TAPAS

IST-2001-34069

Trusted and QoS-Aware Provision of Application Services

TAPAS Architecture

Report Version: Deliverable D6

Report Delivery Date: 31 March 2005 (Revised 30 April 2005)

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

Santosh Shrivastava and Carlos Molina-Jimenez
School of Computing Science, University of Newcastle upon Tyne,
Newcastle upon Tyne, NE1 7RU, England

Table of Contents

Appendix

[1] Carlos Molina-Jimenez, Santosh Shrivastava and John Warne, “A Method for Specifying Contract Mediated Interactions”, Technical Report, University of Newcastle upon Tyne, 2005. (submitted for publication)

[2] Carlos Molina-Jimenez, Jim Pruyne and Aad van Moorsel, “Software Architectures for Service Level Agreements and Contracts”, To appear in "Architecting Dependable Systems III", to be published in the LNCS series by Springer in 2005.

[3] Paul Robinson, Nick Cook and Santosh Shrivastava, “Implementing Fair Non-Repudiable Interactions with Web Services”, Technical Report, University of Newcastle upon Tyne, 2005. (submitted for publication)

[4] Graham Morgan, Simon Parkin, Carlos Molina-Jimenez and James Skene, “Implementing the Monitoring of Service Level Agreements”, Technical Report, University of Newcastle upon Tyne, 2005. (submitted for publication)

[5] Michael Dales, Antonio Di Ferdinando, Paul Ezhilchelvan and Jon Crowcroft, “An Active Network Measurement Service for TAPAS Group Communication”, Technical Report, University of Newcastle upon Tyne, 2005. (paper in preparation)

Summary

Deliverable D6 is a revised version of first year deliverable, D5 “TAPAS Architecture: Concepts and Protocols” and its aim is to describe the final version of the TAPAS architecture for application hosting. The TAPAS project is interested in developing solutions to the problem faced by 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. Three key requirements for application service provisioning have been identified.

1. Enhancing the application hosting middleware platform to be QoS aware. This way, hosting platform will be better equipped to meet the requirements of the hosting applications. In the absence of such a feature, the only alternative available to an ASP is *over provisioning*, which is not particularly desirable.

2. Ability to ensure that all inter-organisation interactions are strictly according to the terms and conditions contracts in force. In the worst case, violations of agreed interactions are detected and notified to all interested parties; for this, an audit trail of all interactions will need to be maintained.

3. Ability to demonstrate that hosted applications are meeting the various QoS requirements of SLAs.

These three requirements underpin the design of the TAPAS architecture.

1. Introduction

The TAPAS project is interested in developing solutions to the problem faced by 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. As explained in the first year deliverable report D5 [1], essentially this means that an ASP should be capable of hosting Virtual Organisations (VOs), meaning, it should be capable of providing facilities for forming and managing VOs. We define a Virtual Organisation (VO) as a strategic alliance among a group of cooperating organisations that share services electronically – say using Web/Internet technology – for the accomplishment of a set of mutually beneficial business goals; these arrangements being made such that each organisation continues to maintain its own autonomy, except for the mutually agreed undertakings of the alliance.

A central requirement of VO operational management is to enable organisations to regulate access to their service resources in a manner, which honours their individual resource sharing policies both securely and with integrity. Regulating such access is made difficult since each potentially accessible organisation might not unguardedly trust the others. Accordingly, all organisations within a VO will require their interactions to be strictly controlled and policed. There will therefore be a need for all business process relationships to be underpinned by guarded trust management procedures.

In the TAPAS project, we have taken the view that to form and automatically manage partnerships within a VO underpinned by guarded trust management procedures, it will be necessary to have electronic representations of contracts that can be used to mediate the rights and obligations that each interacting entity promises to honour. In the worst case, violations of agreed interactions are detected and notified to all interested parties.

The auction demonstrator application developed within the project [2] is a good example, illustrating various contracts involved. The application is a *sealed reverse auction* that is used by a car manufacturer (Toyota, Ford, etc.) for buying car parts (e.g. tyres, radiators, mirrors, etc.) from car part suppliers. Figure 1 shows the parties that participate in our demonstration scenario. We will discuss the roles played by each party first and latter on we will discuss their contractual business relationships, which are represented by double-headed arrows in the figure.

- **The auctioneer** is the representative of a car manufacturing company that at a given time runs one or several instances of the auction with the purpose of buying car parts from car part suppliers; for example, he might run an instance of the auction for buying seats and another one for buying batteries. We assume that he does not want to be disturbed with computer-related issues, thus, he relies on somebody else to provide the infrastructure to run the auction; the business related activities which the auctioneer performs include selecting and sending invitations to potential bidders, opening and closing of bid rounds, declaring winners and so on.

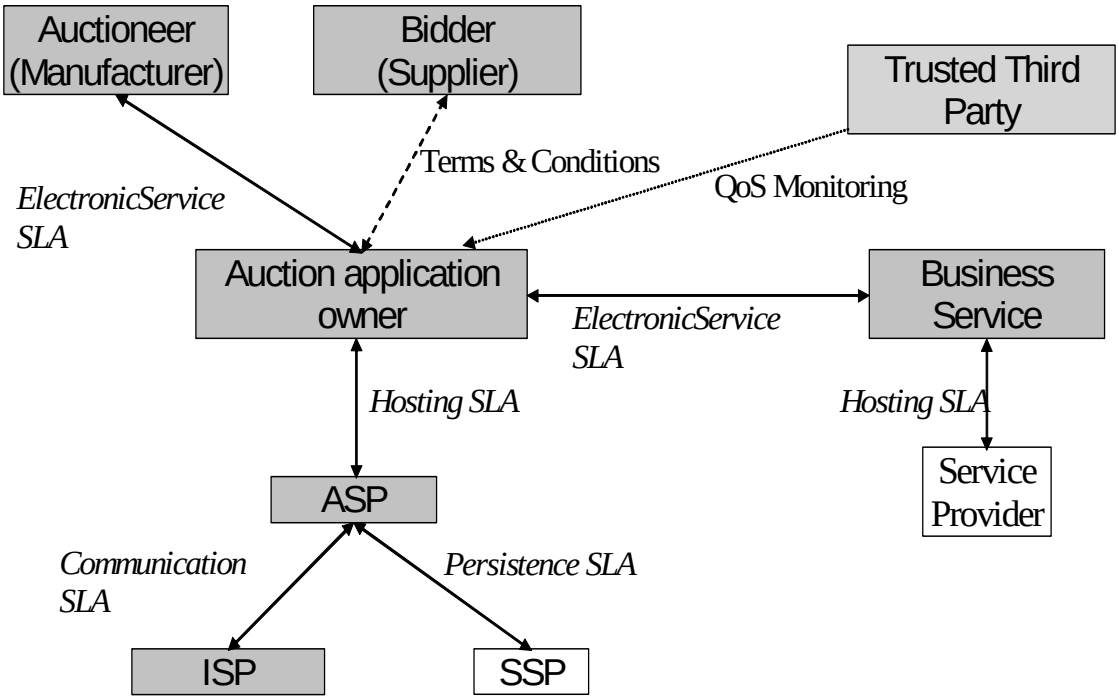


Fig 1. Auction scenario with its participants and their contractual relationships.

- **The bidders** are car part suppliers interested in selling their products to the auctioneer. In our scenario we consider that the number of bidders can be in the order of several hundreds.

- **The auction application owner** is the enterprise that offers the auctioneer the auction application ready to use; we can conceive it as an enterprise that owns a source code of the auction software and builds on-demand customized and ready to use copies of it to different auctioneers. The auction application owner is not involved in business related activities but in technical ones only; its responsibilities include running and tuning the auction software. We assume that the auction application owner does not have the ancillary resources to run his auction software, thus he relies on somebody else to host it.

- **The Application Service Provider (ASP)** is an enterprise that offers hosting services to the auction application owner. It provides all the necessary ancillary resources (CPU, databases, ISP, human, etc.) to host the auction software. The assumption is that at a given time the ASP might be hosting several instances of the reverse sealed auction that belong to the one or several auctioneers as well as other applications of arbitrary nature.

- **The Internet Service Provider (ISP)** is an enterprise that offers Internet connectivity to the APS so that the auction can be reached by other interested parties of the auction scenario.

- **The Storage Service Provider (SSP)** is an enterprise that offers disk space to the ASP.

- **The business service** is an enterprise that offers credit-rating services, or any other ancillary service, such as billing, to the auction application owner.

- **The Trusted Third Party (TTP)** is an enterprise with enough credentials to act as a trusted third party. It measures the performance of a party, assesses it and determines whether the party is honouring its contractual obligations with respect to another party. For the sake of simplicity, Fig. 1 shows the TTP monitoring the contractual obligation only between the auctioneer and the auction application owner; however, a TTP can and should be deployed between any pair of business partners such as the auctioneer and bidder$_1$, and the auction application owner and the ASP.

It is time now to discuss the meaning of the double-headed arrows that join the participants of our demonstration scenario shown in Fig. 1. A crucial assumption in our scenario is that the participants are autonomous and independent organizations that besides being mutually suspicious still want to conduct business together; because of the existence of this degree of mutual mistrust each pair of business partners needs to rely, as in conventional business, on legal business contracts to regulate their business interactions.

If in conventional business there are contracts for different commercial agreements (contracts for the rent of a house, contracts for loan of machinery, etc.) in our demonstration scenario, we identify five different types of contracts, namely, terms and conditions contracts, electronic service level agreement (SLA) contracts, hosting SLA contracts, communication SLA contracts and persistence SLA contracts. From a structural view, the five contract types are rather similar: all of them contain a header (signatories' names, addresses, signatures, start and end date, etc.) and clauses that stipulate the rights and obligations of each signatory party, however, from the point of view of the content of their clauses they are different:

- **Terms and conditions contracts** describe the relationship between the application owner and the bidders, i.e. the suppliers, specifying business action interactions. They specify 'business conversations' constrained by permissions, obligations, prohibitions, actors (agents), time constraints, and message type checking. We define a *conversation* as a small business activity executed between two or more business partners to perform a well defined task, such as submit a bid, issue a purchase order, process payment, refund money, cancel purchase order, etc.

- **Electronic service SLAs contracts** stipulate the QoS expected from the interaction between the auctioneer/auction application owner pair and the auction application owner/credit rating service pair. For the first pair, the electronic service SLA will contain clauses such as "the auction application owner shall guarantee that even during peak periods the invocation of the place_bid operation is successfully completed within two seconds when there are less than 100 users logged in"; for the second pair the electronic service SLA will contain clauses such as "the auction application owner shall never place more that 50 request per second".

- **Hosting SLAs contracts** are used to specify the QoS between the ASP and the application owner. They contain the objectives of the electronic service SLA because the application owner will be eager to delegate the objectives to other providers. Due to the more technically oriented relationship between application owner and ASP, the

hosting SLA contains as well technical objectives such as memory space and utilisation regulations for technical services such as user management, maintenance windows etc.

- **Communication SLAs contracts** specify QoS objectives for the relationship between ASP and ISP.
- **Persistence SLAs contracts** are used to specify QoS objectives for data storage service offered by an SSP.

The distinction of different types of contracts is relevant because we have learnt that the concepts and technology needed to represent, monitor and enforce a contract varies depending of the contract type.

2. TAPAS Architecture

Based on the above discussion, we identify the following three key requirements for application service provisioning.

1. Enhancing the application hosting middleware platform to be QoS aware. This way, hosting platform will be better equipped to meet the requirements of the hosting applications. In the absence of such a feature, the only alternative available to an ASP is *over provisioning*, which is not particularly desirable.

2. Ability to ensure that all inter-organisation interactions are strictly according to the terms and conditions contracts in force. In the worst case, violations of agreed interactions are detected and notified to all interested parties; for this, an audit trail of all interactions will need to be maintained.

3. Ability to demonstrate that hosted applications are meeting the various QoS requirements of SLAs.

These three requirements underpin the design of the TAPAS architecture. Figure 2 shows its main features. If we ignore the three shaded/patterned entities (these are TAPAS specific components), then we have a fairly 'standard' application hosting environment: an application server constructed using component middleware (e.g., J2EE). It is the inclusion of the shaded/patterned entities that makes all the difference.

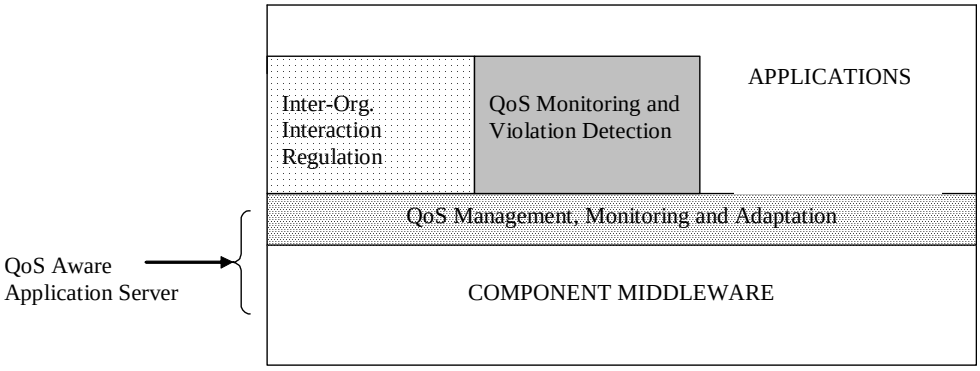


Fig. 2. TAPAS Architecture

The QoS management, monitoring and adaptation layer is intended to make the underlying application server QoS enabled (requirement 1). It is responsible for reserving the underlying resources necessary to meet the QoS requirements of applications hosted by that application server, and monitoring the reserved resources, and possibly adapting resource usage (e.g., reserving some more) in case the QoS delivered by these resources deviates from that required by the applications. Final year deliverable report D11 [3] describes in detail the design and implementation of QoS aware application server.

All cross-organisational interactions performed by applications are policed by the Inter-Organisation Interaction regulation subsystem (requirement 2). First year deliverable D5 [1] described how relevant aspects of contracts can be converted into electronic contracts (x-contracts) and represented using state machines and role based access control (RBAC) mechanisms for run time monitoring and policing. Second year deliverable report D9 [4] described how component middleware can be enhanced to incorporate non-repudiable service interactions providing audit trails of service interactions. These ideas are further developed and summarised in section 3 of this report. This subsystem could be provided by the ASP or one or more trusted third parties.

It is necessary to be able to demonstrate that a hosted application actually meets the QoS requirements (e.g., availability, performance) stated in the hosting contract SLAs (requirement 3). For this reason, we need an application level QoS monitoring service, which must also measure various application level QoS parameters, calculate QoS levels and report any violations. That is the function of the third subsystem shown in the figure. Second year deliverable report, D10 [5] described the design and implementation this subsystem. In TAPAS, QoS requirements in SLAs are specified using the SLAng language described in deliverable report D2 [6]. Section 4 of this report summarises the main ideas. This subsystem could be provided by the ASP or one or more trusted third parties.

3. Inter-Organisation Interaction regulation

All cross-organisational interactions performed by applications are policed by the Inter-Organisation Interaction regulation subsystem. This subsystem could be provided by the ASP or one or more trusted third parties. Each enterprise expects access to other's services. An operation on a service is allowed only if it is permitted by the rules of the contract and then only if it is invoked by a legitimate role player of a participating enterprise. Thus, a contract is a mechanism that is conceptually located in the middle of the interacting enterprises to intercept all the contractual operations that the parties try to perform. Intercepted operations are accepted or rejected in accordance with the contract clauses and role players' authentication. Our approach is to represent service interactions as finite state machines and make use of role based access control mechanisms for authenticated access. In the deliverable report D5, we described how contract clauses can be converted into finite state machines (FSMs). These ideas are further developed here.

Inter-Organisation Interaction regulation subsystem has two main layers (see figure 3). The contract monitoring and enforcement layer makes use of the services of the underlying layer that provides trusted coordination.

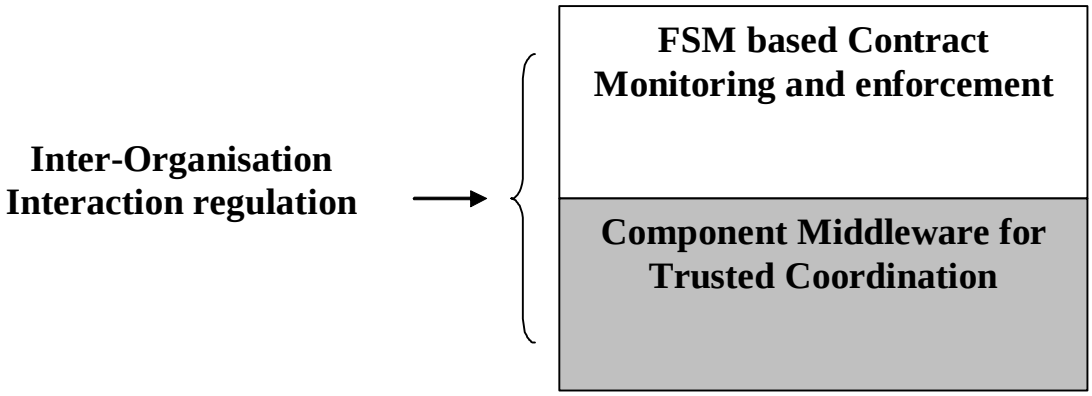


Fig.3. Inter-Organisation Interaction regulation

To regulate the interactions involved, a given action must be attributable to the party who performed the action and commitments made must be attributable to the committing party. For example, it should not be possible for a client to subsequently disavow the request and/or consumption of a service. Similarly, it should not be possible for the service provider to subsequently deny having delivered the service. If information is shared then the parties sharing the information should be able to validate a proposed update, the update should be attributable to its proposer and the validation decisions with respect to the update attributable to the other parties. That is, to regulate an interaction we require attribution, validation and audit of the actions of the parties involved. Non-repudiable attribution binds an action to the party performing the action. Validation determines the legality of an action with respect to interaction agreements. Audit ensures that evidence is available in case of dispute and to inform subsequent interactions.

3.1. Contract Monitoring and enforcement

It is necessary to have electronic representations of contracts that can be used to mediate the rights and obligations that each interacting entity promises to honour. This requirement implies that the original natural language contract that is drawn by lawyers and other non-technical people has to undergo a conversion process from its original format into a piece of executable code or *executable contract* (*x-contract* for short) that works as a mediator of the business conversations. This conversion process involves the creation, with the help of a formal notation, of one or more computational models of the contract with different levels of details.

We describe a general method of representing business interactions as FSMs using a widely used modelling language Promela [7] and discuss how it can be used to represent permissions, obligations, prohibitions, actors (agents), time constraints, and message type checking; that is, all the basic parameters that compose most typical business contracts. Our motivations for using Promela here is that such a representation can be validated with the help of the accompanying Spin model-checker tool [8].

We propose two levels of contract representation. (i) *Implementation neutral*: free of technical details related to technology-related interactions; in other words, specifying only business action interactions (for example, issue a purchase order, send payment, etc.). Such a description can be model checked and used for improving the original natural language

contract to be free from various forms of inconsistencies as discussed in our earlier work [9]. (ii) *Implementation specific*: a representation (also amenable to model checking) that is a refinement of the former to include technical details such as acknowledgements and synchronization messages that form an important part of any implementation; the details will vary depending upon the implementation techniques and standards that are selected (e.g., Rosettanet [10]). Such a representation can be used by technical people for implementing the actual x-contract for business conversation mediator.

3.1.1. Deployment models for contract enforcement

Conceptually speaking an x-contract is placed in between the two business partners so that it can observe their business interactions. Deployment can be either *centralized* or *distributed*. Further, the x-contract could be *reactive* or *proactive*, giving us four deployment models discussed below, where for illustration purposes we assume an interaction from buyer to seller:

(i) *Reactive Central*: The contract is deployed in a trusted third party (TTP), see Fig. 4(a). The job of the x-contract here is to intercept (1) and analyze (2) the messages exchanged between the two business partners; correct messages are forwarded (3) to their final destination whereas incorrect ones are dropped (3').

(ii) *Proactive Central*: The contract is deployed in a TTP, see Fig. 4(b); the contract is proactive in that it coordinates the conversational interactions between organisations by invitation only. It sends (1) an invitation message to the business partner; the response is received (2) by the x-contract and analyzed (3); correct messages are forwarded (4) to the seller, whereas incorrect ones are dropped (4').

(iii) *Reactive Distributed*: Distributed version of reactive central: the contract is split and deployed in two TTPs, see Fig. 4(c).

(iv) *Proactive Distributed*: Distributed version of proactive central.

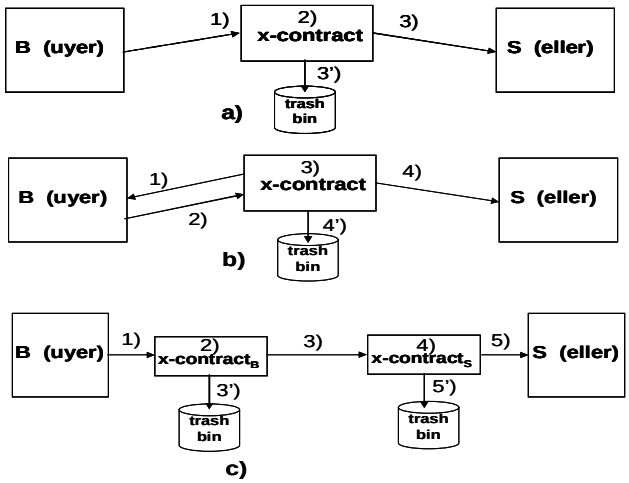


Fig.4. Deployment models (a) reactive central (b) proactive central and c) reactive distributed.

Which particular model is suitable in a given VO setting is a very interesting research problem, worthy of further investigation. We note that distributed deployments face the difficult challenge of keeping contract state information synchronised at both ends. For example, a valid message forwarded by the buyer's x-contract could be dropped at the seller's end because intervening communication delays render the message untimely (and therefore invalid) at the seller side. State synchronisation is necessary to ensure that both the parties either agree to treat the message as valid or invalid. Non-repudiable state synchronisation protocol using B2Bobjects, discussed in deliverables D5 and D9 provides such synchronisation (see also section 3.2).

3.1.2. Formal representation of electronic contracts

We assume that an x-contract is composed out of $N \geq 1$ conversations. We define a *conversation* as a small business activity executed between two business partners to perform a well defined task, such as issue a purchase order, process payment, refund money, cancel purchase order, etc.

The challenge here is to find a convenient formal notation that captures all or at least the most important parameters present in most business conversations. It is common knowledge that business contracts can be abstracted as a set of permissions (P), obligations (O) and prohibitions (F) that are expected to be fulfilled by actors (also called agents or role players) for the benefit of others by means of performing (or not performing) operations (also called actions). We define a *permission* as an action that an actor, for example a buyer or a seller, can perform if she wishes to; for instance, "The buyer can use his discretion to send a purchaser order to the seller" is a buyer's permission for the benefit of the seller. Likewise, an *obligation* is defined as an action that an actor is expected to perform; an example of a seller's obligation for the benefit of the buyer is "The seller is obliged to respond to the buyer within three days after the receipt of the purchase order". A *prohibition* is an action that an actor is not expected to perform; an example of a seller's prohibition for the benefit of the buyer is "The seller shall not send offers to the buyer unless they are requested". The execution of a permission operation is optional in the sense that there are no penalties for not executing it; conversely, a failure to execute an obligation operation and daring to execute a prohibited operation is considered a contract violation and the offending actor may be subjected to a *sanction*. A sanction can take different forms, for instance, it can grant the offended actor a permission (for example, the permission to charge and extra 10% on top of the original price) the offending actor can be refused a permission (for example, as from now, the buyer is not allow to pay by American Express card) or the offending actor can be assigned a new obligation (for example, the obligation to pay a fine). In several applications, permissions, obligations and prohibitions are dischargeable as they become and cease to be in effect depending on the occurrence of events.

A promising approach for modelling the concepts we have just discussed is Deontic Logic, a formal notation that is sometimes referred to as the logic of permissions, obligations and prohibitions. The form of Deontic Logic that has been thoroughly studied is the Standard Deontic Logic (SDL) of von Wright [11]. However, as argued in [12], SDL can precisely describe impersonal ought ("it ought to be that the payment is sent") but it is not expressive enough to describe situations where actions are assigned to specific agents ("it ought to be

that the payment is sent by Alice"). Another limitation of SDL is that it is static in the sense that it cannot describe permissions, obligations and prohibitions that become and cease to be in effect depending on the occurrence of time and other events. We are aware that currently there are several researchers exploring the possibility of enhancing SDL with additional logical constructs to overcome its limitations; for instance, there are suggestions to mix constructs from SDL with constructs from Modal Logic, Temporal Logic, Logic of Action or from their combinations. These combinations result in hybrid logic systems that can certainly express complex situations; unfortunately, as pointed out in [13], such logical systems have not yet been thoroughly studied and understood, consequently, the logical rigour of a contract expressed in such notations is questionable.

Our view is that it will take time for Deontic Logic approaches to reach a degree of maturity where the contract designer can automatically verify the correctness of his notation by proof-theoretical means or model checking. This fact discouraged us from using Deontic Logic notation to describe our contracts and motivated us to resort to Promela, perhaps a less elegant, yet a practical solution that is widely used for protocol specification and validation.

3.1.3. Implementation neutral representation: an example

In this section we will discuss an example part of very small (hypothetical) business contract that stipulates business action interactions between an auctioneer and a bidder for the purchase of goods.

1 *Offer to bid*

1.1 The auctioneer may use his discretion to send invitations to bid to the bidder.

1.2 The bidder is obliged to confirm acceptance or rejection of the invitation within 24 hrs of receiving the invitation.

2 Payment for participation in the auction

2.1 The auctioneer is obliged to send an invoice for participation to an accepting bidder within 3 days of receiving an acceptance notification.

3 Invalid messages

3.1 The auctioneer and the bidder are forbidden to send invalid messages.

4 Sanction

4.1 Failures to honour obligations and prohibitions will result in fines equal to 20 euros. The offended party shall be granted permission to issue an invoice notification to the offending party.

4.2 Failure to respond to a fine shall be sorted out outside this contract.

5 Synchronization and handling of transaction failures

5.1 Should the auctioneer and/or the bidder detect a technical failure that prevents them from continuing the normal course of a transaction, they are obliged to send a failure notification message by any other means.

Table 1 lists the permissions, obligations and prohibitions that compose the contract: P stands for permission, O for obligation and F for prohibition (forbidden). The number after P, O and F is the number of the clause in the contract from where the permission, obligation or prohibition was extracted. Notice that in the contract, clause 3.1 specifies a prohibition for the auctioneer and for the bidder; to distinguish between these two cases we named them $F3.1_A$ and $F3.1_B$, respectively. Similarly, $P4.1_A$ and $P4.1_B$ stand for permission for the auctioneer and the bidder, respectively, extracted from clause 4.1.

Permissions	**Subject**	**Beneficiary**	**Sanction**
P1.1 Send invitation to bid.	auctioneer	bidder	none
$P4.1_A$ Issue invoice to fine.	auctioneer	bidder	none
$P4.1_B$ Issue invoice to fine.	bidder	auctioneer	none
Obligations			
O1.2 Send confirmation.	bidder	auctioneer	$P4.1_A$
O2.1 Issue invoice to participate.	auctioneer	bidder	$P4.1_B$
Prohibitions			
$F3.1_A$ Send invalid messages.	auctioneer	bidder	$P4.1_B$
$F3.1_B$ Send invalid messages.	bidder	auctioneer	$P4.1_A$

Table 1. Permissions, obligations, prohibitions and sanctions.

As it is, the above contract might not be detailed enough for the technical people in charge of creating its executable version, yet it contains information of great value for this stage, for instance, it has enough information to begin reasoning about correctness of the contract. We believe that reasoning about the contract at this early stage is important because, text contracts are very likely to contain inconsistencies, to detect them we need to convert into a formal notation (a computational model of the contract) and validate it, perhaps with the help of automatic software tools. To illustrate our ideas, we will build a *reactive central* Promela model.

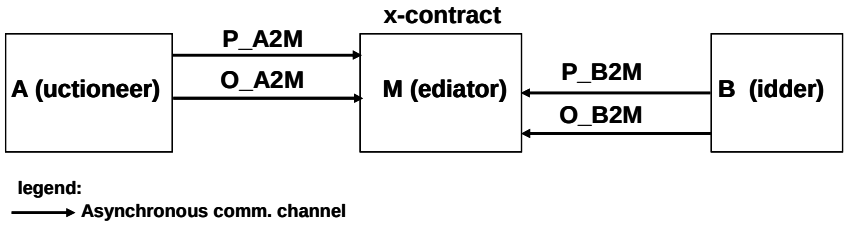


Fig. 5. Reactive central Promela model.

To make our paper self contained, we will very briefly discuss the main features of Promela [7]. Statements in Promela are either executable or blocked. A process trying to execute a blocked statement waits until an event that makes the statement executable occurs. Control flow in Promela is based on guarded commands which are represented by a double colon. Let us assume that $Stat_1$,…,$Stat_6$ are Promela statements and analyze the following construction where the symbol "->" is a statement separator that can be interpreted as a casual relation between its left and right statements.

```
if
::(Pay == 100)-> Stat1 -> Stat3          /* option1 */
::(Pay != 100)-> Stat2 -> Stat4-> Stat6 /* option2 */
fi
```

In the above construction either the sequence of statements of $option_1$ or of $option_2$ executes. Guards are not necessarily mutually exclusive, if more than one guard is executable, one of them is selected randomly. Messages are transferred from one process to another over asynchronous channels. The statement *chan!msg* can be executed by a sending process to send the message *msg* over the channel *chan;* whereas the statement *chan?msg* can be executed by a receiving process to receive a message from the channel *chan,* and store it in the variable *msg*. Promela supports type messages; the type of the message is a constant sent together with the message. A receiver will block until the expected typed message arrives through the channel. For example, the receiver executing the structure will block until either a message of type T1 or T2 is available from the channel.

```
if
::chan?T1(val)->…/*block until msg of type T1 arrives */
::chan?T2(val)->…/*block until msg of type T2 arrives */
fi
```

With this background in mind we can now present our contract represented as Promela code. The notion of obligation and permission as well as the notion of the role player obliged to perform an operation and the beneficiary of the operation is captured in the name of the communication channel; thus a channel named O_A2M suggests that an obligation (O) is expected to be fulfilled by the auctioneer (A) for the benefit of the mediator (M); strictly speaking, the beneficiary of the operation is the bidder, the name of the mediator appears in the name of the channel because the mediator is in control of the interaction; the receipt of the message at the mediator is taken as fulfilment of the obligation.

A complete Promela model for the system shown in Fig.5 would include the Promela description of three processes (the auctioneer, mediator and bidder) communicating over the

two channels; by complete here we mean a model that can be validated with Spin. To save space, we will show only a simplified version of the mediator; we do not show the mediator forwarding messages to their final destination; this would involve two additional channels in Fig. 5, namely, a mediator to bidder (M2B) and a mediator to auctioneer (M2B); in the same order, the Promela code would include mediator to bidder (M2B!IB(value)) and mediator to auctioneer (M2A!ACCEPT(value)) send statements, the former is shown commented and in bold font on the OFFERTOBID construction. We do not show either different types of ACK messages. We believe that this simplified code is explicit enough to help us understand (bare-eyed) the behaviour of the contract.

```
Proctype Mediator  (…)
/* prefix/suffix A and B stand for Auctioneer and Bidder  */
/* IB=Invitation to Bid,INVOICENOTIF=invoice notification*/
/* INVMSG= invalid message, val=value                    */
mtype={IB,ACCEPT,REJECT,INVOICENOTIF,ACK,INVMSG}

A_OFFERTOBID: /*A permitted to send IB to B*/
  if
  ::P_A2M ? INVMSG(val)-> goto B_SANCTION_A /*B fines A*/
  ::P_A2M ? IB(val)->/* M2B ! IB(val)*/ goto B_CONF_IB
  fi

B_CONF_IB:/*B obliged to confirm IB within*/
  if      /* 24 hrs or pay fine           */
  ::O_B2M ? INVMSG(val)-> goto A_SANCTION_B /*A fines B*/
  ::timeout             -> goto A_SANCTION_B /*A fines B*/
  ::O_B2M ? REJECT(val)-> goto ENDCONTR_OK
  ::O_B2M ? ACCEPT(val)-> goto A_SENDINVOICE
  fi

A_SENDINVOICE: /*A obliged to send invoice within 3days */
  if
  ::O_A2M ? INVOICENOTIF(val)-> goto ENDCONTR_OK
  ::O_A2M ? INVMSG(val)-> goto B_SANCTION_A /*B fines A*/
  ::timeout             -> goto B_SANCTION_A /*B fines A*/
  fi

A_SANCTION_B:/*A permitted to fine B*/
  if
  ::P_A2M ? INVOICENOTIF(val)-> goto ENDCONTR_OK
  ::P_A2M ? INVMSG(val)      -> goto ENDCONTR_DISP
  fi


B_SANCTION_A: /*B permitted to fine A*/
 if
 ::P_B2M ? INVOICENOTIF(val)-> goto ENDCONTR_OK
 ::P_B2M ? INVMSG(val)      -> goto ENDCONTR_DISP
 if

ENDCONTR_DISP: /*end of contract: dispute*/
/* do something here */

ENDCONTR_OK: /*end of contract: success*/
/* do something here */
```

We refer the reader to [9], where we describe how correctness properties can be model checked. Once implemented as an executable code, the contract above will guarantee that only legal messages (right type, right sequence, and time) reach their final destination. This is a great advantage for the auctioneer's and bidder's applications since they can blindly take incoming messages as correct and act upon them under the guarantee that they have already been approved by the mediator; furthermore, the applications are guaranteed that illegal messages sent accidentally will never reach their counterpart. A proactive version of the

contract would also offer the auctioneer's and bidder's applications the guarantee that they will be precisely instructed what actions to perform next.

3.1.4. Implementation specific representation

The contract shown in the previous section is not complete enough for technical people commissioned to convert into an x-contract: they will need to agree on the implementation related messages to be exchanged in order to enable business interactions to occur. The exchange of these messages will need to be expressed as additional permissions, obligations and prohibitions that the business partners are expected to honour.

An implementation-oriented contract will inevitably include sending both business action messages and business signal messages (for example, acknowledgements) to help state synchronisation. What signal messages need to sent depends on the technology used for implementing the contract. For example, the Rosettanet standards body [10] has specified a large number of business conversations (called Partner Interface Processes, PIPs). A Rosettanet based conversations (using PIPs PIP 3A4 and 3C3) for sending an invitation to participate in an auction and for sending an invoice for participating in the auction, is shown in Fig. 6.

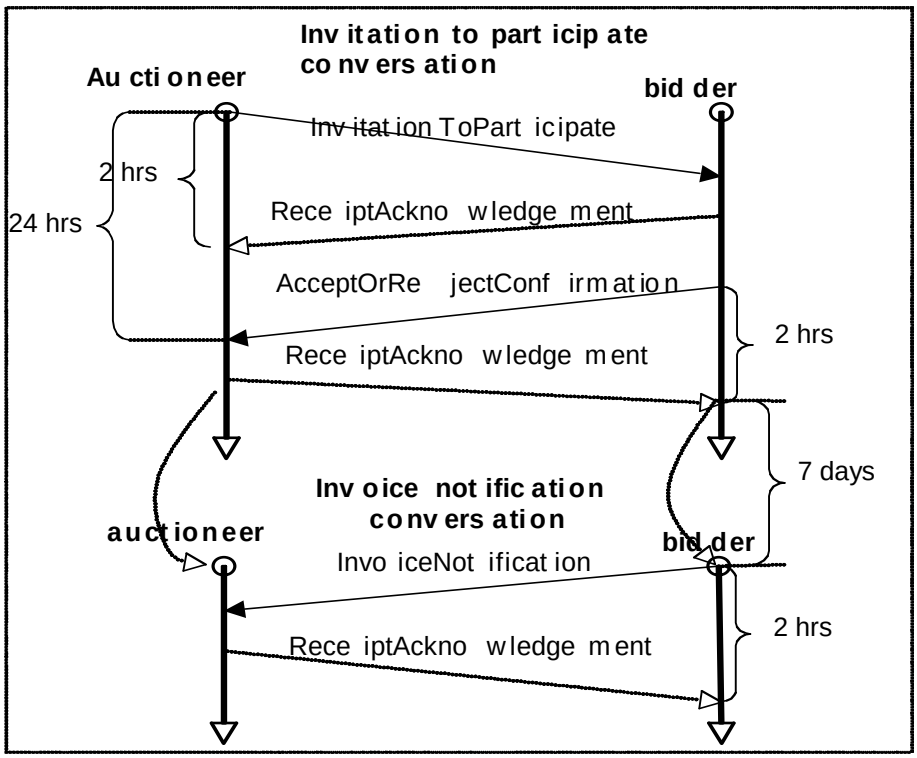


Fig.6. Implementation oriented auctioneer-bidder conversation.

In our view, a large class of typical business contracts can be regarded as composed out of a set of standard Rosettanet PIPs interactions.

The modified English text version of the implementation oriented contract is shown next. This version is different from the original one in that, it includes (in addition to the business

actions messages) business signal messages to help the two business partners synchronize their interactions. The new clauses appear in bold font.

1 Offer to bid

1.1 The auctioneer may use his discretion to send an invitation to participate in an auction, to the bidder.

1.1 ***The bidder is obliged to acknowledge the invitation within 2 hrs of receiving the invitation.***

1.2 The bidder is obliged to confirm if the invitation is accepted or rejected, within 24 hrs of receiving the invitation.

1.3 ***The auctioneer is obliged to acknowledge the invitation confirmation action within 2 hrs of receiving the message.***

2 Payment for participating in the auction

2.1 The auctioneer is obliged to send an invoice to the bidder within 3 days of accepting the invitation.

2.2 ***The bidder is obliged to acknowledge the invoice notification within 2 hrs of receiving it.***

3 Invalid messages

3.1 The auctioneer and the bidder are forbidden to send invalid messages.

4 Sanctions

4.1 Failures to honour obligations and prohibitions will result on fines equal to 20 euros. The offended party shall be granted permission to issue an invoice notification to the offending.

4.2 ***The offender is obliged to acknowledge the invoice notification within 2 hrs of receiving it.***

4.3 Failure to respond to a fine shall be sorted out outside this contract.

5 Synchronization and handling of transaction failures

5.1 Should the auctioneer or the bidder detect a technical failure that prevents it from continuing the normal course of a transaction, it is obliged to send a failure notification message by means outside this contract.

5.2 The counterpart is obliged to acknowledge the failure notification message within 2 hrs of receiving it.

As we did with the implementation-neutral example, it is possible to abstract the clauses of this implementation specific contract as a list of permissions, obligations and prohibitions; it

is also possible to express these permissions, obligations and prohibitions as Promela code. We will not show this here (for details, see the first paper in the Appendix).

3.1.5. Role based access control

Finally, we need to ensure that only legitimate role players of a participating enterprise are involved in a conversation. The goal of trust enforcement in TAPAS is to guarantee that messages received by a business partner are sent by legitimate senders, that is, by agreed upon role players (managers, accountants, engineers, secretaries). The assumption here is that different role players will have the permissions, obligations and prohibitions to send messages of different types (request purchase orders, invoice notifications, payments, etc.) In the deliverable report D5, we have described in detail how role based access control (RBAC) mechanisms can be used. So, we will only provide a summary here.

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

- Each enterprise 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 the contractual partnership, 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 contracting enterprise.

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:

- 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.

The main idea behind our model can be summarised as follows: When one role player (for example B) from one enterprise interacts with another role player from another enterprise, within the contractual partnership, the former presents (sends) its PKC certificate to the x-contract, together with a statement of its specific right's instance; in practice this is realized as a message type (purchase order, invoice notification, cancellation, etc.). The authentication process of the x-contract checks that the requesting entity is a legitimate party and that its requesting message is legal. This process also checks that the requesting public/secret key identities are complementary.

3.2. Component middleware for trusted coordination

Next we turn our attention to the bottom layer of the Inter-Organisation Interaction regulation subsystem (fig. 3). This layer provides two building blocks for regulated interaction between organisations: non-repudiable service invocation (NR-Invocation) and non-repudiable information sharing (NR-Sharing). Design of these building blocks has been described in the TAPAS deliverable report D9 [4], where an implementation using the JBoss application server is also presented. Here we summarise that design. In section 3.4, we also describe how the ideas can be extended to the world of Web services.

3.2.1. Trusted interceptor abstraction

In this section we introduce the abstraction of *trusted interceptors* that mediate inter-organisational interaction and then model non-repudiable service invocation and non-repudiable information sharing in terms of this abstraction. The trusted interceptor abstraction is sufficiently general to apply to a variety of interaction scenarios. For example, it is not bound to any particular non-repudiation protocols but can be seen as a flexible framework in which protocols can be deployed as appropriate to the regulatory regime governing an interaction or to the trust relationships between the parties to an interaction.

As shown in Figure 7, each organisation conceptually has a trusted interceptor that acts on its behalf. The introduction of the trusted interceptors transforms an unregulated domain into a trust domain that safeguards the interests of each party. The interaction between interceptors is regulated, audited and fair. That is, trusted interceptors provide a trust domain by policing access to the domain and regulating and auditing actions within the domain. The fairness guarantee is that honest parties will not be disadvantaged by the behaviour of dishonest parties. In the worst case, a break down in an interaction will lead to dispute. To support dispute resolution, the fact that trusted interceptors mediated the interaction will provide any honest party with irrefutable evidence of their own actions within the domain and of the observed actions of other parties. The trusted interceptor abstraction insulates the parties to the interaction from the detail of underlying mechanisms used to meet regulatory requirements. Interceptors can implement different mechanisms to meet different interaction

requirements and can be reconfigured to meet changing requirements as inter-organisational relationships evolve.

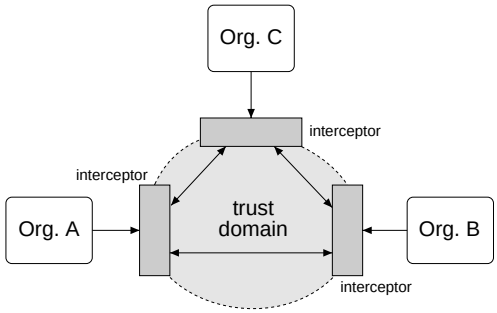
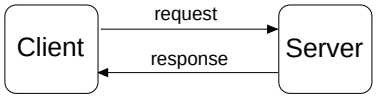
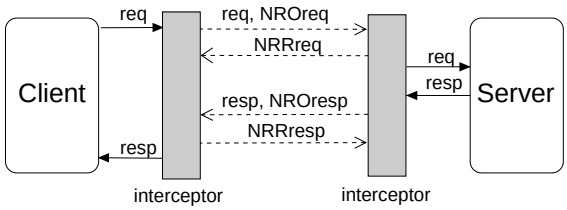


Figure 7. Trusted interceptors

3.2.2. Non-repudiable service invocation (NR-Invocation)



(a) Service invocation



(b) Non-repudiable service invocation

Figure 8. NR-Invocation through trusted interceptors

Figure 8(a) shows a typical two-party, client-server interaction. The client invokes a service by sending a request to the server who issues a response. Non-repudiable service invocation provides the following assurances to the client:

1. that following an attempt to submit a request to a server, either: (a) the submission failed and the server did not receive the request; or (b) the submission succeeded and there is proof that the request is available to the server; and:

2. that if a response is received, there is proof that the server produced the response.

For the server, the corresponding assurances are:

1. that if a request is received, there is proof identifying the client who submitted the request; and:

2. that following an attempt to deliver a response to the client, either: (a) the delivery failed and the client did not receive the response; or (b) delivery succeeded and there is proof that the response is available to the client.

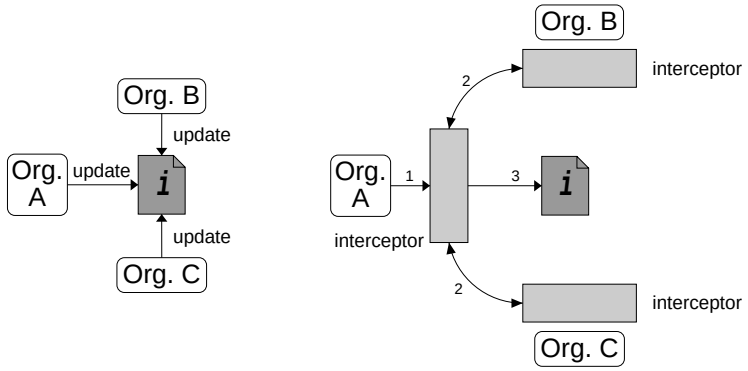
To provide the above assurances, trusted interceptors execute a non-repudiation protocol that ensures the following:

1. a request is passed to a server if, and only if, the client (or its interceptor) provides non-repudiation evidence of the origin of the request (NROreq) and the server (or its interceptor) provides non-repudiation evidence of receipt of the request (NRRreq)

2. the response is passed to the client if, and only if, the server (or its interceptor) provides non-repudiation evidence of the origin of the result (NROresp) and the client (or its interceptor) provides non-repudiation evidence of receipt of the response (NRRresp).

Non-repudiation tokens include a unique request identifier, to distinguish between protocol runs and to bind protocol steps to a run, and a signature on a secure hash of the evidence generated. Figure 8(b) models the exchange of evidence achieved by the execution of an appropriate non-repudiation protocol between interceptors acting on behalf of client and server. The client initiates a request for some service. The client's interceptor generates an NROreq token and then sends both the request and the token to the server's interceptor. The server's interceptor generates an NRRreq token and returns it to the client's interceptor. The server's interceptor then passes the request to the server to generate a response. On receipt of the response, the server's interceptor generates an NROresp token and sends both the response and the token to the client's interceptor. The interceptors ensure that irrefutable evidence of the exchange is both generated and stored.

3.2.3 Non-repudiable information sharing (NR-Sharing)



(a) Information sharing (b) Non-repudiable information sharing

Figure 9. NR-Sharing through trusted interceptors

Figure 9(a) shows three organisations (A, B and C) accessing and updating shared information. If, for example, A wishes to update the information, then they must reach agreement with B and C on the validity of the proposed update. For the agreement to be non-repudiable: (i) B and C require evidence that the update originated at A; and (ii) A, B and C require evidence that, after reaching a decision on the update, all parties have a consistent view of the agreed state of the shared information. The latter condition implies that there must be evidence that all parties received the update and all parties know whether there was unanimous agreement to it being applied to the information. Figure 9(b) shows A proposing

an update to the information shared by A, B and C. Interceptors are used to mediate each organisation's access to the information. In step 1, A attempts an update to the information. A's interceptor intercepts the update and, in step 2, executes a non-repudiable state coordination protocol with B and C to achieve the following:

1. That A's update is irrefutably attributable to A and proposed to B and C.

2. That B and C independently validate A's proposed update, using a locally determined and application-specific process, and their respective decisions are made available to A and are irrefutably attributable to B and C.

3. That the collective decision on the validity of the update (in this case, responses from B and C to A) are made available to all parties (A, B and C).

If the resolution of the protocol executed at step 2 represents agreement to the update then the shared information is updated in step 3. Otherwise, the information remains in the state prior to A's proposed update. Non-repudiable connect and disconnect protocols govern changes to the membership of the group of organisations sharing the information

The use of interceptor's allows us to abstract away the details of state coordination and insulate the application from protocol specifics. From the application viewpoint, the update to shared information is an atomic action that succeeds or fails dependent on the agreement of the parties sharing the information. Thus the interceptors may execute any protocol that achieves non-repudiable agreement on: the origin and state of a proposed update; the state of the shared information after application of an update; and the membership of the group that agreed to, or vetoed, the update.

3.3. Trusted Coordination API

In this Section we describe the specific components of the API for non-repudiable invocation and non-repudiable information sharing respectively and the tasks required of the application programmer

The non-repudiation middleware is organised in three layers:

1. the non-repudiable invocation and information sharing interceptor layer that intercepts application-level operations and initiates protocol execution

2. the protocol execution layer where protocol-specific handlers are instantiated for participation in non-repudiation protocols

3. the generic, protocol-independent coordination layer for exchange of protocol messages.

In addition an event and validation listener API has been defined for notification of protocol events and application-level validation of messages with respect to contract (for example, to validate updates to shared state).

The protocol execution and coordination APIs are shown below.

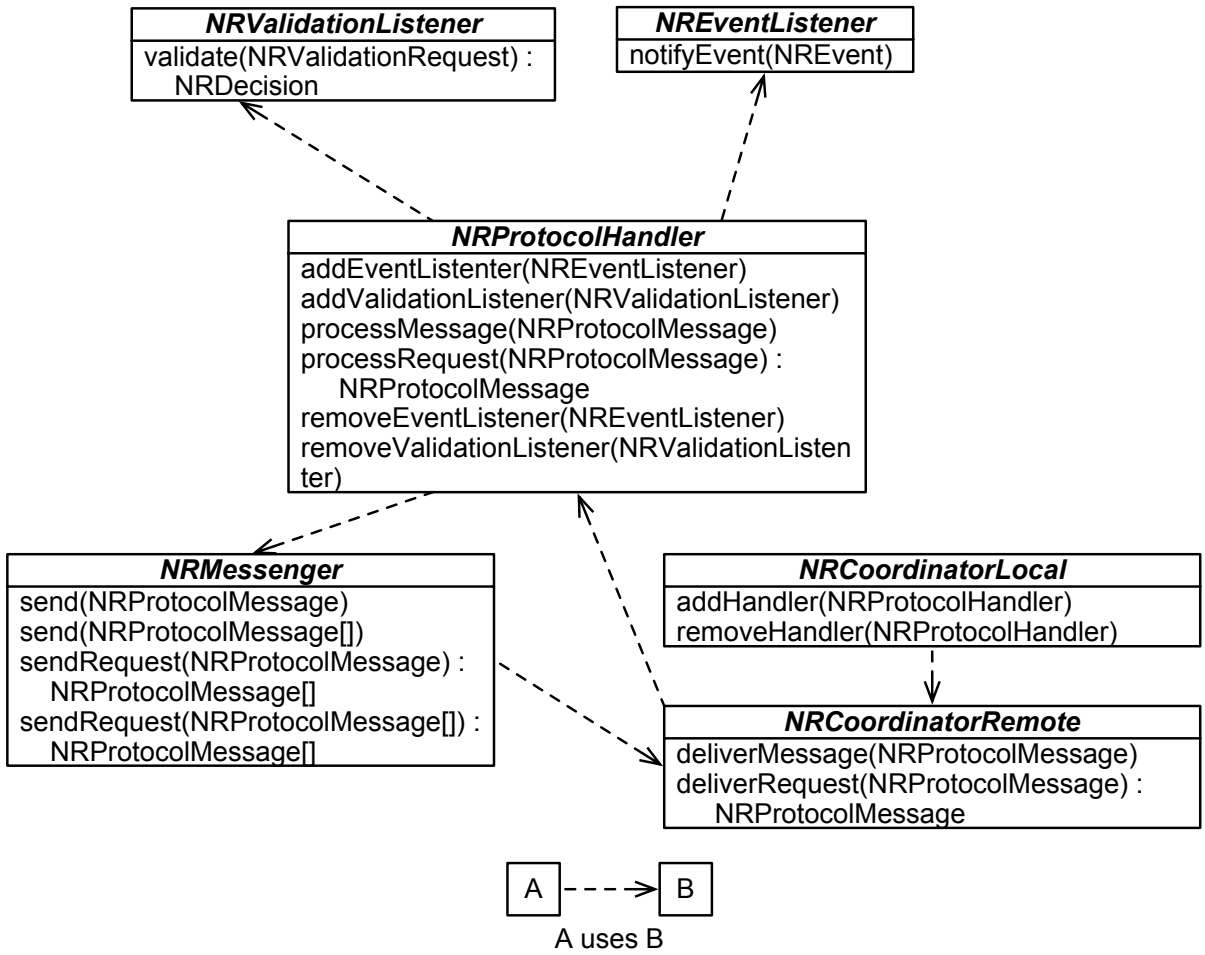


Figure 10. Protocol execution and coordination API

Each party provides a NRCoordinator service for the exchange of protocol messages. This service is the external entry point for execution of non-repudiation protocols. Remote invocation of `deliverMessage` results in delivery of the given message from the remote party (as a parameter to the call). `deliverMessage` can be used for synchronous or asynchronous protocol execution. `deliverRequest` is a convenience method that allows a remote party to deliver a message and then to wait synchronously for a response (the result of the call). NRProtocolMessage defines the interface to content that is common to protocol messages — request (protocol run) identifier, sender, protocol step, signed content, payload etc. Concrete implementations of NRProtocolMessage meet protocol-specific requirements. The coordinator is responsible for mapping an incoming protocol message to an appropriate handler. The coordinator also provides access to local services that are not protocol or platform specific. All protocol handlers provide an interface to the local coordinator service to process incoming messages. Protocol handlers use event and validation listeners to notify events and to trigger application-level message validation. Protocol handlers use a generic NRMessenger interface to send messages to remote parties. A concrete implementation of an NRMessenger is responsible for binding to remote coordinator services and for delivering messages to the remote services using the NRCoordinatorRemtoe interface.

NR-Invocation API

To render JBoss/J2EE service invocations non-repudiable the server-side application programmer identifies when non-repudiation is required and provides non-repudiation configuration information. On the client-side, configuration information is provided to determine the client's reaction to a server's demand for non-repudiation. NR-Invocation is declarative, requiring no additional implementation on the part of the application programmer.

The relationship of the NR-Invocation interceptor API to protocol execution and coordination is shown below.

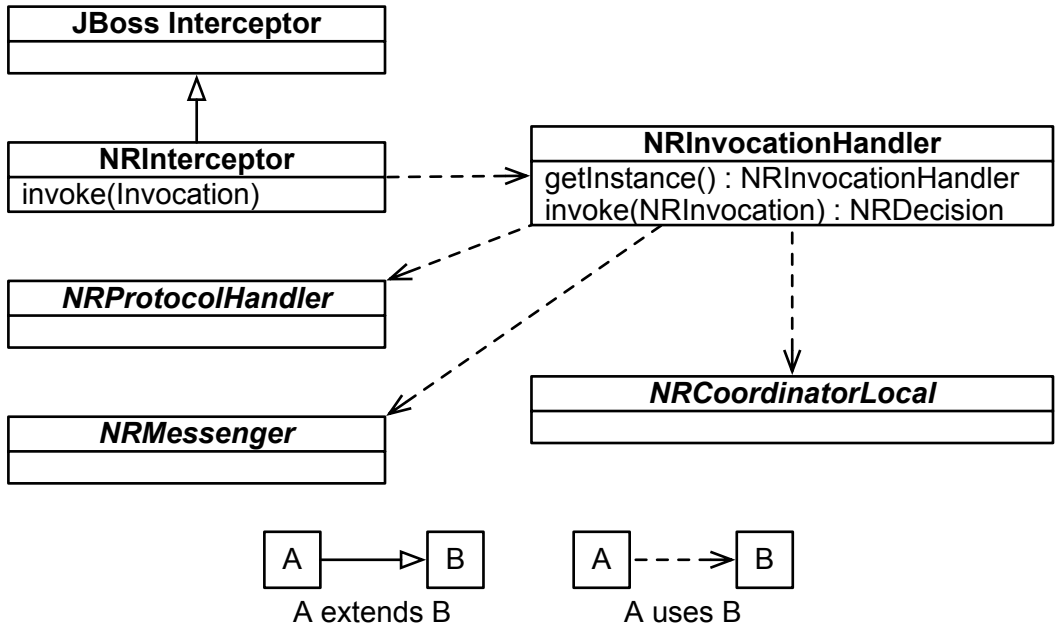


Figure 11. NR-Invocation interceptor API

Every JBoss interceptor has an invoke operation. The invoke operation takes an Invocation object as a parameter that the interceptor processes in some way. The interceptor then passes the Invocation to the next interceptor in the chain by calling that interceptor's invoke operation.

Client and server NRInceptors are responsible for instantiating appropriate NRInvocationHandlers to manage participation in protocols. The NRInvocationHandler acts as a bridge between the JBOSS-specific interceptor and the generic protocol execution framework shown above. The invocation handlers manage any conversion of information between layers (including protocol outcomes), initiate protocol execution and instantiate the protocol-specific handlers.

The server-side application programmer determines whether non-repudiation is activated for a given invocation. They can request that the client participate in the generation of non-repudiation evidence at three levels: (i) for all components with a remote interface in a container; (ii) for all remote methods of a given component; or (iii) for specific methods of a component.

Whether the client complies with the request is determined by the configuration information provided on the client-side to their non-repudiation middleware. Non-compliance will result in failure of a service request.

NR-Sharing API

For non-repudiable information sharing, B2BObjects middleware manages the state coordination process, including initiation of local validation of state updates to ensure that they are unanimously agreed and non-repudiable. In the JBoss/J2EE version of the middleware, an interceptor mediates access to the underlying information state and invokes operations on components of the middleware to effect the necessary coordination. The interceptor interacts with the two components of the middleware shown below:

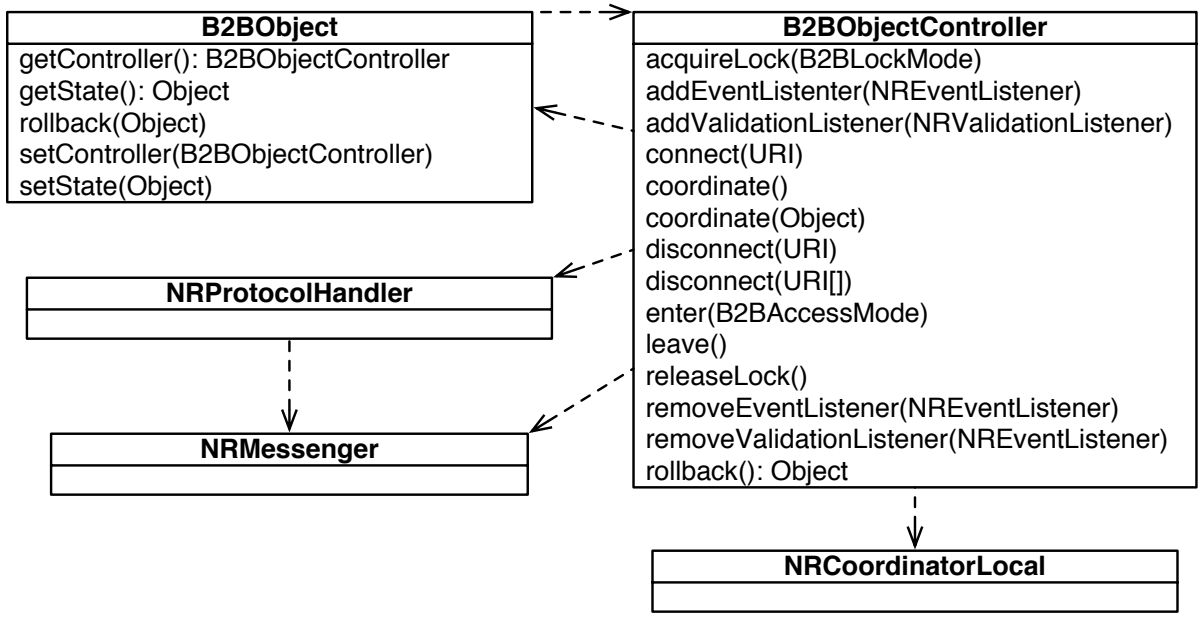


Figure 12. B2BObject and B2BobjectController API

The B2BObject interface defines methods that are invoked by the middleware to manage a bean's state during coordination with remote parties. The implementation of these methods is automatically generated by the middleware. As described below, the application programmer simply ensures that the component that represents the information state extends this interface. The interceptor uses the B2BObjectController interface of the middleware to wrap application-level access to the information state in enter/leave to effect state coordination, concurrency control etc. The wrapper code is generated automatically and is transparent to the application programmer.

Local entity bean representation of shared information

An entity bean is used to represent the information that is shared by organisations. The application programmer defines this local representation of shared state. Get and set methods provide access to the state by co-located business logic session beans. To ensure that local invocations on these accessor methods are mediated by the B2Bobjects middleware, the application programmer must:

1. Declare that the entity bean's local interface extends B2BEJBLocalObject, which in turn extends the standard EJBLocalObject interface and the B2BObject interface. Extension of B2BEJBLocalObject provides the middleware with an interface to the entity bean to control access and update to the bean's state. The JBoss interceptor reflects on the local object interface to determine which controller operations should be invoked as a result invocations on the bean. Further, any entity bean that implements the B2BEJBLocalObject interface is rendered thread-safe since a controller lock is acquired for all access to the bean.

2. Declare that the entity bean implementation extends B2BEntityBean. The B2BEntityBean is a middleware provided implementation of the standard EntityBean interface and of the B2BObject interface. Middleware invocations on the B2BEJBLocalObject interface result in execution of code defined in the B2BEntityBean implementation. The middleware automatically generates the B2BEntityBean implementation by reflecting on the application programmer defined entity bean.

3. Specify, in the bean's deployment descriptor, an object identifier that is used by the middleware to key configuration information and to construct a URI for remote parties to identify the object as a partner replica for state coordination. The implementation classes of any validation listeners and event listeners are also identified in the deployment descriptor.

For example, suppose parties wish to share the state of an order represented by an *OrderBean*, the following figure shows the interfaces that the entity bean components (the local interface and implementation) must extend.

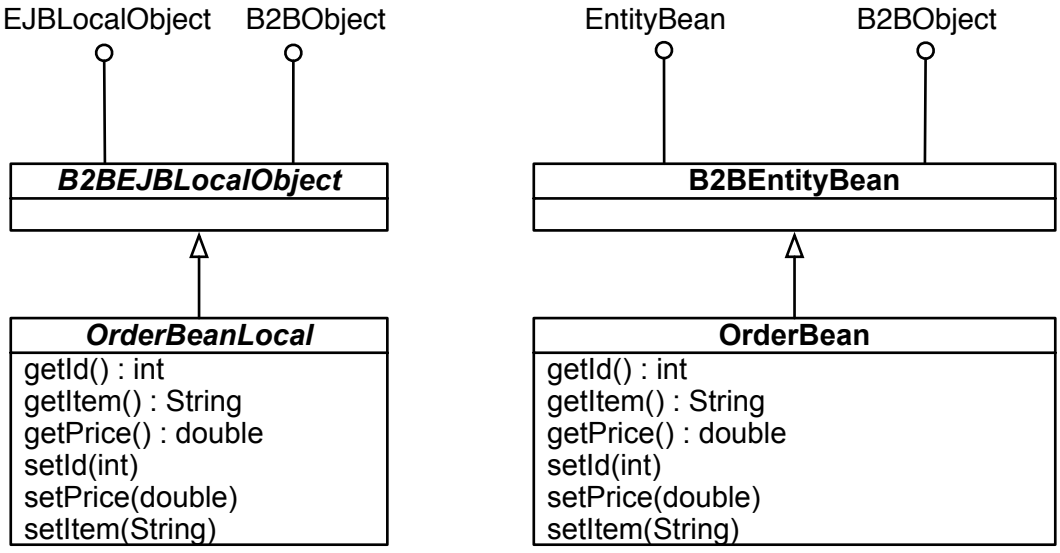


Figure 13. Order entity bean

As shown, OrderBeanLocal defines the business methods (get/set accessors) that can be invoked by co-located beans. This interface extends B2BEJBLocalObject that in turn extends the standard EJBLocalObject interface and the middleware-provided B2BObject interface. OrderBean is the abstract implementation of the bean and extends B2BEntityBean that

implements the standard EntityBean interface and the middleware-provided B2BObject interface. The declaration of the relationships shown is sufficient for an entity bean to become a B2BObject that is able participates in state coordination with remote parties. The application programmer is not responsible for the concrete implementation of B2BEntityBean.

3.4 Non-Repudiable Interactions with Web Services

The ideas presented in the previous two sections have been incorporated in the J2EE middleware (JBoss application server), see deliverable D9 [4]. We describe here how the interceptor approach can also be used for implementing non-repudiable interactions with Web Services. Despite the fact that Web services are increasingly used for enabling B2B interactions, there is currently no systematic support for non-repudiation. Here we assume the typical pattern of XMLbased business messages that should be exchanged between partners to execute some function (such as order processing). Business messages typically follow the Rosettanet PIP style: a message is acknowledged 'immediately', followed by a second valid/invalid message that is sent by the receiver after some application specific validation (see fig. 14).

Our design and implementation is based on a third party delivery agent (DA, a trusted third party) that takes on most of the responsibilities of evidence verification and storage and, thereby, simplifies the tasks for end users. Figure 15 shows what the delivery agent does. Four types of evidence are generated; (i) the proof of submission (NRS) that the DA received msg and is able to continue the protocol; (ii) Proof of origin (NRO) that msg originated at A; (iii) Proof of receipt (NRR) that B has received msg; (iv) Proof of validation result (NRV) regarding the outcome of B's validation of msg. As shown, A starts an exchange by sending a message, with proof of origin, to DA. This is the equivalent of message 1 in Figure 14 with the NRO appended. DA exchanges msg and NRO for NRR with B (before application-level validation of msg). The DA provides NRR to A — equivalent to message 2 in Figure 14. Subsequently, B performs application-level validation of msg (as in message 3 of Figure 14) and provides NRV to DA. The DA, in turn, provides NRV to A. Note the exact sequence of message exchange will be dictated by the actual protocol used and should not be inferred from Figure 15.

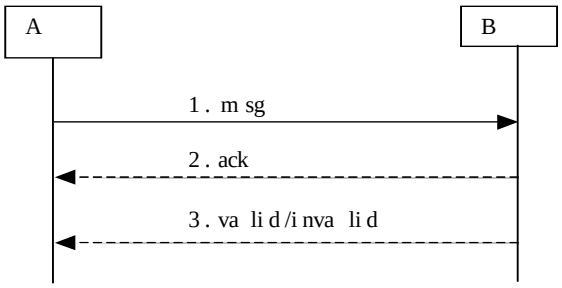


Fig. 14. Business message delivery with acknowledgements

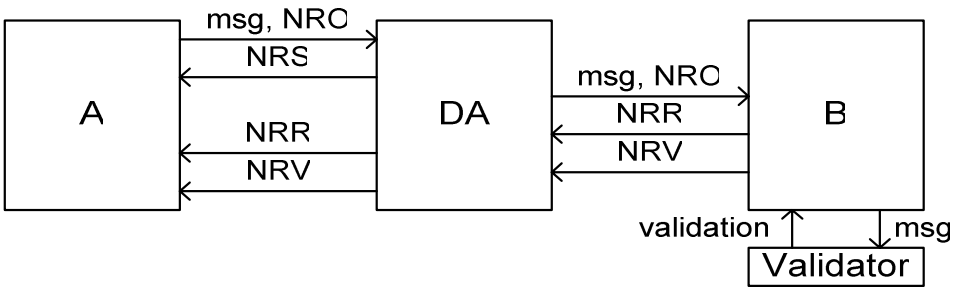


Fig. 15. Executing a business interaction through a delivery agent

As shown in Figure 16, our approach is to deploy interceptors that act on behalf of the end users in an interaction. An interceptor has two main functions: (i) to protect the interests of the party on whose behalf it acts by executing appropriate protocols and accessing appropriate services, including trusted third party services; and (ii) to abstract away the detail of the mechanisms used to render an interaction safe and reliable for its end user. In this case, the mechanism used is to communicate through a trusted third party — DA. It is the responsibility of the DA to ensure fairness and liveness for well-behaved parties in interactions that the DA supports. Further, the DA's fairness and liveness guarantees hold for well-behaved parties in spite of any misbehaviour by any other party involved in an interaction (including misbehaviour by interceptors). Since the cooperation of misbehaving parties cannot be guaranteed, in extremis the DA will ensure that any disputes that arise can be resolved in favour of well-behaved parties.

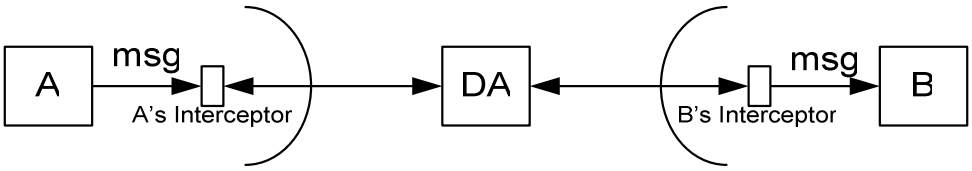


Fig. 16. Interceptor approach

Complete details of the Web service based implementation, including the non-repudiation protocols are presented in the third paper of the Appendix.

4. QoS monitoring and violation detection

As the name suggests, monitoring of contractual SLAs is about collecting statistical metrics about the performance of a service to evaluate whether the service provider complies with the level of QoS that the consumer expects. Such monitoring is frequently required to be carried out with the help of third parties to ensure that the results are trusted both by the provider and consumer. The state of art in the monitoring of SLAs by third parties is not yet well advanced: current contracts frequently leave SLAs open to multiple interpretations because they either contain ambiguous specifications of SLAs or no specification at all; likewise, they often do not unambiguously specify how the QoS attributes are to be monitored and evaluated.

The TAPAS QoS monitoring and violation detection subsystem (see fig. 1) overcomes these shortcomings. This subsystem could be provided by the ASP or one or more trusted third parties. Second year deliverable report, D10 [5] described the design and implementation this subsystem. In TAPAS, QoS requirements in SLAs are specified precisely using the SLAng language described in deliverable report D2 [6]. Month 30 deliverable report D15 [2], chapter 3, describes how QoS monitoring of the auction application was performed. This work has been written up for publication, and is attached as paper 4 in the Appendix. A review paper on service level agreements and their monitoring etc, written by TAPAS members and a colleague from industry (HP) is attached as paper 2 in the Appendix.

4.1. Architecture

The architecture that we propose for monitoring the level of QoS delivered by a provider to a given service $consumer_i$ at a given service point of presence ISP_i, is shown in Fig. 17. In the figure we assume that the interaction between the provider and the service consumer is regulated by a signed contract. The goal of monitoring is to watch what a business partner is doing, to ensure that it is honouring its obligations. We assume that monitoring is to be carried out with the help of third parties to ensure that the results are trusted both by the provider and consumer.

Notice that for the sake of simplicity only one point of presence and one service consumer is shown in the figure. However, in a general scenario, the provider would have one or more points of presence; each of them with an arbitrary number of service consumers.

To keep the figure and our discussion simple and without loosing generality we assume that the provision of the service is unilateral, that is, only the provider provides a service. Because of this, only the performance of the provider needs to be measured and evaluated. In practice, it is quite possible to find applications with bilateral service provision, where the contracting parties deliver something to each other and applications where the performance of the consumer affects the performance of the provider. We will show the generalisation of our architecture later. Though it is not shown in the figure, the assumption here is that the business between the provider and each of its service consumers (service $consumer_i$ for instance) is regulated by a signed contract. The contract clearly stipulates the SLAs at the service point of presence. Similarly the contract stipulates metrics that are to be measured and with which frequencies, to asses the performance of the provider. With these observations in mind, it makes sense to think that a provider will have several instances of the scheme shown in the figure, that is, one instance for each of its service consumers.

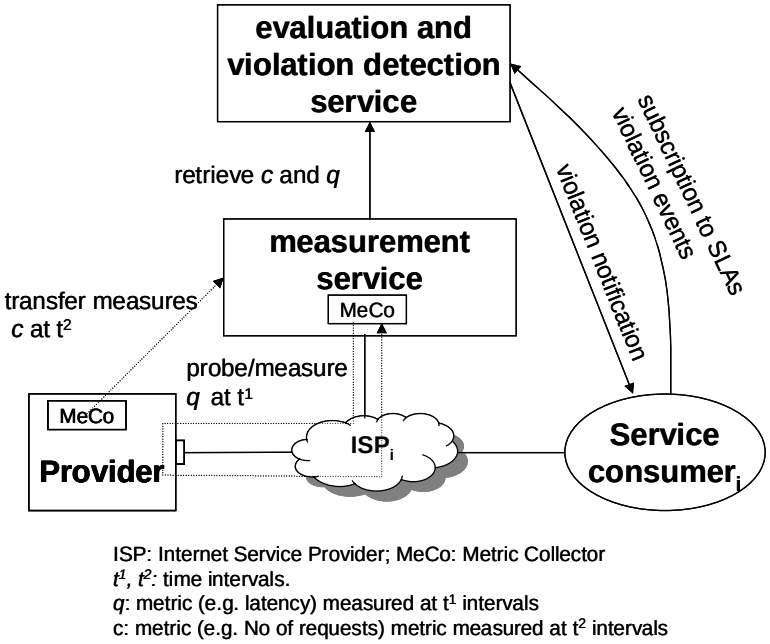


Fig. 17. Architecture for unilateral monitoring of QoS.

Two third party services are required:

- **Measurement service:** an enterprise trusted by the provider and the service consumer and with expertise in measuring a given list of metrics at specifies intervals and storing the collected results in its databases.
- **Evaluation and detection violation service**: an enterprise trusted by the provider and the service consumer. It is there to retrieve metrics from the databases of the measurement service, perform computation on them, compare the results of the computation against high or low watermarks and send notifications of violations to the service consumer when violations of SLAs are detected.

Notice that, for the sake of simplicity, in the figure we show single enterprises performing the functions of the measurement, and the evaluation and detection violation services. In practice, the measurement service can be performed by several enterprises that compensate their functionality with each other or replicate them to provide more reliability. Naturally, the evaluation and detection violation service can be realised in a similar way.

Notifications of violations are represented as events. We envisage an event notification system offering the service consumer the possibility to subscribe to events in which it is interested. It is not difficult to imagine that the service consumer can dynamically subscribe and unsubscribe to different events, perhaps in accordance with the momentary needs of the applications that it is running. To simplify the figure, notifications of violations are sent only to the service consumer; however, these notifications can be sent to other parties (for example, the provider) who express interest by means of subscriptions. The issue about where and how notifications of SLA violations are processed by the service consumer falls out of the interest of this work. However, we can briefly mention that such notifications can

be caught by a contract management system that will, after interpreting them, take the necessary actions, such as sending a complaint note or a penalty bill to the provider.

4.2. Metric collection

The contract would stipulate the level of QoS that the provider is obliged to deliver to the service consumer at the service point of presence ISP_i when certain conditions (for example, no more that 10 requests per second) in the usage of the service hold. This implies that although the service $consumer_i$ of Fig. 17 is not delivering any service to the provider, it still has obligations to honour; consequently it has to be monitored as well. It can be said that in general, monitoring is a symmetric activity. This is why measurement services rely on two kind of MeCo.

- Provider's performance MeCo: a MeCo for collecting metrics about the level of QoS delivered by the provider at the service point of presence.
- Consumer's behaviour MeCo: a MeCo for collecting metrics about the behaviour of the service consumer.

The critical issue here is to find a suitable approach for deploying the two MeCo. Different alternatives for deploying the MeCo were discussed in-depth in D10 [5], so in this discussion we will only mention that the architecture shown in Fig. 17 illustrates the situation where the service consumer does no wish to be disturbed unduly with metric collection responsibilities and where the service provider can be trusted to collect information about the consumer's behaviour.

In the figure, the metrics about the provider's performance are collected by the measurement service which is hired by the contracting parties as a trusted third party, to work as a probe equipped with a MeCo. The dotted arrowed line that goes from this MeCo to the provider and back to the MeCo, is there to show that to probe the service, the provider's performance MeCo issues a synthetic operation (at agreed upon intervals) and waits for a response.

In our architecture, we placed the MeCo to monitor the consumer's behaviour within the service provider's infrastructure. This MeCo is in the right location to collect metrics about the number of requests issued by the service consumer, about the resources (number of CPUs, database servers, disk memory, cryptographic keys and TCP ports) demanded by each request, etc. Likewise, it can tell whether the service consumer is maliciously or accidentally placing illegal operations on the provider. The dotted arrowed line pointing from this MeCo to the measurement service is meant to show that the metrics collected by this MeCo are transferred at some point and over the Internet to the measurement service who stores them.

The specific nature of the metrics to be collected depends on the application. On the application and on the SLAs depends also the interval at which the metrics are to be collected. This information is extracted from the contract and provided to the measurement service. For example, the measurement service might be requested to collect metric about the latency (to perform a given operation) of the service every five minutes or to collect metrics about the availability of the service every three minutes.

In the figure, we can imagine that the evaluation and detection violation service is retrieving the latest *n* value of the metric *c*, and the latest *k* values of the metric *q*. We can imagine that *q* is a metric that defines the latency of an operation and *c* is the metric that defines the number of employees from the service consumer's logged into the provider at a given moment of time, that is, the working conditions of the provider. If this is true then the evaluation and detection violation service can compute the latest average latency under the latest average number of users, with an accuracy that depends on the interval (t^1 and t^2 respectively) with which *q* and *c* are measured by the measurement service.

4.3 QoS monitoring and violation detection API

This section describes the configuration and application programming interfaces of the previously described service. We assume inter-organisational communication is enacted over the Internet using SOAP, and the measurement service uses Java messaging Service (JMS), see figure 18.

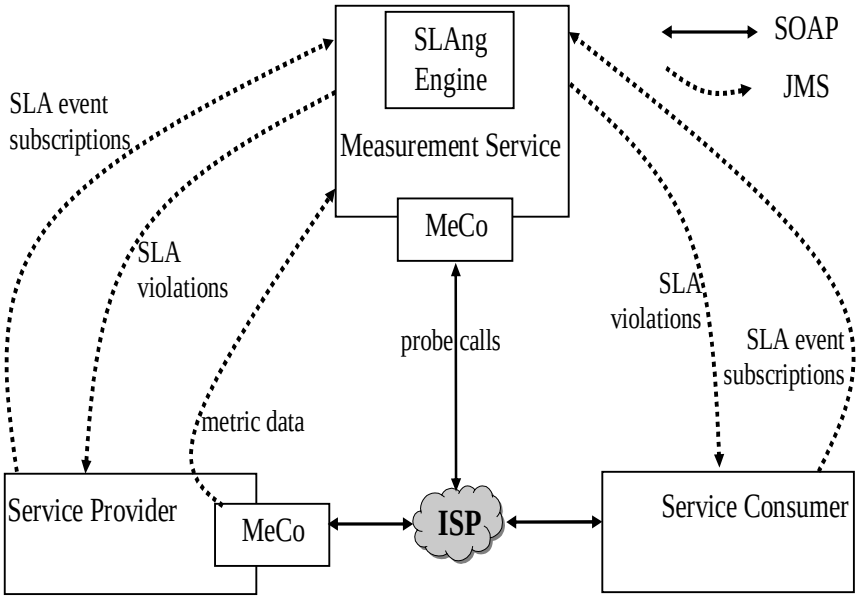


Figure 18. System components

Configuration

Executing SLA monitoring and evaluation requires some configuration of the software within the service provider and measurement service environments. Assuming a probe style deployment (as advocated in our approach), there is no requirement to alter the configuration of the service consumer environment.

Configuration (e.g., identifying metric data to gather), is achieved via updating configuration files (text) and minor code modifications at both the service provider and measurement service. We now continue with an overview of the configuration required to ready our SLA monitoring and evaluation service for runtime. For brevity, we take an abstract view of our configuration files without going into great detail of the tailoring of the configuration files associated with the middleware platform (which is extensive).

Service Provider

In the service provider we are primarily concerned with identifying what metric data to collect and the JMS topics that allow data to be sent to the measurement service. Configuration of the middleware platform associated with service provision must be accomplished to incorporate MeCo hooks into the service platform. The manner in which this is achieved depends on the vendor providing the platform. We assume JBOSS as the provider of the EJB platform and Apache's Axis as providing the Web Services interface.

Irrelevant of middleware platform, MeCo hooks determine what metric data to gather from loading classes (*metric data classes*) from the class repository (classes generated directly from SLAs). A wrapper class (*platform wrapper*) is required to allow integration of the metric data classes into a specific platform (a MeCo hook is the combination of platform wrapper classes and metric data classes). This approach allows individual classes in the class repository to be reused for many different middleware platforms.

Intercepting requests/replies to allow MeCo hooks to gather data is middleware dependent: for Axis style Web Service deployment the `<request-flow>` and `<response-flow>` elements in the `server-config.wsdd` file must be updated, for JBOSS EJB deployment the bean interceptor stack must be updated.

```
<?xml version="1.0" encoding="UTF-8"?>
<meco-server
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="/opt/jboss/server/TAPAS/conf/meco-server.xsd">
  <config>
    <slaEngine>SLAng</slaEngine>
    <messaging>JMS</messaging>
  </config>
  <SLAng>
    <contract>file:/opt/jboss/server/TAPAS/conf/auction_demo_contract.xmi</contract>
  </SLAng>
  <JMS>
    <namingFactory>org.jnp.interfaces.NamingContextFactory</namingFactory>
    <providerURL>127.0.0.1:1099</providerURL>
    <connectionFactory>ConnectionFactory</connectionFactory>
    <jndiPrefix>topic</jndiPrefix>
    <providerID>AuctionProvider</providerID>
  </JMS>
</meco-server>
```

Figure19. Example `meco-server.xml` file.

The `meco-server.xml` is the configuration file (example shown in figure 19) used by the MeCo environment and identifies the location of the SLA, JMS related items (e.g., JMS server location – note that `jbossmq-destinations.xml` is for JBOSS's use only) and other configurable flags (such as the path of the class repository).

Measurement service

The measurement service consists of two parts: (i) SLA evaluation and notification, (ii) probing of service provider. For the evaluation and notification aspects of the measurement service there exists a configuration file (`measurement-service.xml` – example extract shown in figure 20) that maintains information relating to the location of SLAs and JMS server.

```
<measurement-service
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:/eclipse/workspace/MeCo_03_03/misc/measurement-service.xsd">
  <config>
    <slaEngine>SLAng</slaEngine>
    <messaging>JMS</messaging>
  </config>
  <SLAng>


<contract>file:C:/eclipse/workspace/MeCo_03_03/misc/contracts/auction_demo_contract.xmi</contract>
    <maintainUsageHistory>true</maintainUsageHistory>
    <useEngine>false</useEngine>
  </SLAng>
  <JMS>
```

Figure 20. Extract from `measurement-service.xml` configuration file.

Configuration relating to the probing of the service provider is located in a Web Service Descriptor Language (WSDL) file named in `measurement-service.xml`. WSDL files are used to describe how to communicate with a Web Service, and as such can be used to configure the probe to send messages to the target server. The (Java) classes required to enact probing (probe classes) are created via the parsing of additional extensibility elements defined in the given WSDL file. These elements also provide a realistic set of parameters to supplement this approach to probing. The structure of these elements is defined in the `meco-probe.xsd` file. As with the platform wrapper class in the service provider MeCo, a platform wrapper class is used for implementing the probing for a specific middleware platform (EJB/RMI or Web Services/SOAP).

APIs

Service provider

The service provider has a pre-runtime element (*SLAng manager*) that parses the SLA files to create the Java classes responsible for gathering metric data. This element is executed as part of the service provider start-up routine, by specifying the absolute path of the SLA file to parse (indicated in the `meco-server.xml` configuration file).

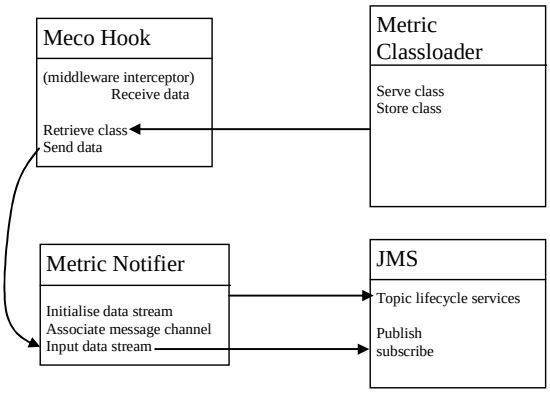


Figure 21. Abridged view of component APIs for server side.

Figure 21 provides an abstract description of the server components' APIs (excluding SLAng manager that is a pre-runtime component). In figure 21 the elements of the MeCo hooks that are middleware dependent are shown in brackets. This is where the metric data is actually

retrieved via interceptors on the observed platform and is a vendor dependent action. Once retrieved, the metric data is passed to the *metric notifier* that assumes responsibility for publishing the appropriately formatted data on the JMS server. The metric notifier also assumes responsibility for gaining references to the JMS topics.

Measurement service

The measurement service has a pre-runtime element (*Metric Manager*) that parses SLA files and generates the appropriate Java classes for readying incoming messages for consumption by the SLAng engine. The metric manager is executed at the command line, passing the absolute path of the configuration file to be parsed. Alternatively, if no command line parameters are provided all SLA files are parsed and appropriate Java classes produced (location of SLA files indicated by `measurement-service.xml` configuration file).

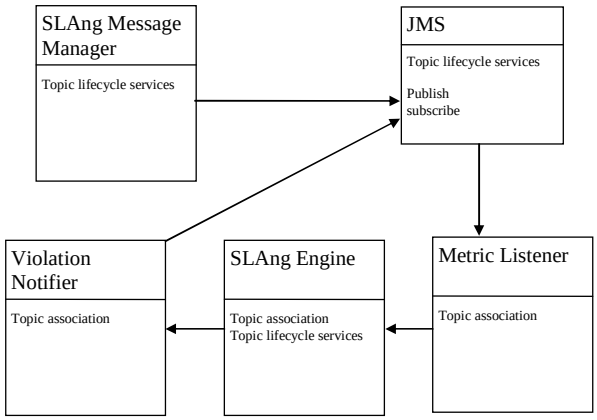


Figure 22. Abridged view of component APIs for measurement service.

Figure 22 provides an abstract description of the measurement service's APIs (excluding pre-runtime element metric manager). The *SLAng manager* manages the lifecycle services associated with JMS topics (e.g., creating topic when required by an SLA). The *metric listener* consumes metric data and passes such data to the SLAng engine. The metric listener also prepares the messages gained from the JMS into a format that may be consumed by the *SLAng engine*. The *violation notifier* receives notifications of SLA violations from the SLAng engine and passes these notifications to the appropriate SLA participants (via JMS).

5. Evolution of the TAPAS Architecture

We describe here how the architecture has evolved during the life time of the project. The architecture as originally envisaged was very much seen as enhancements to the J2EE component middleware. This is reflected in the titles of the deliverables concerned with design and implementation of various parts of the architecture, namely D7, D8, D9, D10 and D11 listed in TAPAS description of work (DOW) document:

D7: QoS Container Interface Specification

D8: Container for Group Communication

D9: Container for Trusted Coordination

D10: Container for QoS Monitoring

D11: Revised QoS Container Interface Specification

At the start of the project, we expected that research work on trust, security and QoS adaptability features will lead to enhancements to the functionalities of component containers. This turned out to be a rather narrow view of the architecture.

It was only after the first year's work that the overall 'big picture' emerged, as documented in the TAPAS deliverable report, D5 Supplement, June 2003, entitled 'An overview of the TAPAS Architecture'. We identified three key requirements for application service provisioning (stated in section 2 of this report).

1. Enhancing the application hosting middleware platform to be QoS aware. This way, hosting platform will be better equipped to meet the requirements of the hosting applications. In the absence of such a feature, the only alternative available to an ASP is *over provisioning*, which is not particularly desirable.

2. Ability to ensure that all inter-organisation interactions are strictly according to the terms and conditions contracts in force. In the worst case, violations of agreed interactions are detected and notified to all interested parties; for this, an audit trail of all interactions will need to be maintained.

3. Ability to demonstrate that hosted applications are meeting the various QoS requirements of SLAs.

These three requirements underpin the design of the TAPAS architecture, and led to the three subsystems shown in fig. 2. It became clear to us that rather than enhancing component containers, we need to design complete subsystems, that could work with each other, and also work independent of each other. This is reflected in the title and contents of deliverables D7 – D11 that were subsequently produced:

D7: TAPAS Architecture: QoS Enabled Application Servers

D8: QoS adaptive Group Communication

D9: Component middleware for Trusted Coordination

D10: QoS Monitoring of Service Level Agreements

D11: QoS-aware Application Server: Design, Implementation, and Experimental Evaluation

To reiterate, an important feature of TAPAS architecture is that the three subsystems can be deployed independent of each other. For example, an ASP might decide to use a 'standard' application server, without the need for QoS management features, because in a given scenario, over provisioning might be acceptable. The ASP still might need one or both of inter-organisation interaction regulation and QoS monitoring and violation detection subsystems. Another important feature of the TAPAS architecture is that the inter-organisation interaction regulation subsystem, as well as the QoS monitoring and violation

detection subsystem could be provided by the ASP or one or more trusted third parties, thereby providing extreme flexibility in deployment.

Not surprisingly, the emergence of Web services during the life time of the project as the dominant mode for inter-organisation interactions has influenced the TAPAS architecture. Inter-Organisation Interaction regulation subsystem has two main layers (see figure 3). The contract monitoring and enforcement layer makes use of the services of the underlying layer that provides trusted coordination. Our current design and implementation for trusted coordination, summarised in sections 3.2 and 3.3 is based on J2EE middleware, with interactions taking place using RMI. Figure 23 (taken from deliverable D9), shows how trusted interceptors have been used in the JBoss application server for performing non-repudiated invocations.

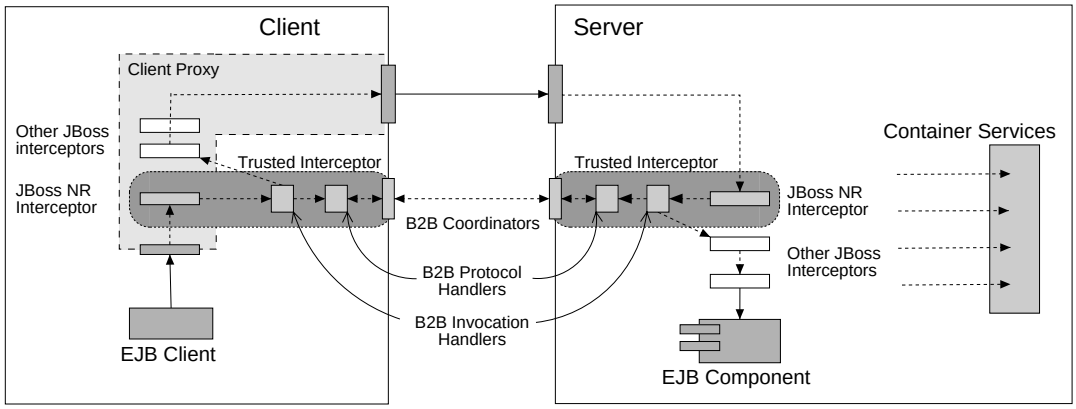


Figure 23. JBoss/J2EE-based implementation of non-repudiable invocation

In the world of Web services, it is increasingly common for interactions to take place using (reliable, persistent) messages. Thus we expect the Web services version of our trusted coordination layer to be message based, rather than RPC based. During the tail end of the project, we did spend some effort in designing and implementing a non-repudiation system for Web services, that makes use of SOAP message based Web services middleware (see section 3.4 and fig. 24). The preliminary design is described in detail in appendix 3 of this report.

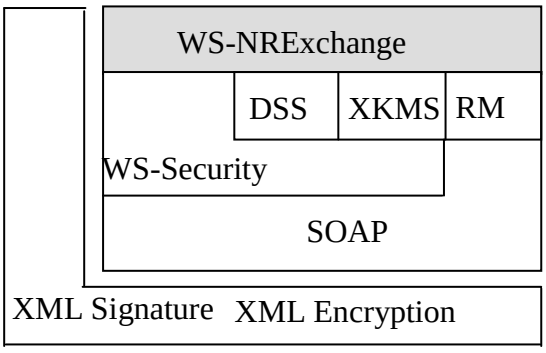


Fig. 24. Architecture of non-repudiation system for Web services

Of the three subsystems we have designed, we believe that the architecture of the two subsystems, namely, QoS monitoring and violation detection and QoS aware application server to remain fairly stable. The RPC based trusted coordination subsystem architecture, as

developed for J2EE middleware will require substantial re-design for the world of Web services; here we have performed initial work. It is likely that the architecture presented in fig. 24 might need considerable revisions, as the Web services middleware stack is still evolving, and has not reached the degree of maturity of CORBA or J2EE middleware.

6. Concluding Remarks

The TAPAS project is interested in developing solutions to the problem faced by 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. Three major subsystems were identified, see fig. 2. The QoS management, monitoring and adaptation subsystem is intended to make the underlying application server QoS enabled). This system is described separately in deliverable report D11. All cross-organisational interactions performed by applications are policed by the inter-organisation interaction regulation subsystem described in section 3. It has two components: one for mediating contractual interactions and the other for providing non-repudiable service interactions. We also need an application level QoS monitoring service, which must measure various application level QoS parameters, calculate QoS levels and report any violations. Its design was summarised in section 4.

References

[1] TAPAS Deliverable report D5, “TAPAS Architecture: Concepts and Protocols”, March 2003.

[2] TAPAS Deliverable report D15, “TAPAS QoS-aware Platform: technology and demonstration”, September 2004.

[3] TAPAS Deliverable report D11, “QoS-aware Application Server: Design, Implementation, and Experimental Evaluation”, March 2005.

[4] TAPAS Deliverable report D9, “Component middleware for Trusted Coordination”, March 2004.

[5] TAPAS Deliverable report D10, “QoS Monitoring of Service Level Agreements”, May 2004.

[6] TAPAS Deliverable report D2, “Specification Language for Service Level Agreements”, March 2003.

[7] Gerard J. Holzmann: Design and Validation of Computer Protocols. Prentice Hall, (1991).

[8] Gerard J. Holzmann: The SPIN model checker, Primer and reference manual. Addison-Wesley, (2004).

[9] Ellis Solaiman, Carlos Molina-Jimenez, Santosh Shrivastava: Model Checking Correctness Properties of Electronic Contracts. In Proc. of the Int. Conference on Service Oriented Computing (ICSOC03).Trento, Italy, Dec. 2003. Lecture Notes in Computer Science Vol. 2910, Springer (2003).

[10] Rosettanet implementation framework: core specification, V2, Jan 2000. http://rosettanet.org

[11] George Henrik von Wright: An Essay in Deontic Logic and The General Theory of Action. Acta Philosophica Fennica, North-Holland Publishing Company-Amsterdam, (1968).

[12] Peter Linington, Zoran Milosevic and Kerry Raymond: Policies in Communities: Extending the ODP Enterprise Viewpoint. In Proc. of The Second Int. Enterprise Distributed Object Computing Workshop (EDOC98). La Jolla, San Diego, Ca, Nov. 2-5, (1998).

[13] M. J. van den Hoven and G.J.C. Lokhorst: Deontic Logic and computer-supported computer ethics. Metaphilosophy, Vol. 33, N. 3, (2002).