Accordingly, such controls must be kept to the indispensable minimum of effective simplicity and convenience.

This is the goal of the TAPAS RBAC architectural model described below.

6.1.1. RBAC model requirements

The provision of an extensible RBAC model for TAPAS is being designed to satisfy several requirements:

- Each enterprise within a VE is autonomously responsible for its own role management and role playing assignments, thereby ensuring that each enterprise controls its own people and resource management policies.
- Each x-contract unambiguously specifies for each enterprise the associated role playing managers and the rights and obligations of their assigned role players.
- Each x-contract is capable of authenticating the identities of each enterprise within a VE, its role playing managers and its role players, and vice-versa.
- Each x-contract ensures authorised access to rights by role players within the rules laid down by their associated obligations, and provides safeguards against unauthorised access.
- Each x-contract serves as a trustworthy custodian of the trust model shared by each enterprise in their respective VE.

These requirements are considered to be the minimum set necessary to realise an effective and usable RBAC scheme for secure inter-organisational interactions within VEs.

6.1.2. Components of the TAPAS RBAC Model

The TAPAS RBAC model is an extended form of the emerging NIST (National Institute of Standards and Technology) core RBAC standard [11]. These extensions are identified as *interfaces* and *obligations* in the RBAC component model shown in Fig. 14.

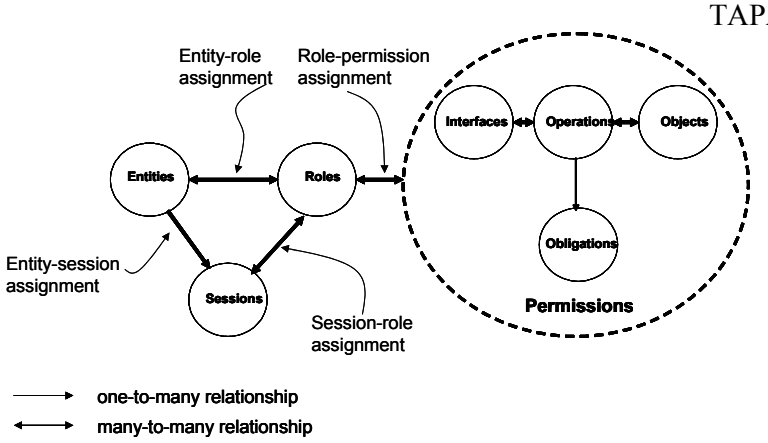

Fig. 14: Extended NIST Core RBAC Model

The basic concept of RBAC is that entities (users, machines, services, etc.) in each enterprise of a VE are assigned to roles, permissions are assigned to roles, and entities acquire permissions by being members of roles. In this instance, ***permissions comprise rights and obligations***, and ***rights comprise interfaces***, themselves ***comprising operations on objects*** (resources or services) of one or more other enterprises within the VE. It is highlighted here that ***each entity (role player) has an associated state machine***, as described earlier in Section 4.7. The number of entities, and therefore role players, associated with each state machine instance is termed a ***role set***, each of which joins[3] a ***session*** that identifies that role set's transaction. All members of a role set must first join a session before they may interact with an x-contract and conduct business. These state machine instances are maintained by the x-contracts of their respective VEs.

The basic NIST core RBAC model includes requirements that entity-role, session-roles and permission-role assignments can be many-to-many; this is represented in Fig. 14 by the double-headed arrows. Also, in this model each session is associated with a single entity; however, a given entity can be associated with one or more sessions (a one-to-many relation); once an entity is associated to a session, the entity can activate many roles. Similarly, for permissions, a single permission can be assigned to many roles and a single role can assigned to many permissions. Finally, it is required that each entity can if desired simultaneously exercise permissions of multiple roles, each of which belongs to some role set[4].

6.1.3. RBAC with digital signature authentication and authorisation

The proposed TAPAS RBAC scheme is based on a simple Public Key Infrastructure (PKI) which uses public key/secret key pairs for signing and verification.

While PKI standards are prevalent and well understood, the following summarises the essential characteristics of the scheme necessary to TAPAS RBAC controls in x-contracts:

---

[3] This join protocol is a subject of future research.

[4] The allowed concurrency instances of role set types permitted in x-contracts is a subject of future research.

- Each entity and thus role player defined in an x-contract is unambiguously identified by its Public Key Certificate (PKC), issued by its trusted Certificate Authority (CA).
- Each identified entity possesses a public/secret key pair, namely PK and SK respectively. A copy of the public key is always included in the entity's PKC, whereas the secret key is kept secret by the entity.
- Each secret key of an entity can be used to form a digital signature that can be verified by the use of the corresponding public key, using a standard digital signature validation process.
- Each entity registers (either directly or indirectly via an authorised proxy) with a trusted CA to obtain a PKC for the role it wishes to play. The PKC is digitally signed by the CA using its secret key; so the authenticity of the PKC can be digitally verified by the owner, and by any other interested entity, using a digital signature validation process and the CA's public key.

These characteristics can now be related to the principal identities and roles specified in X-contracts, and known to the respective enterprises of their associated VEs.

6.1.4. Principal role players in x-contracts

Five types of roles are identified for each x-contract and its associated VE:

1. The Enterprises (E_1, E_2 , ….., E_n) participating in the VE;
2. The specific Role Managers ($RM_{E1(1-n)}$, $RM_{E2(1-n)}$,….., $RM_{En(1-n)}$) of each participating Enterprise;
3. The Role Players ($RP_{RM1(1-n)}$, $RP_{RM2(1-n)}$,….., $RP_{RMn(1-n)}$) assigned specific roles by their respective RMs;
4. The Contract Manager (CM) responsible for enacting the x-contract;
5. The x-contract itself.

Each of these identity role types is securely defined by a PKC issued by its respective trusted CAs within each VE, of which there are four types:

i. The Contract Certificate Authority (CCA) responsible for issuing each enterprise's PKC_E and their contract manager's PKC_{CM} ;

ii. The Contract Manager Certificate Authority (CMCA) responsible for issuing its $PKC_{x\text{-}contract}$;

iii. The Enterprise Certificate Authority (ECA) responsible for issuing its role manager's PKC_{RMs};

iv. The Role Manager Certificate Authority (RMCA) responsible for issuing its role player's PKC_{RPs}.

These CAs and their issued PKCs are arranged hierarchically to form a trust chain for certificate validation, authentication and access rights authorisation, as shown in Fig. 15.

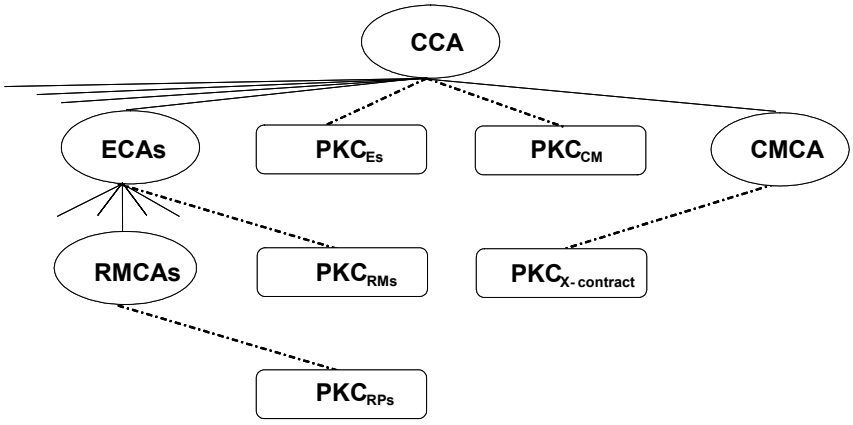

Fig. 15: Virtual Enterprise (x-contract) CA and PKC Hierarchies

6.1.5. Digital signature authentication and authorisation processes

When one role player (for example B) from one enterprise interacts with another role player from another enterprise, within the same VE, the former presents (inputs) its PKC certificate to the VE's x-contract, together with a statement of its specific right's instance, namely, the interface and the operation of the object it wishes to invoke on its target. The authentication process of the x-contract checks that the requesting entity is a legitimate party and that its request can be authorised. This process also checks that the requesting public/secret key identities are complementary.

Let us assume that B's PKC was issued by A. In the picture shown in Fig. 16, B's has already been received by the x-contract. B's PKC contains the following information:

- B: the name of the role player.
- PK_B: the public key of B as a role player.
- A statement of the specific right's instance, namely, the interface and the operation of the object it wishes to invoke on its target.

The above information is double signed: It contains DS_A, that is, the digital signature of the issuer of the certificate. The result of this is signed again, this time by the owner of the PKC, this is represented as DS_B

As shown in the figure, the x-contracts is in possession of a copy A's PKC. A's PKC contains the following information:

- A: the name of the role player.
- PK_A: the public key of the role player.

The above information is digitally signed with DS_O, that is, by the enterprise certificate authority responsible for issuing A's PCK that entitles A as a role manager.

As indicated by the arrowed line that goes from PK_A to DS_A, the x-contract can use A's public key to verify that DS_A is actually A's digital signature; correctness of DS_A would indicate that B's PKC was issued by A. Similarly, as indicated by the arrowed line that goes from PK_B to DS_B, the x-contract can retrieve PK_B from B's PKC and check that the PKC signed with DS_B conserves its integrity.

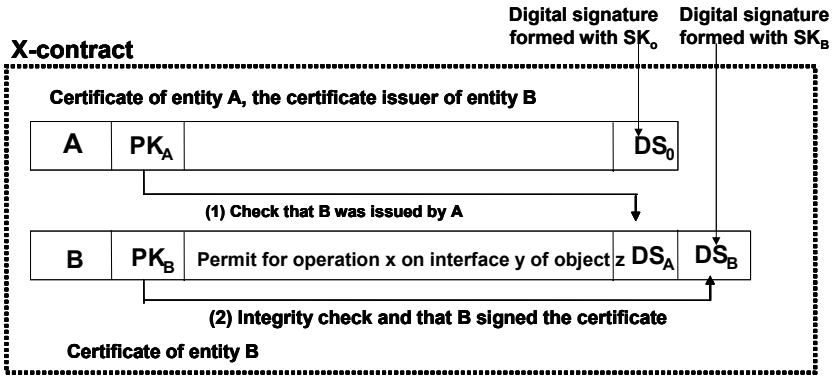

Fig. 16: X contract authentication and authorisation processes

6.1.6. The authentication protocol between enterprises and their Virtual Enterprise x-contracts

When an x-contract is activated (at the time it is scheduled) the following is assumed to be true:

- Each enterprise identified in the x-contract knows the PKC of that contract; and the x-contract knows the PKC of each enterprise.
- Each x-contract also knows the PKCs of all Role Managers within the VE, and each enterprise knows the PKC of the Contract Manager.

Such knowledge became true when the x-contract was negotiated and created.

The PKC state of one enterprise within a Virtual Enterprise (VE) and its x-contract are shown in Fig. 17, where the terms CM, XC, E and RM denote instances of the Contract Manager, the Executable Contract, the Enterprise, and the Role Managers respectively, within the VE.

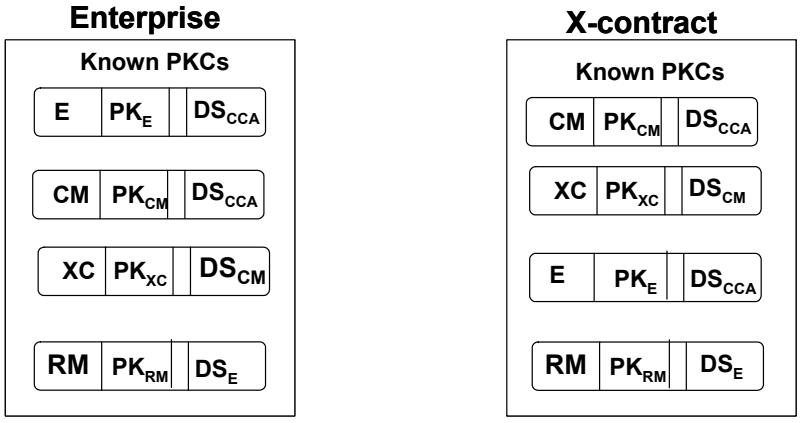

Fig. 17: X-contract and Enterprise known PKC states

Both the Enterprise and the X-contract know the public key of the Contract Certificate Authority (CCA) and are thus capable of following the certificate chain of trust from its root.

Given this situation, the following mutual authentication process between each enterprise (E) within a VE and specified by its associated X-contract (XC) is as follows:

` XC -> E : (Challenge (a once only value)

E -> XC : (Challenge, PKC_E)DS_E

XC -> E : (Challenge, $PKC_{X\text{-}contract}$)$DS_{X\text{-}contract}$

E <-> XC : ((OK), DS_{EP} < -> DS_{XC}))

This authentication protocol uses the same process of verifying two digital signatures as described earlier. Here, the authenticity (digital signatures) of E and XC are verified using the public keys defined in their respective PKCs. The certificates are respectively authenticated using the public keys of their issuing CAs (i.e., the CCA and the CMCA respectively). This protocol can of course be conducted over a secure channel for the purpose of confidentiality using any method of the IPSEC standard [12].

Likewise, a similar authentication protocol is performed by role managers and their assigned role players for each enterprise in a VE. The specific roles defined in an x-contract are associated with the PKC of its role manager. In this way, each specific role player in one enterprise interacts with a different role player in another enterprise by issuing a request to the x-contract of the form

Invocation-request (PKC_{RP}, session, required permission) DS_{RP}

PKC_{RP} is the certificate of the role player as described in Fig. 16. It identifies its issuing role managers PKC_{RM} and the required permission denotes by the (***interface, operation, object***) tuple.

The target enterprise receives a corresponding request of the form

then knowing that this request has been validated by the x-contract of the VE.

Again, such request transmissions can be conducted over private and thus confidential channels using symmetric rather than public key cryptography.

The ***session value*** noted here identifies the invocation context between one role-player instance and another role-player instance between their interacting enterprises and the contractual agreement laid down in their associated X-contract, within the same session, and without possibility of forgery. Each such ***session value*** for these contractually bound roles comprises a ***unique nonce*** identifying both roles based on a summary (MAC) on their respective PKC certificates, together with their assigned session number (common to all role pairs within the same session), undersigned by the X-contract's digital signature. This is the fundamental requirement for secure communications between the session participants in one enterprise, and the session participants in another enterprise both defined by their X-contract for a given VE. This session value will always remain secure, unless it is stolen, together with the secret key of the X-contract. The security of this method in general relies upon the secure safeguards of all secret keys for and within an X-contract – otherwise, the trust relationships of a VE will break down, and remain in place until their breach is discovered and rectified, but only by the authorised observers of the associated X-contract rules.

This principle uniquely underlies the nature of secure communications within VE's and across their associated X-contracts.

6.1.7. Other aspects of the TAPAS RBAC model

There remain two outstanding issues of importance, namely, ***replay attacks***, and ***Public Key Certificate (PKC) revocation*** as described below.

Reply attacks

Replaying previous authenticated invocation requests is the ploy (attack) perpetrated by the ***man-in-the-middle***. Prevention of such attacks can be ensured if each interaction between the enterprises in a VE via its X-contract always specifies a ***one-time-only value*** with each request, such as an increasing sequence number or time-stamp, signed by the legitimate transmitter's unforgeable digital signature.

PKC revocation

Since the management of roles and their role playing assignments is the responsibility of each enterprise in a VE, it seems natural to assume that each enterprise should be responsible for revoking such role management and role playing assignments, and disallowing the use of cancelled PKCs with respect to their X-contracts. It is essential for each enterprise of a VE to notify its X-contract whenever it revokes an active PKC manager or role player, so that it may close down the active associated session within it, and accordingly notify interested parties of this event, including its X-contract's legal parties.

6.2. Middleware for x-contracts: non-repudiable information sharing

Regardless of the details of the implementation, any software designed to implement an x-contract will contain two main components: a *contract monitor and enforcer* and a *middleware service* for regulated, non-repudiable information sharing.

The contract enforcer is a piece of software that guarantees that the rights and obligations stipulated in the contract are monitored and enforced. An example of obligation that can be enforced by this piece of software is *send offer within three days after signing the x-contract*.

The middleware service is a layer of software that regulates the interaction between two (or more) contracting parties who, despite mutual suspicion, wish to interact and share information with each other. Thus the middleware provides generic services that can be used to support arbitrarily complex interactions between contracting parties. From the viewpoint of each party involved, the overarching requirements are (i) that their own actions meet locally determined policies; and that these actions are acknowledged and accepted by other parties; and (ii) that the actions of other parties comply with agreed rules and are irrefutably attributable to those parties. These requirements imply the collection, and verification, of non-repudiable evidence of the actions of parties who interact with each other. An example of evidence that can be collected is a non-repudiable record that a payment was placed on a certain date.

The details of the contract monitor and enforcer were discussed at large in Section 4, where we proposed the use of FSMs for this purpose. The RBAC ideas presented here will have to be integrated with the approach presented in Section 4, and is the subject of further work. Next we discuss how to provide services for regulated, non-repudiable information sharing.

A promising appproach that seems to meet the requirements of the middleware service that we need is the B2Bobject middleware developed by us [13]. The B2Bobj middleware collects non-repudiable evidence about information sharing between parties that do not necessarily trust each other. Once deployed, each party holds a local copy of shared information encapsulated in objects. Access to and update of this information is subject to non-repudiable validation by each party. It is assumed that each organization has a local set of policies for information sharing that is consistent with the overall information sharing agreement between the organizations (this agreement will be encoded in the x-contract). The safety property of our system ensures that local policies of an organization are not compromised despite failures and/or misbehavior by other parties; whilst the liveness property ensures that if all the parties are correct (not misbehaving), then agreed interactions would take place despite a bounded number of temporary network and computer related failures.

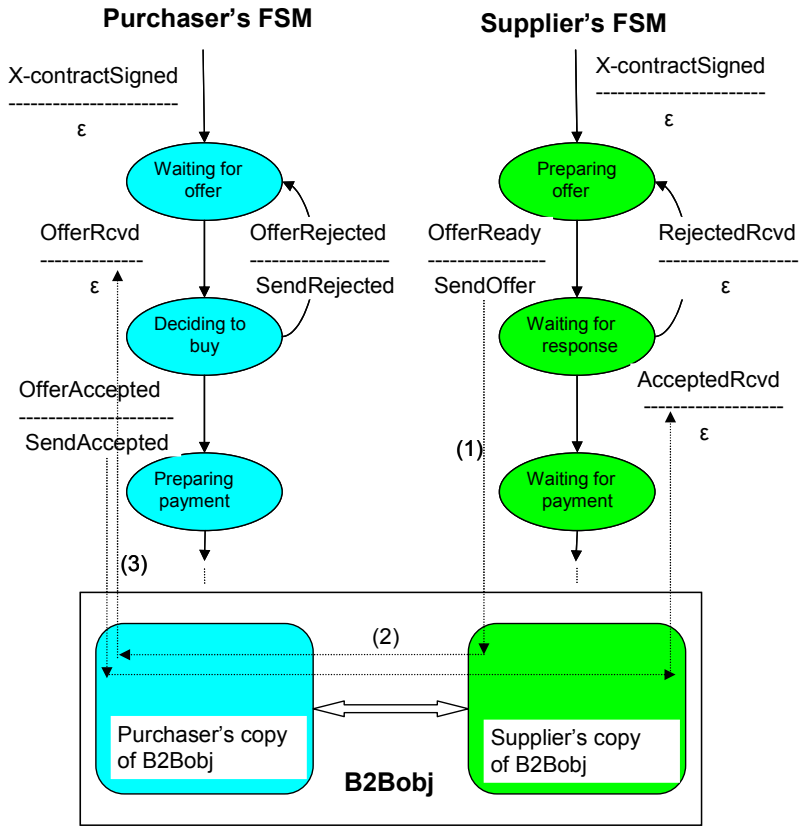

Fig. 18: Collection of non-repudiable digital evidence with a B2Bobj.

Essentially, our middleware resembles a transactional object replica management system where each organization has a local copy of the object(s) to be shared. Any local updates to the copy by an organization ("proposed state changes" by the organization) are propagated to all the other organizations holding copies in order for them to perform local validation; a proposal comprises the new state and the proposer's signature on that state. Each recipient produces a response comprising a signed receipt and a signed decision on the (local) validity of the state change. All parties receive each response and a new state is valid if the collective decision is unanimous agreement to the change. The signing of evidence generated during state validation binds the evidence to the relevant key-holder. Evidence is stored systematically in local non-repudiation logs. For protocol details, see [13].

With this much background, we can hint at the overall implementation of an x-contract. The implementation of a x-contract that involves a purchaser and a supplier is shown in Fig 18. As can be seen, the rights and obligations of each contracting party are described, monitored and enforced by two FSMs. Consequently, these two FSMs determine the behaviour of the x-contract. In addition to this, a B2Bobj is used for collecting non-repudiable digital evidence about the operations executed by the purchaser and the supplier.

The dashed line that goes from the supplier to the purchaser shows what happens when the supplier sends an offer. When the offer is ready, the supplier invokes a send operation; the supplier's FSM switches to its *Waiting for response* state and makes a *SendOffer* call to the local copy of a shared B2Bobj that implements the operation (1). The local B2Bobj collects, and signs, evidence of the operation and requests coordination of the proposed update to its

state with the purchaser's B2Bobj (2). The purchaser's B2Bobj verifies the evidence provided and makes an up-call to the purchaser's FSM to validate the B2Bobj operation (3). Upon receiving the up-call, the purchaser's FSM switches to the *Deciding to buy state*. The dashed line from the purchaser's FSM to the supplier's FSM shows how the purchaser's response is transmitted to the supplier. The B2Bobj middleware ensures that all operations performed by the purchaser and the supplier are recorded and are non-repudiable. One of the major advantages of B2Bobj is that it ensures this without the need of involving centralised trusted third parties.

7 Related work

7.1 Virtual enterprises

The term virtual enterprise is relatively new, it has been around only for the last ten years or so, consequently, it is still used by different authors to refer to different systems. The question about whether companies like Amazon.com that sell their products through the Internet are virtual enterprises would trigger a debate. What is not clear yet is when an organization should be considered virtual. This question is addressed in [14]. In this work, the authors classify the existing implementations of virtual enterprises into six categories:

- Virtual faces: In this group of arguably virtual enterprises fall all the companies that have extended their customer windows to the Web. Basically, they offer the same service as they offer over the phone, fax and face-to-face.
- Co-alliance: In these virtual enterprises each composing enterprise brings approximately equal amount of resources. There is not a clear leader in the enterprise.
- Star-alliance: In these enterprises the existence of a dominant enterprise and group of two or more satellite enterprises is evident.
- Value-alliance: The main feature of these enterprises is the existence of a well-defined supply chain. The existence of the chain strengthens interdependence.
- Market alliance: These enterprises typically bring together a range of services, provided by the composing enterprises, in a single package.
- Virtual brokerage: A virtual broker is basically an ad hoc enterprise built with the intention of capturing the value of a short term (for example, at Christmas time) market opportunity.

It can be argued that the there are not clear cut ways between the six categories of virtual enterprises. However, this work is a good starting point for discussing virtual enterprises. It is relevant to our project because it gives an idea about practical implementation of virtual enterprises. It is relevant to mention that thanks to its high level of abstraction, the model for virtual enterprises that we introduced in Section 3 captures all the six categories of virtual enterprises discussed above.

The category of virtual enterprise that deserves additional discussion is the virtual brokerage. We argue that among the six categories, the virtual brokerage is the most general and

challenging as it suggests dynamic creation of virtual enterprises. Dynamic virtual enterprises are an emerging category of virtual enterprises [15, 16]. As one can guess, dynamic creation of virtual enterprises implies dynamic negotiation of business contracts, and dynamic creation of executable contracts (see Section 4.2). Although these concepts sound exiting and promising for enhancing the architecture of our project, we decided to leave out of our current research interest. We might have an interest on them in the future when we consider that static negotiation and creation of virtual enterprises is well-understood.

7.3 Contract Representation, Monitoring and enforcement

In this report (see Section 4) we use finite state machines as a formal notation for describing business contracts. We are aware that FSMs are just a promising alternative. Other researchers are experimenting with different approaches.

Monitoring and controlling electronic transactions is addressed by Minsky et al in a number of papers on Law Governed Interaction (LGI) [17]. LGI is an infrastructure that allows members of a group to interact using agents, where agents are entities that interact with each other. A *policy* in LGI is defined as four-tuple: (M,g,CS,L) where: M is the set of messages regulated by this policy; g is an open and heterogeneous set of agents that exchange messages belonging to M; CS is a mutable set of control sates, one (CSx) for every member (x) of group g; L is an enforced set of rules that regulate the exchange of messages between members of g.

Law enforcement is achieved as follows: the law L is enforced by a set of trusted entities called controllers that mediate the exchange of messages (M) between members of group g. For every active member x in g, there is a controller Cx placed between x and the communication medium. Every controller carries the law L. The controller Cx assigned to x computes the ruling of L for every event at x, and the ruling is carried out locally. Controllers act similarly to our Contract Enforcer (the FSM), which enforces the agreed contract, and regulates interactions between the parties.

Another work of relevance to contract monitoring and enforcement is the Ponder Policy Specification Language [18]. Ponder is an object-oriented declarative language for specifying management and security policies for distributed systems. In other words, Ponder can specify, monitor and enforce what actions (operations on objects) are permitted within a system, who can invoke the actions and under which conditions. Ponder comes with a toolkit for editing, compiling and managing policies, that can be downloaded from its Web page at the Department of Computer Science of the Imperial College in London. As mentioned above, ponder was designed to govern actions executed within a system, it is not clear to us whether its semantic is descriptive enough to regulate tight interactions between two or more independent systems, namely, between the members of a virtual enterprise.

7.2 Trust, trust models and RBAC

Section 5 of this document presented the notions of trust, trust models and the security principles required by the TAPAS RBAC architectural model. We are optimistic that the RBAC model architecture developed in OASIS [19, 20] is a promising approach to this

requirement. OASIS, with its notions of roles, sessions, its role principals, together with its assigned role-players seems to fit well with the principles of the TAPAS security architecture presented herein. Moreover, OASIS with its extended notions of appointments, for delegation of role-playing, and of multiple, mutually aware domains for mobile roles, re-located and still able to communicate without confusion, is entirely relevant to TAPAS. Such notions of mobility and multiple communication domains are subjects of future research in TAPAS, just as are the engineering model of OASIS and its engineering principles for TAPAS x-contracts and the enterprises within Virtual enterprises (VEs).

References

[1] TAPAS project, Annex 1- Description of Work

[2] D. Lamanna, J. Skene and W. Emmerich, "Specification language for Service Level Agreements", TAPAS Deliverable, D2, March 2003.

[3] Giovanna Ferrar and Giorgia Lodi, "TAPAS Architecture: QoS Enabled Application Servers", TAPAS Deliverable, D7, March 2003.

[4] Carlos Molina-Jimenez, Santosh Shrivastava, Ellis Solaiman and John Warne, "Contract Representation for Run-time Monitoring and Enforcement", Report, School of Computing Science, University of Newcastle Upon Tyne, Jan 2003. (Also, in the Appendix, this report)

[5] Gerard J. Holzmann. "Design and Validation of Computer Protocols", Prentice Hall, 1991.

[6] J. C. Laprie (Editor), Dependability: Basic Concepts and Terminology, Dependable Computing and Fault-Tolerant Systems, Vol. 5, Springer-Verlag/Wien, ISBN 3 211 82296 8.

[7] T. Grandison and M. Sloman, "A survey of trust in Internet applications", IEEE Communications Surveys, Fourth Quarter 2000,

[8] E. Gerck, *Towards Real-World Models of Trust: Reliance on Received Information*, published on 23rd June 1998 in the mcg-talk list server.

[9] Paul D Ezhilchelvan and Santosh K Shrivastava, "Systematic Development of a Family of Fair Exchange Protocols", Report, School of Computing Science, University of Newcastle Upon Tyne, June 2002. (Also, in the Appendix, this report).

[10] Nicola Mezzetti, "Modelling Trust in Collaborative Environments", Dept. of Computer Science, University of Bologna, Internal publication, Feb 2003. (Also, in the Appendix, this report).

[11] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed NIST standard for Role-Based Access Control", ACM transactions on Information and System Security, Vol. 4, No. 3, Aug. 2001.

[12] IETF RFC24 document series

[13] N. Cook, S.K. Shrivastava, and S.M. Wheater, "Distributed Object Middleware to Support Dependable Information Sharing between Organisations", Proc. IEEE Int. Conf. on

Dependable Syst. and Networks (DSN-2002), Bethesda USA, June 2002. (Also, extended version in the Appendix, this report).

[14] Janice Burn, Peter Marshall and Martyn Wild, "Doing Business on the Internet", Edited by Fay Sudweeks and Celia T. Romm, Chapter 3, Springer, 1999.

[15] Vaggeis Ouzounis and Volker Tschammer, "Towards Dynamic Virtual Enterprises", Towards the E-society: The First IFIP Conference on E-commerce, E-Business, E-Government (I3E 2001), Octover 3-5, 2001, Zurich, Switzerland, Kluwer Academic Publisher.

[16] Ricardo J. Rabelo and Rolando V. Vallejos, "A Semi-Automated Brokerage for a Virtual Organization of Mould and Die Industries in Brazil", Towards the E-society: The First IFIP Conference on E-commerce, E-Business, E-Government (I3E 2001), Octover 3-5, 2001, Zurich, Switzerland, Kluwer Academic Publisher.

[17] Naftaly H. Minsky, Victoria Ungureanu, "Law-Governed Interaction: A Coordination and Control Mechanism for Heterogeneous Distributed Systems". ACM Press, New York, NY, USA, TOSEM 9(3): 273-305, 2000.

[18] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "The Ponder Policy Specification Language", in Proc. Int. Workshop on Policies for Distributed Systems and Networks (POLICY), Bristol, UK, Springer-Verlag LNCS 1995, Jan. 2001.

[19] Jean bacon, Ken Moody and Walt Yao, "Access Control and Trust in the use of Widely Distributed Services", IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001), November 2001, Heidelberg,, Lecture Notes in Computer Science. VOL. 2218, pp. 300-315.

[20] W. Yao, K. Moody and J. Bacon, "A Model of OASIS Role-Based Access Control and its Support for Active Security", ACM Trans. On Information and System Security, 5, 4, November 2002.

Appendix