Market Managed Multi-service Internet

M3I

European Fifth Framework Project IST-1999-11429

Architecture

The M3I Consortium

Hewlett-Packard Ltd, Bristol UK (Coordinator)
Athens University of Economics and Business, GR
BT Research, Ipswich GB
Eidgenössische Technische Hochschule, Zürich CH
Technical University of Darmstadt DE
Telenor, Oslo NO

© Copyright 2000 the Members of the M3I Consortium

*For more information on this document or the M3I Project,
please contact:*

Hewlett-Packard Ltd,
European Projects Office,
Filton Road,
Stoke Gifford,
BRISTOL BS34 8QZ,
UK
Phone: (+44) 117-312-8631
Fax: (+44) 117-312-9285
E-mail: sandy_johnstone@hp.com

©Copyright 2000, the Members of the M3I Consortium

Document Control

Title: Architecture
Type: Restricted Deliverable
Author: Bob Briscoe
e-mail: bob.briscoe@bt.com
M3I-deliverable 2
Work package: 3

Origin: BT Research
Doc ID: m3i:arch_pt1_1.doc

AMENDMENT HISTORY

Version	Date	Author	Description/Comments
V0.0	20-Apr-2000	Bob Briscoe	Initial draft version
V0.1	12 Jun 2000	Bob Briscoe	Re-worked to move introductory focus on service creation to later. Clarified interfaces, incorporating joint TUD/ETH views on architecture from PM design and engineering workshop. Added section 2.1 on principles. Added section 2.1 on use cases. Added sections 2.6 & 5 on interfaces. Added price reaction building block Changed terminology throughout:
V0.2	21 Jun 2000	Bob Briscoe	Interim draft for teleconference. Altered source of user charge advice in edge-centric and edge-control use case figures. Added GSP use cases and referred to DPH scenario; Added omitted text on charge advice interfaces.
v0.3	6 Jul 2000	Bob Briscoe	Added GSP use case and Compositions sections.
v0.4	7 Jul 2000	Bob Briscoe	Added Top level architecture section. General improvements following review.
v1.0	7 Jul 2000	Bob Briscoe	First Issue

Legal Notices

The information in this document is subject to change without notice.
The Members of the M3I Consortium make no warranty of any kind with regard to this document, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Members of the M3I Consortium shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

©Copyright 2000, the Members of the M3I Consortium

Table of Contents

1 INTRODUCTION 5

2 OVERVIEW 7

2.1 Guiding principles 7
2.1.1 Charging granularity 7
2.1.2 Commercial openness 7
2.1.3 Edge pricing and contracts 7
2.1.4 Scalability 7

2.2 Top level architecture 8

2.3 Typical use cases 8
2.3.1 Edge-centric use case 9
2.3.2 Edge-control use case 14
2.3.3 Inter-network use case 15
2.3.4 Guaranteed service provider use case 18
2.3.5 Further use cases 27

2.4 Applications overview 27

2.5 Building blocks overview 29
2.5.1 Service operation building blocks overview 29
2.5.2 Configuration building blocks overview 30

2.6 Interfaces overview 31
2.6.1 Interface distribution 32
2.6.2 Interface interaction mode 33
2.6.3 Message payload type(s) or type signature 34

2.7 Interim summary of parts 36

2.8 Compositions overview 36
2.8.1 Service components 37
2.8.2 By deployment granularity 38
2.8.3 By role granularity 39
2.8.4 By event granularity 40

3 DEFINITIONS 41

4 BUILDING BLOCKS 43

4.1 Network service operation building blocks 43
4.1.1 Networking 43
4.1.2 Classifying 44
4.1.3 Metering 45
4.1.4 QoS, rate, admission or access control 45
4.1.5 Charge advice 47
4.1.6 Price/charge reaction 48
4.1.7 Distribution 48
4.1.8 Aggregation 49
4.1.9 Settlement 50

4.2 Configuration building blocks 51
4.2.1 Directory 51
4.2.2 Service definition 52
4.2.3 Tariff 52
4.2.4 Offer 53
4.2.5 Offer location 56
4.2.6 Offer acceptance 56
4.2.7 Price setting 57
4.2.8 Address allocation 59

©Copyright 2000, the Members of the M3I Consortium

4.2.9 Classifier and meter configuration 59
4.2.10 Task-reaction policy association 60
4.2.11 Price reaction policy configuration 61
4.2.12 Distribution and aggregation configuration 61

4.3 Utility building blocks 62
4.3.1 Correlation 62
4.3.2 Transformation 62

4.4 Applications 62

4.5 Clarifications 64

5 INTERFACES 65

5.1 Network and lower layer interfaces 65

5.2 Service operation interfaces 66
5.2.1 Session characterisation interface family 66
5.2.2 Policy interface family (operation) 67

5.3 Configuration interfaces 68
5.3.1 Policy interface family (configuration) 68
5.3.2 Context interface family 70
5.3.3 Association interface family 70

6 COMPOSITIONS 71

6.1 Components 71
6.1.1 Meter system service component 72
6.1.2 QoS manager service component 72
6.1.3 Mini charging & accounting service component 73
6.1.4 Mediation service component 75
6.1.5 Charging and accounting system 76

6.2 Scenario compositions 78
6.2.1 Dynamic price handler with explicit congestion notification (DPH/ECN) 78
6.2.2 Other scenarios 80

6.3 Deployment engineering 80

7 LIMITATIONS & FURTHER WORK 80

7.1 Status 80

8 CONCLUSIONS 81

9 ACKNOWLEDGEMENTS 82

10 REFERENCES 82

©Copyright 2000, the Members of the M3I Consortium

1 Introduction

We wish to design a self-managing communication system that encourages those using it to use it however they want and it will just serve them, faithfully and predictably. This is the essence of a market managed, multi-service Internet.

This document presents our architecture for a market managed multi-service Internet. In fact, a way of describing architecture has been developed that encompasses all multi-service Internet architectures - simultaneously. That is, it encompasses the creation, configuration, operation (and destruction) of:

- **network service**
- its **load control**
- the **applications** it supports
- and **charging** for all this.

The technique is to identify a small but sufficient set of **service building blocks** from which we can compose all four major sub-systems. The definition of each building block includes its function and the type of its input and output interfaces. This leads to a small set of **interface types** that can be used to connect together any composition. Altogether there are fourteen service building blocks and five families of interface types. Most interface types are sub-typed to define more specific interfaces, but whole families are also dealt with together in some circumstances, motivating their high level classifications.

A primary separation in the architecture is between **operation** and **configuration**. Each is treated as a distinct phase from the other throughout. Configuration occurs at different levels of granularity. For instance, two further distinct phases are configuration of the **service** itself then configuration of **sessions** that use the service.

The great majority of the service building blocks are in continuous use as service operation proceeds, which is characterised by a continuous flow of management information around various control loops. Just one further service building block is required for configuration: a directory in which to put the configurations (which may be a soft state directory). This approach is deliberately intended to allow anyone to synthesise higher level sessions or services by merely creating new information structures that specify relationships between pre-existing operational services.

Unlike many high level architectures, the concentration on building blocks leads to a fairly detailed approach. As with many large-scale systems, the overall behaviour **emerges** from the behaviours of the parts, therefore it is important to be exact about the small things. However, a line has to be drawn to avoid the document becoming the equivalent of the concatenation of 25 Internet standards. Thus important points are highlighted, and detail is omitted where it would otherwise be simply repetitive.

The style employed makes heavy use of figures. This is useful for those that think in pictures, but might make reading difficult for people who think in text. Although there are accompanying descriptions, these would not stand without the figures. The figures suggest pseudo-code statements, which could be added if this proves to be a problem.

The architecture is primarily a logical one, focusing on the enterprise, functional computing and information viewpoints. The approach that will be adopted towards

©Copyright 2000, the Members of the M3I Consortium

defining the engineering and technology viewpoints is introduced, but this document avoids a systematic definition from these viewpoints, preferring occasional comments where important engineering or technology issues are at stake.

In the rest of this document, first we give an overview of the whole architecture in Section 2. We start by setting down general architectural principles that will guide the work. Then we outline the top-level architectural structure within which the rest of the document sits. This leads into a number of use cases (Section 2.3) that show in more detail the implications of the overall structure in particular scenarios. We draw scenarios from the M3I requirements specification [14], but we follow an order that builds up one use case from the last, rather than rigidly following the M3I requirements scenarios.

The rest of the overview section is an extended executive summary of the second half of the document. In essence the overview describes our 'bottom up' approach of defining building blocks and their interfaces and then how to compose them together in useful ways.

First, Section 2.4 puts the discussion in context by introducing the classes of applications that will use the building blocks, then we briefly describe and classify the service building blocks and describe our approach to defining interfaces between them. We then introduce the technique we propose for succinctly describing compositions of the service building blocks in Section 2.8.

The body of the document begins with section 3, where we formalise our definitions and notation.

Section 4 gives the definition of each **service building block**. Firstly we identify those building blocks necessary for **operation** of services (e.g. metering, network service itself, quality of service (QoS) control, charge advice) and the main information structures that are passed between them (e.g. session characterisations). This allows us to be clear about which further building blocks are necessary to **configure** (or create) operational services and sessions that use them. We end this section by attempting to classify the applications that will use our market-managed infrastructure, so that these may, to a degree, be treated as service building blocks, despite being external to the infrastructure.

Section 5 is a systematic classification of all the **interfaces** between pairs of service building blocks. Interfaces are placed in families and the main elements of each interface family are defined. However, likely differences between members of the same family are also highlighted, so that the end result is a set of interface and protocol definitions that will be required for the project.

Section 6 starts by defining a few useful compositions of service building blocks. These **service components** are then used to build a complete definition of the architecture that would implement some of the M3I scenarios. We end by highlighting limitations and further work and drawing conclusions.

Thus, we take a 'bottom up' approach by defining the service building blocks and interfaces for operation and configuration. This allows others to use the configuration building blocks in a top-down approach when defining their own services. However, the operational building blocks are *too* primitive to be composed directly into useful services. Therefore, we describe how a service operator would define and build

©Copyright 2000, the Members of the M3I Consortium

compositions of building blocks into service components for deployment of a specific architecture, thus filling in the middle between 'bottom-up' and 'top-down'.

2 Overview

2.1 Guiding principles

As well as defining how to build the various scenarios used as examples, this architecture is deliberately designed to facilitate the building of any other scenario. Each building block has the types of its possible inputs and outputs tied down, so that it is impossible to put the pieces together in illegal ways. However, what is legal is not always sensible. Therefore, we start with a few guiding principles to complement the more general guidance that example scenarios are intended to give. Further justification for these principles is given in [30].

2.1.1 Charging granularity

Where possible and efficient, granularity of charging should match granularity of service provision. That is, if service is provided per packet, it should be charged per packet. This principle is often broken for valid efficiency reasons. Typically, charging will be per some larger aggregation of packets, such as per flow, or per service level agreement. This requires major assumptions about the utilisation of the aggregates being sold. This is fine as long as there is awareness of the implications of these assumptions being broken, perhaps due to the sudden release of a new type of application. It should also be noted that new applications that exploit assumptions about under-utilisation are clearly encouraged by such assumptions.

2.1.2 Commercial openness

Any functions or protocols relating to commercial decisions shouldn't be embedded in the network infrastructure or protocols. Rather such functions and protocols should be placed at the application layer so that it is easy for providers to differentiate themselves and customers to switch between them. This principle is primarily due to the long time-scales necessary for infrastructure changes and their standardisation. Thus, ideally, the whole charging system should be as separate as possible from the transmission system.

2.1.3 Edge pricing and contracts

Each contract-related relationship (pricing, responsibility for service failure, charge advice, etc) should be between a single customer and a single provider for each invocation of service (most often discussed in terms of edge pricing [28]). This is particularly important, given the high number of providers on the average Internet path (five with nine not uncommon [34]).

2.1.4 Scalability

Wherever possible, compute intensive and data intensive operations should be distributed as far to the edge of the network as possible for scalability. The implications for support and manageability should be taken into account before taking this principle to the extreme of placing functions at the edge customer.

©Copyright 2000, the Members of the M3I Consortium

2.2 Top level architecture

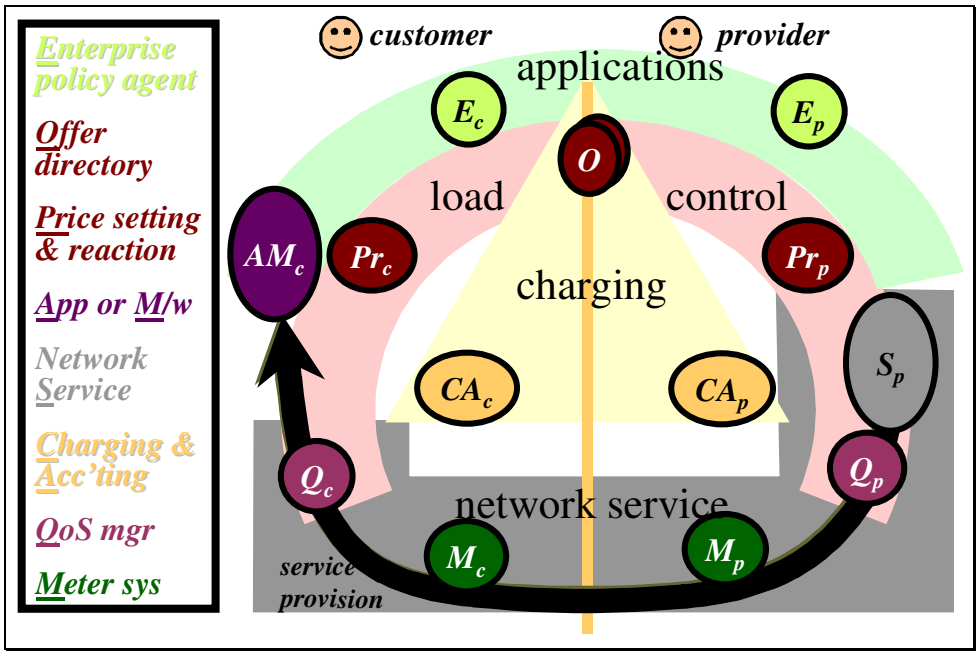


Fig 1 - High level architecture

Fig 1 shows a high level view of the architecture. The primary items to note are:

- the focus on the interface between a customer and a provider
- that network service flows outwards towards the customer, whichever direction traffic flows
- the four main categories of sub-system:
  - applications
  - load control
  - charging
  - network service
- That there is overlap between the categories. For instance an offer is central to both load control and charging. Or QoS management is used for load control as much as it is part of the network service.

This pattern is a general one in that it includes every aspect of a market managed multi-service Internet infrastructure. The first use case we will examine next goes into this architecture in more depth. However, some subsequent use cases omit some parts of this general architecture.

2.3 Typical use cases

Our approach is to allow many different compositions of systems, from the bottom up. When defining the precise building blocks to enable this, we had in mind a number of high level scenarios. Therefore, to give context for later, we will present a few of the top-level systems we had in mind with variations before we introduce the building blocks. Neither the constituent parts of these systems nor the interfaces between them are precisely defined at this overview stage. Only when the technicalities within each part have been clarified will it become possible to define the various interfaces between them. Also these top-level descriptions shouldn't in any way be construed as the only

©Copyright 2000, the Members of the M3I Consortium

possible scenarios, some more of which are given in the M3I specification of requirements [14].

The essence of the market-managed approach is to bring the customer's systems into the control loop through a price mechanism. Therefore it is inappropriate to focus solely on the provider's systems, only worrying about the customer interface when it comes to bill payment. This approach was common in the past when designing communications management and charging systems for telephony and has unfortunately shaped much recent thinking behind Internet charging systems. Instead, in all cases, we focus on the interface between one provider and one customer, giving scenarios for different customer and provider types. This **stakeholder view**[1] results in some repetition of the systems shown in each stakeholder's domain. An alternative to showing some systems in both the provider's and customer's domains would be to show a functional diagram with just one of each system without being specific about the fact that each stakeholder operated one. We take the position that the stakeholder view is primary in setting the context and where possible should be made explicit rather than hidden, even when focusing on other viewpoints. In particular, this approach highlights a number of cases where an interface is required between two systems of the same type, because in practice both customer and provider operate one and they need to talk to each other. Where each of provider and customer operate the same type of sub-system, X, we will distinguish between them with subscripts X_p and X_c.

2.3.1 Edge-centric use case

Fig 2 shows a scenario or 'use case' of particular interest to the M3I project, where the network service is monitored and controlled as far as possible from the edge-customer's systems. The edge-customer is given the incentive to behave as the provider desires through the market mechanism of price control. The motivation of moving functions to the edge is to reduce the load on the provider that fine-grained price control would bring if a more traditional system were used (e.g. as in Section 2.3.2).

This use case is intended to be generic, but can be used to describe either half of the scenario termed **dynamic price handler** in the M3I requirements document [14]. The architectural approach is to allow each network service provider to offer a different service proposition to its own customers, hence the focus on half the scenario. The whole scenario can be constructed by considering the use case described here to be repeated symmetrically for both customer-ISP interfaces (I5 & I6 in the requirements document). The clearinghouse use case (Section 2.3.4.2) can then be used for the interface between the two customers (I5), while the inter-network use case (Section 2.3.3) can be interposed between any number of network providers on the path. The design of the whole dynamic price handler scenario is discussed in the M3I price reaction design [31].

The line down the middle separates the two stakeholders. The legend on the left expands the abbreviations. The arrows represent interaction between sub-systems and point in the primary direction of service. Manual intervention is represented by the face

[1] The stakeholder view is usually called the enterprise view in the distributed systems field [26], but the term 'stakeholder' has been chosen for the M3I project, partly because it is less suggestive of purely commercial organisations. The term enterprise policy is also used interchangeably with stakeholder policy.

©Copyright 2000, the Members of the M3I Consortium

symbols at the top with user interaction arrows being shown in light grey. Some interactions that concern detailed configuration are omitted for this overview. Although the numbers on the interactions give the general order of how things get started, once the system is in operation, interactions run in loops of varying sizes and no significance can be attached to the numerical values - they are simply labels for reference.

The general idea is for the representatives of each stakeholder to define their policy, then the customer interacts with her application to draw service from the network provider. This flow of data causes a flow of management information through the pricing and charging systems, which self-manage the system within the policies set by the stakeholders. Thus, in general, the sub-systems at the bottom operate on fine-grained time-scales while those interactions towards the top occur much less frequently.

The network service is modelled as a single logical routing and forwarding function, $\mathbf{S_p}$. The direction of service is outward from the network service, but the direction of traffic may be either inward or outward. The network service effectively encapsulates all the connected network services behind. It handles traffic the customer presents to it on behalf of all the onward networks, and it handles all traffic addressed to the customer from anywhere else, presenting a single service interface to the entire Internet. Although this use case doesn't have to say anything about what pricing is used, this edge network service encapsulation naturally matches the edge pricing approach [27][28].

This is not to say that the customer cannot simultaneously be using interfaces with competing network services, rather the sub-systems shown here are specific to the provider in question. Of course the network service will also have interfaces with other customers, but again, the sub-systems shown here are specific to the customer in question.

This exclusion of the aspects of both stakeholders systems that are not dedicated to each other covers both their *network* service interface and their *communications management* interfaces. Thus, for instance, the charging and accounting system, $\mathbf{CA_p}$, does *not* represent the full monolithic system serving all customers. It represents the aspects of the charging and accounting functions dedicated to this one customer, which in practice might be a vertical slice of a larger system that might either be monolithic or more distributed. However, note that this includes the aspects of the charging system that aggregate across customers[2]. This same point applies, as far as possible, to the other sub-systems shown. However, some of the higher level sub-systems are more likely to be common across all customer interfaces. For instance, it is highly likely that the provider's enterprise policy concerning one customer will be common to all similar customers. Similarly, the price setting function will probably take into account the demand from all customers. Similarly, but on the customer side, her policy with respect to one provider is likely to be common to most other providers. But the point is that, whether there is customisation or not, focusing on a single provider-customer interface allows for either case.

[2] The apparent symmetry of the diagram doesn't imply that each customer's system will be as large or complex as a telephony billing system. The operational load per-customer in a charging system is trivial, involving the odd multiplication and addition per chargeable event. The complexity of one customer's operations viewed in isolation is also trivial. The complexity and load of a telephony billing system stems from the sheer numbers of customers and their variety.

©Copyright 2000, the Members of the M3I Consortium

We deliberately don't specify where all the sub-systems run that are shown above the path of network service provision. This use case applies whether they are embedded on routers, running on the customer's and provider's respective support systems or on third party hosting services. We leave it open, whether the customer has a single host, or a large network. Clearly, however, the network service is implemented on routers, while the customer application or middleware run on her host(s). Provider QoS management might run on a router or specialist network device (e.g. a shaper). In the case of the customer, QoS management might also run on a network device, but it is most likely to run on the end-host(s).

We will now briefly run through the edge-control use case in Fig 2 in more detail.

First the network provider decides on an offer, (or service plan) to put to customers to sell its service. In general, it may make multiple offers for the same service, each of which captures a different type of market. Privately, it also defined its policy on how the pricing of these offers will change as demand evolves. It sets its pricing policy using the interface (1) of its private **enterprise policy directory, E_p**, and it places the offers[3] (2) in a more public **offer directory, O**, perhaps along with those of other providers. In parallel, the customer decides on her own buying policy for the type of application/task in question, which she sets using the interface (3) of her **enterprise policy agent[4], E_c**. This policy may be delivered by default with an application on installation, but the customer would be able to tailor the default if required. Alternatively, on initiation of a communications session, the initiator (or more likely the party paying) might define the default buying policy for all participants and deliver it to them with the session description. This may be defined per session, or perhaps a default would be applied by the session creation application.

[3] An offer could strictly be called an 'offer policy', but we just use 'offer' for brevity.

[4] Note that the enterprise policy sub-systems are described as a directory in the case of the provider and an agent in the case of the customer. In both cases they are agents of their respective stakeholders supported by a directory of policies. However, in the provider case, it is unlikely the function of deciding how to sell service will be automated, therefore the role of the agent is minimal - reduced to looking up policies in a directory, hence we just use the term directory.

©Copyright 2000, the Members of the M3I Consortium

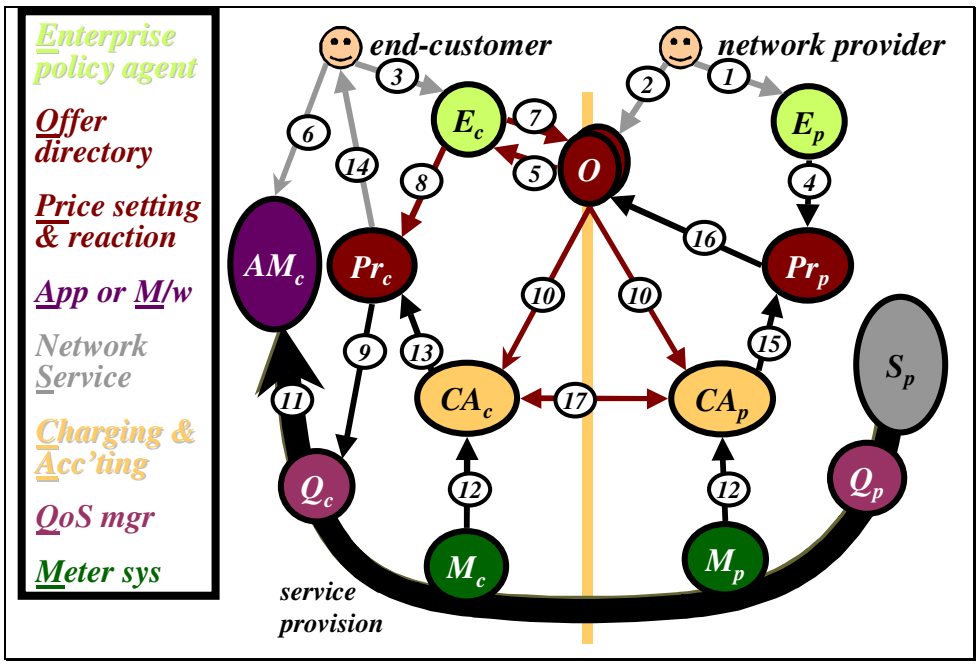


Fig 2 - Edge-centric use case

The **price setting** function, **Pr_p**, monitors its enterprise policy directory for changes in order to pull in any new policy on how it should behave (4). Similarly, the customer's enterprise policy agent monitors new offers (5).

Let us now assume the customer launches a new **application** that needs communication services, or a new instance of customer **middleware** launches, perhaps to support a new communications session. We show either as **AM_c** as it is irrelevant to the network whether an application is using the network service directly or some middleware is interposed. The opening of a communications channel will be detected by the enterprise policy agent (perhaps implicitly using the mechanism in [29] or by being explicitly notified by the application or middleware). Based on its buying policy, this agent may accept a suitable offer (7) or find it has already accepted a suitable offer for the particular type of communications session.[5] At the same time, the agent will configure the **price reaction** function, **Pr_c**, with its QoS control policy for the type of session and the tariff in force (8). In practice it would be likely that a reference to the tariff would be passed to the price reaction function for it to receive immediate updates. In turn, the price reaction function initialises the **QoS manager**, **Q_c**, for this session (9). In parallel to all this, acceptance of the offer triggers configuration of both **charging and accounting** systems, **CA**, with the new tariff and its scope, as defined in the offer acceptance (10). This completes both the policy set up phase and the set up for a specific session (meter and charging system configuration is not shown in this use case).

As the application starts to use the service of the network (11), the **customer's QoS manager**, **Q_c**, keeps the traffic within its QoS policy. It may do this itself, or by signalling QoS requests along the data path to the **provider's QoS manager**, **Q_p**. The data load and any QoS requests are **metered**, **M**, as they pass. Before going into detail, we will briefly explain why, in this edge-centric use-case, both the customer (**M_c**)

[5] Sometimes offer acceptance is implicit in use of certain type of service.

©Copyright 2000, the Members of the M3I Consortium

as well as the network provider (**M_p**) meter the load. The customer's readings are fed back through the customers charging system to the price reaction sub-system[6], which regularly tunes the QoS policy. The customer's enterprise agent can also assess these readings against competing tariffs (not shown) in order to make longer-term decisions concerning switching tariffs. The provider's readings determine how much this customer is charged and may be used to alter the price, although usually only when collected together with the level of demand from other customers. There is also scope for taking advantage of the redundancy between the two charging systems, which will be discussed at the end of this section.

In detail, the customer's metering sub-system, M_c, reports usage (12) to the customer's charging and accounting system, CA_c, which calculates the consequent charge on a regular basis using the current tariff. This charge is fed back to control service usage and quality through *two* possible routes:

- For functional usage (controlled directly by interaction with the application) the charge has to be presented to the user (14) in the expectation that this may affect how the application is driven (6) either in the long or short term.
- For non-functional usage (where control has been delegated to the QoS manager by the application or user), the price reaction function, Pr_c, can regularly request advice of the charge (13) and consequently alter the QoS policy (9) it gives to the QoS manager, Q_c. As before, this may lead to direct edge-control of QoS, or to an adaptation of the earlier request to the provider's QoS manager, Q_p.

In practice, the control of nearly all QoS mechanisms has to be exerted at just one of either the sender or receiver. But most involve some degree of intercommunication between the two. For this high level view, the figure represents the way control would be exerted at one end if it were needed. For each specific mechanism some of the interfaces within the customer side might actually be end-to-end interfaces between two (or more) customers (see [31] for details).

On the provider's side, the provider's metering sub-system, M_p, reports usage (12) to the provider's charging and accounting system, CA_p, which calculates the consequent charge on a regular basis using the current tariff, usually on a longer term and aggregated basis. The revenue and utilisation from this customer (15) is reported to the price setting function, Pr_p, which might occasionally update pricing (16) in the offer based on reports across all customers and the price setting policy. This may also require manual intervention. Alternatively, it may have a more discriminatory role, and alter pricing more regularly just for this specific customer. Note this is not the same as congestion pricing, where a price is attached to the technical fact of congestion on a path. In such a case, the price of congestion is constant over long periods. It is congestion that is altering, which in turn makes the charge vary.

So far we have focused on using the charging and accounting function to set and react to pricing, but, of course, it also has the more basic role of calculating the charge due for services delivered. Traditionally, the provider's charging system has served this purpose. In this use case, the customer has her own charging system in order to make price reaction responsive without loading the provider, therefore she also has the

[6] Note that 'price reaction' is strictly a misnomer, as it typically reacts to the charge, which is the product of pricing and usage.

©Copyright 2000, the Members of the M3I Consortium

ability to check the provider's idea of how much she owes (17) against her own version. Alternatively, the provider may wholly delegate the charging function to the customer, asking her to regularly report her own bill (17). This can then be used to drive price setting as before (15). However, the provider will need some way to trust the customer's reports, perhaps operating its own charging system during a random sample of the customer's billing periods of to audit that the customer's reports are trustworthy [30]. Note that delegating primary responsibility for charge calculation to the customer puts extra reliability and accuracy requirements onto the customer's charging system. These burdens are not present if it is merely used for price reaction.

Finally, we should note that there is also a longer-term price reaction control loop that is not apparent in Fig 2. The customer's enterprise agent can also take the records of usage being reported to the accounting system and hypothetically apply a selection of competing tariffs to them to evaluate other competing offers of network provision.

2.3.2 Edge-control use case

This edge-control use-case (Fig 3) is very similar in most respects to the previous edge-centric case (Fig 2). Control of QoS is still focused at the edge customer's system, but not monitoring of charges. In order to be able to dynamically react to the level of her charges, the edge customer relies on regular charge advice feedback from her provider instead of calculating her own. This use case would not be appropriate in highly dynamic charging scenarios, but has relevance otherwise.

This case is shown in Fig 3, where the numbered labels on each interaction have been preserved from the previous case, leading to some gaps in the sequence. All the steps in the set up phase (1 to 10) are identical to before, except there is no customer charging system to apply the offer acceptance to in step (10).

Once service is invoked (11), only the provider meters it (12). The customer's QoS manager will still be monitoring the load to keep to its very short term control policy. However, the medium term control loop for functional and non-functional traffic can no longer be wholly within the customer's domain. Both the price reaction function (for non-functional) and the human customer (for functional) have to receive their charge feedback (13 & 14 respectively) from the provider's charging system, not having one of their own. Other than this, the feedback loops continue as before (9 & 6). The provider's price setting feedback loop is unchanged (15 & 16).

Although the tight QoS control loop might react on a per-packet basis, the frequency with which the customer's price reaction *policy* changes can sit anywhere on a wide spectrum in different scenarios. It may be refined every few seconds, or at the extreme, it may never be altered, being set at a standard default, much as transmission control protocol (TCP) is today[7].

[7] For example, if explicit congestion notification (ECN) [32] gradually replaces packet drop as the primary congestion control signalling mechanism on the Internet, congestion pricing isn't essential for it to work, let alone adaptive policies controlling the reaction to congestion pricing.

©Copyright 2000, the Members of the M3I Consortium

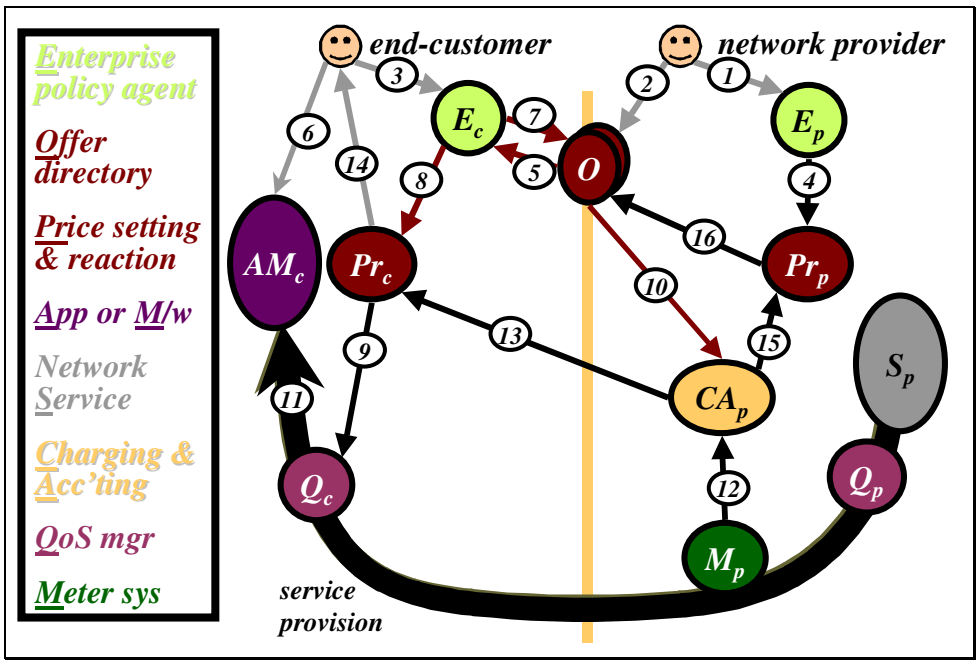


Fig 3 - Edge-control use case

Billing can only be done traditionally in this use case, and the customer has no means to check its accuracy as before (step 17 is therefore absent).

As for the previous edge-centric use case, the edge-control case can be used to describe either half of the scenario termed **dynamic price handler** in the M3I requirements document [14]. However, the highly variable pricing envisaged in that scenario is likely to compromise the performance of this edge-control case. Thus the edge-centric case is recommended for that scenario.

2.3.3 Inter-network use case

We now present a use-case that focuses on how market control might be used between two network providers. In traditional networks, as any point approached capacity, admission control would kick in to dampen further demand. The M3I project is interested in an alternative where the end-systems are made aware of approaching congestion and encouraged to behave appropriately through a rise in price. Similarly, when spare capacity is available, a drop in price encourages an increase in demand. This use case shows a high level view of the sub-systems necessary across an inter-network interface to support such a mechanism.

Many of the scenarios in the M3I requirements document include more than one network provider as a general case of a single provider. This use case is intended to be applicable to all of these scenarios. As long as the edge contracts principle is adhered to, any edge network provider encapsulates all the interfaces of more remote providers. Therefore, this use case can represent any number of network providers on a path, by simply chaining it together repeatedly. As always, the use case focuses on the interfaces between the parties to enable such composition of bigger use cases from smaller ones.

The sub-systems in Fig 4 appear very similar to the previous edge-customer cases, the primary difference being little scope to directly alter QoS. Thus the price reaction function feeds forward to the next price setting function, instead of controlling QoS as in

©Copyright 2000, the Members of the M3I Consortium

the earlier edge cases or admission as in the traditional case. Because of this similarity, the numbered labels on each interaction have been preserved from the previous cases, leading again to some gaps in the sequence. However, there are some further real differences beneath the superficial similarity.

First, however, we must explain the provider-customer relationship between two parties that are both ostensibly providers. The idea is to focus on the provider role of the right-hand party and the customer role of the left. The reader should imagine a similar but opposite diagram superposed on this showing the provider aspects of the left-hand party and the customer aspects of the right. This technique helps disentangle the confusion caused by two providers simultaneously offering each other service in a somewhat symbiotic relationship. The systems used for each role simply treat all prices or charges as the negative of the other, resulting in a price that is the balance between that offered and that accepted. Each party can use the same single accounting system for its two roles, but the purpose of the system and hence security context will differ depending on the role. The provider's system is tasked with finding the liability of the customer to charges. The customer's system is tasked with protecting the customer from any false charges (whether intentional or not). Thus, after combining the two roles, each system has to work within both security contexts.

Note that, although we don't show the provider role of the left-hand party with respect to the right, we do show part of its provider aspect with respect to other (unshown) parties further to the left. The parts shown are sufficient to highlight the interfaces between its customer aspect and its onward provider aspects.

Unlike in the edge-customer cases above, it is more likely that the price setting and reaction functions would be implemented in a distributed fashion across each party's routers. Although enterprise policy is still likely to be set centrally, it is also likely to be disseminated to every router, to allow local, autonomous decision-making. However, charging and accounting are likely to run on associated support systems with metering probably done using specialist hardware. Again, meter and charging system control isn't shown for clarity.

Steps (1 to 5) to set up and apply the respective enterprise policies and the offer are similar to before, only differing in the nature of the policies and the offer, which are likely to be very different from the edge-case. In particular, they are likely to be phrased in terms of aggregate measures rather than detailed packets or flows. Also, it is likely that internal policy as well as published pricing will be differentiated with respect to different classes of routes. For all these reasons, the protocols used across these interfaces are very unlikely to be the same as those used at the edge of the network between superficially similar functions.

Only one offer is shown, rather than the multiple offers likely in markets at the network edge. Offer acceptance (7) used before is therefore unnecessary, as use of the network service interface is likely to be taken as implicit acceptance of a single offer. However, there is no inherent reason why multiple offers or explicit acceptance of them couldn't arise in this scenario, although it is unlikely.

As before, the customer enterprise policy agent will be monitoring multiple providers to find the best deal, corresponding to a re-routing decision. Still continuing with our earlier steps, the customer enterprise policy agent will configure the price reaction function with its QoS control policy relevant to the provider's offer (8). However, as

©Copyright 2000, the Members of the M3I Consortium

already pointed out, the primary difference from the edge case is that once alternative routes have been exhausted, quality control can only be achieved indirectly by altering onward pricing. So, instead of controlling QoS directly, the price reaction function is likely to be closely related to onward price setting, most likely combined within the same function. Thus our earlier step (9) (applying policy to QoS management), would be replaced by invocation of the internal interface between price reaction and setting within the combined function.

Again, as earlier, the the offer acceptance including the tariff is also communicated to the two party's charging and accounting systems (10) so that, as traffic flows (11) and it is metered (12), charges can be calculated. The customer's pricing function needs to understand the cost of its traffic (13) in order to decide whether it should try to dampen onward demand or encourage it. The provider's price setting function similarly needs to know current demand for its network (15) in order to set its prices (16) (it will also take into account its costs as a customer of previous networks in the chain).

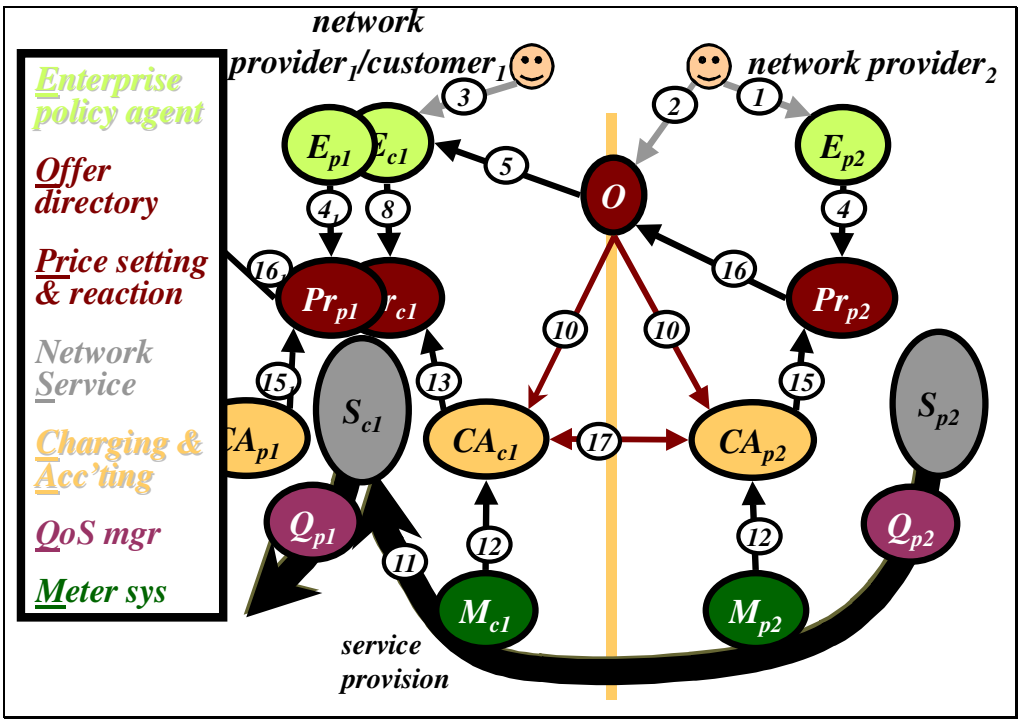


Fig 4 - Inter-network use-case

As in the edge-centric use case, the two parties will reconcile their billing records in order to determine the level of settlement required (17).

Note the important **wholesaling pattern of interaction** between the two charging and accounting systems of the left-hand party in Fig 4. There is *no direct inter-communication* between the charging and accounting system representing the wholesaler as a customer, CA_{c1}, and the one representing it as a provider, CA_{p1}. The charge records of the costs that the left-hand provider incurs *as a customer* feed instead into its price reaction and then price setting functions. The charge records of the revenues it earns *as a provider* also feed its price setting function. How a wholesaler meters its customers can be completely different from how it is metered as a customer of other providers. There is no need for compatibility of the basis of metering across a wholesaler, as long as any two systems of measurement can be related together in money terms. Of course, there is pressure for standardisation of the basis of

©Copyright 2000, the Members of the M3I Consortium

metering to simplify the supply of metering and charging equipment, but there is no absolute necessity. This is the result of adhering to the principle of **edge pricing and contracts** (Section 2.1.3).

2.3.4 Guaranteed service provider use case

The M3I requirements document [14] describes a scenario where guaranteed service is synthesised from a rudimentary dynamically priced, variable QoS network service, based on explicit congestion notification (the GSP/ECN scenario for short). In that document it is considered necessary to combine the two rudimentary roles of risk broker and clearinghouse into one service in order for a separate stakeholder to have a chance of operating a practical business.

In this document, we attempt to show that it is possible to separate these two roles into stand-alone businesses. We also attempt to allow heterogeneity of service propositions by *not* requiring the risk broker to span both ends of a communication.

2.3.4.1 Risk broker

Fig 5 shows one use case to achieve the M3I risk broker requirements. Note that, as in earlier use cases, the focus is on one end-customer at a time to allow heterogeneity of service propositions. The use case however differs in its details depending on whether the end-customer is sender or receiver. Nonetheless, the one figure is used for either case as they only differ in their description and can use a common diagram. For this use case, both (all) ends are assumed to pay, therefore the terms end-customer and end-user are equivalent. The clearinghouse use case (see later) will add the ability for one party to pay for another.

The most obvious difference from previous use cases is that the risk broker takes the full end-customer relationship from the network provider, offering charging, management (pricing) and service interfaces. The risk broker doesn't operate a network service (routing and forwarding). However, it does involve itself with the traffic flow in order to affect the quality of the network service it re-sells.

To preserve continuity, the steps have been numbered substantially as in previous use cases. Because there are three parties and two interfaces, many of the sub-systems appear in triplicate (one for each party) and most of the steps appear twice (one for each interface). Therefore, we have suffixed the risk broker sub-systems with a 'b' (for broker) and we have suffixed the steps related to the end-customer interface with an 'e' (for end). The suffices 'c' for customer and 'p' for provider remain across each interface, so that the broker sub-systems have two suffices.

Fig 5 appears relatively complex, however it should be remembered that it represents interactions during both configuration and operational phases and for two pairs of parties, and one with a fairly complex tariff. Most of the interactions only occur rarely, being for long-term configuration. We start with the **configuration** phase.

Glossing over initial policy definition, which is unchanged (1), the use case starts with the network provider making an offer, O, (2) that includes charges for sent and received traffic. The price for received traffic is per explicit congestion notification (ECN) marked packet [22]. The price for sent traffic might be on any other basis. The risk broker programmes its enterprise policy agent (buying agent) to look for and accept the best offers from a number of network providers (3). We have shown a risk broker that doesn't operate a router, preferring to switch providers on a more long-term basis,

©Copyright 2000, the Members of the M3I Consortium

probably switching connections at the link layer. A risk broker could operate a router to switch providers depending on price, but would then have to pay any charges for each unused link to each provider. Offer announcement and acceptance (5, 7) proceed as before. Of course, the risk broker doesn't run any applications itself, hence step 6 from previous cases is missing. In parallel with offer acceptance, the buying enterprise policy agent, E_{cb}, configures its price reactor with a policy, Pr_{bc}, the purpose of which will be described below.

Although it doesn't run any applications, the risk broker does retail the network service it is buying. It therefore sets its own pricing policy (1e & 4e) and makes an onward offer of guaranteed service (2e). This offer sells what appears to be reserved capacity, but which in reality is synthesised by the risk broker from the congestion priced service it buys. In the M3I GSP/ECN scenario that motivates this use case [14], the end-customer must give a maximum duration for the reservation - the longer the duration the higher the risk and therefore price per unit time. An alternative might be for the risk broker to describe the formula by which its price per unit time rises with duration. As long as it also charges a fixed fee per reservation, this can also be incentive compatible and doesn't require the end-customer to commit to a duration in advance. A third alternative might be for the risk broker to offer a price per unit time independent of duration that on average covers the cost of the risk. This would emulate traditional telephony charging.

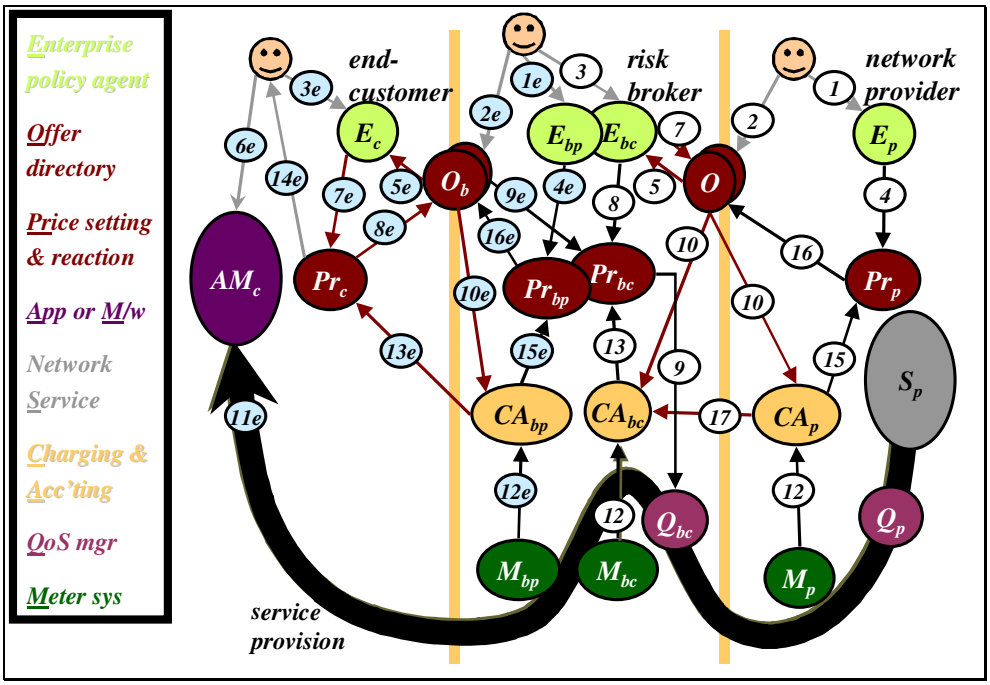


Fig 5 - Risk broker use case

The complete tariff in required in the M3I GSP/ECN scenario includes a price per volume as well as a duration-based price. This is often termed an 'abc' tariff after the formula C = aT + bV +c [37]. C is the total charge, T is the duration or the reservation, V is the volume transmitted and a, b & c are the price coefficients of each parameter including a constant to cover per session costs. As explained above, the price per unit time, a, may depend on the duration the customer wishes to commit to.

A variant of this scenario might be motivated by the risk broker's desire to simplify retail pricing. In such a case, the volume charge might be removed (b=0) and the duration

©Copyright 2000, the Members of the M3I Consortium

price might remain stable over long time scales so that end-customers could monitor and accept it manually as in most present retail communications markets. In such a case, the end-customer's policy agent and price reactor would simply disappear from Fig 5, being replaced by manual methods. However, in the general case, pricing could vary often, even if it were guaranteed fixed once accepted for a session. Also, with volume charging, the ultimate charge is only as predictable as the volume itself, which is often unknown in advance for many applications. Therefore in this use case we focus on the sub-systems necessary for an automated end-customer response.

The end-customer sets her policy across all tasks (3e), which is then used while monitoring multiple provider offers (5e). Once a specific task is initiated (6e), the customer's enterprise policy agent determines its price sensitivity against quality for this task (7e). Then the price reactor, Pr_c, is able to accept the offer itself (8e), including committing to the duration of the session if relevant (see above). This is slightly different from earlier use cases, where an offer was accepted for multiple sessions by the enterprise policy agent. In this case, because an abstraction of a communications session is offered (as opposed to the raw packet granularity service), a separate request must be placed to accept the offer for each communications session. Per session decisions can be controlled by a single policy, therefore the power to make them is delegated to the lower level[8].

Below, we present two alternatives for offer acceptance (8e), QoS request (9e) and QoS management (9). The main differences are in the handling of the QoS specification and the duration commitment, which together form the offer acceptance, but need to end up in different places and take different routes in each alternative.

However, before diving into this detail, we will assume offer acceptance has been completed and QoS request and management are proceeding in parallel. Just a single step is then necessary to complete the policy configuration phase. Both charging and accounting systems (of risk broker and network provider) ensure that they have an up to date copy of the offer acceptance of their respective customers, so that they can calculate charges correctly (10e & 10).

Returning to steps 8e, 9e & 9, first we will cover the simpler alternative for **non-corporate** end-customers.

Step 8e involves the customer sending a request to the risk broker to guarantee a certain quality of service for a certain duration and for a certain session scope. This set of information is nearly identical to that in an RSVP message but with the addition of the duration information, and no need to communicate with routers to pin a route. Thus much of the RSVP protocol can be re-used, but re-implemented as an application layer protocol. The direction of the traffic for which a guarantee is required will also have to be made clear, which will replace the PATH/RESV distinction of RSVP.

The duration of the guarantee is purely to do with the provider side of the risk broker, and therefore need only be destined for the offer acceptance directory on the provider side of the risk broker. However, the QoS specification must be forwarded on further, as it is also applied to the risk broker's price reactor (9e), which is on its customer side, facing the network provider's service. The price reactor takes this QoS specification

[8] Note steps 7e and 8e map to steps 8 and 7 respectively in previous use cases.

©Copyright 2000, the Members of the M3I Consortium

and calculates the QoS specification it would like its own QoS manager, Q_{bc}, to work to (9), given current pricing and the enterprise policy in force.

The alternative **corporate** approach to steps 8e, 9e & 9 is shown in Fig 6. All other interactions are identical to the main use case and are therefore not explicitly labelled. This alternative would be used if it was necessary to guarantee QoS in the customer's own network between the end-customer and the risk broker (perhaps if it were a corporate customer with a degree of contention for its own network). We will assume this reservation would be free of charge, as it is in the customer's own network.

In this corporate alternative, the QoS specification and the duration commitment no longer travel together direct to the risk broker's offer acceptance directory. Instead they are split, with the former sent via the routers on the path, in the traditional intserv manner (9e - shown multiple times), while the duration commitment still takes its original path direct to the offer acceptance directory (8e). Note the original step 9e in Fig 5 is no longer required (shown dotted and crossed out in Fig 6).

The QoS specification is inserted into the network traffic as RSVP signalling, but it is filtered out at the risk broker to ensure it ends up at the broker's price reaction function, Pr_{bc}, as before. Step 9 would then continue as before, setting the risk broker's QoS control policy.

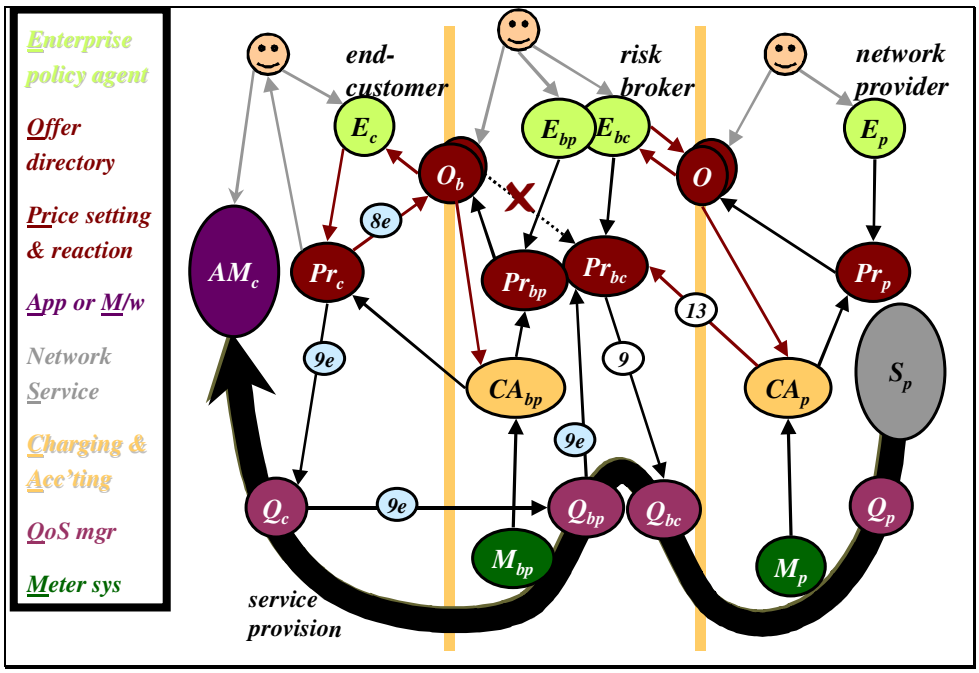


Fig 6 - Alternative risk broker use case

However, note that in the corporate alternative there appears to be no communication of the required QoS specification between one end-customer and another across the network in the middle. The QoS specification appears to be inserted from both (all) ends, then filtered out before it reaches the networks in the middle. In the non-corporate alternative, the QoS spec was never placed in the traffic in the first place and therefore would certainly never be seen in the middle. Although this avoids the need to signal QoS to every backbone router, thus avoiding the alleged scalability problems of intserv, it still leaves the problem of how the QoS specifications are communicated in the first place to the customer's price reactors, Pr_c, at all ends.

©Copyright 2000, the Members of the M3I Consortium

QoS knowledge may be hard-coded into the application, but a more likely scenario is that it would be declared in a session description used to configure both senders and receivers (part of session control - not shown). This is a generalisation of the pattern used in the integrated services architecture where the sender 'somehow' knows the QoS to recommend and then signals this through to the receiver at the same time as it signals potential reservations to relevant routers on the network path. On the contrary, this use case avoids the assumption that there is any homogeneity of QoS technology across the whole end-to-end path. Instead the QoS specification is signalled end-to-end 'over the top of' the middle. Nonetheless, the QoS specification in the session description does still need to be understood by all ends (just not the 'middles'). An adapted version of the standardised message content of RSVP is likely to be used in both this session description and in the offer acceptance used to request guaranteed service (see also [31]). Note that, this is a generalisation of intserv, as intserv only solved the problem of transporting the QoS specification from sender to receiver, not from session initiator to sender (if they were not the same party).

Once the network service starts to be used by the end-customer (11e in Fig 5), we enter the **operational** phase of the risk broker use case.

Before stepping through how charging is applied and price feedback occurs, we will describe how the risk broker delivers its promise to guarantee QoS, taking first the receiver, then the sender. Note, whether this works in practice is for investigation within the M3I project.

At the receiver, the risk broker's QoS management function intercepts congestion marks on their way to the receiver. It pays for these itself. In order to minimise its own costs, it feeds back signalling to the sender in order to control its rate, but without asking the sender to drop below the limit of the QoS guarantee it has retailed to the receiver. If the broker allows the congestion marks to continue on to the receiver, the broker would certainly have to filter out any feedback the receiver returned so that it didn't interfere with the risk broker's own feedback. The receiver's risk broker would have to be able to accept a reservation from the sender if the sender were paying for the receiver. This case is discussed further in the next section on the clearinghouse and in [31].

Moving to the sender, the most likely scenario is that congestion pricing will only be applied by the receiver's provider. The sender's provider might offer a more stable pricing regime. Therefore, a risk broker in front of the sender's network provider will probably not have to smooth price, but it may have to smooth QoS to keep to a guarantee it has sold to the sender. So the sender's risk broker might absorb congestion feedback from the receiving end and use it to shape the traffic coming from the sender. However, this area is little understood and needs further investigation in the project.

Note that, although the risk broker is more relevant for a receiver than a sender, in most communications each end-customer is both a sender and a receiver. Therefore, each risk broker will, at least, have a role at both ends, even if only used half the time at each.

A highly likely scenario is that at least part of the receiver's congestion charges may be paid for by the sender. In this case the sender could pay the receiving risk broker's stable charges, rather than the dynamic ones of the network provider at that end. If

©Copyright 2000, the Members of the M3I Consortium

there were no risk broker at the receiver's end, the receiver's network provider would charge the sender end-to-end for the dynamic congestion feedback it passes back. There would most likely be a clearinghouse between the two. In either case, there would be a role for a risk broker interposed directly between either the receiver's provider and the clearinghouse (to protect the clearinghouse) or between the clearinghouse and the sender (to protect the sender). Each case simply involves a merger of the risk broker and clearinghouse cases (see next) in slightly varying configurations.

Having described how the broker guarantees QoS and price, we now continue to describe how the charging systems operate in this use case. The general arrangement of the broker includes two charging and accounting systems: one representing its provider aspect using the 'abc' tariff, and the other monitoring its interests as a customer of the network provider, using the congestion pricing tariff.

On its provider side, the broker's metering system, M_{bp}, measures the volume of traffic for each session and passes this to its charging and accounting system, CA_{bp}, (12e) to calculate the volume charge element of the 'abc' tariff. Earlier CA_{bp} was notified of the offer acceptance (10e), so it can also calculate the duration and fixed charge to advise the total charge to the end-customer's price reactor (and onward to the end-customer if required) on a regular basis (13e & 14e).

From the broker's customer aspect, meter system, M_{bc}, measures ECN marks in the traffic and passes them to the customer side of the broker's own charging system, CA_{bc}, (12). This isn't necessarily audit-grade, but gives the feedback needed into the price reaction function, Pr_{bc}, (13), which feeds into onward price setting. Note that the broker's customer-side charging system, CA_{bc}, is logically separated from the provider side, CA_{bp}, conforming to the **wholesaling interaction pattern** described in Section 2.3.3.

The network provider's meter system, M_p, measures ECN marks in the traffic and passes them to its charging and accounting system, CA_p, (12) to enable it to bill the risk broker. This *is* audit grade accounting, but the broker might still want to reconcile its results with its own accounting system if required (17).

It is feasible for the risk broker not to operate its own charging system, instead using charge advice from the network provider's. This causes more charge advice load across the interface between them, but leads to a simpler system for the broker. We show this alternative charge advice arrangement in Fig 6 as well as showing the alternative QoS arrangement described earlier.

Finally, the two provider side charging and accounting systems (CA_{bp} & CA_p) regularly feedback current charges to the two provider's price setting functions (15e & 15). As before, these might in turn alter the pricing of future offers (16e & 16).

This completes the risk broker use case. We have shown how a risk broker can be separated from the clearinghouse function, by implementing it at either or both ends of a communication.

2.3.4.2 Clearinghouse

The use case described here uses the end-to-end clearing architecture of [28], rather than the embedded clearing of [38] or the centralised clearing of the open settlement

©Copyright 2000, the Members of the M3I Consortium

protocol [39]. The primary reasoning for this choice was to avoid the tight coupling with routing and quality of service of the alternatives, in order to promote more openness.

The clearinghouse use case involves at least five stakeholders who are denoted by the suffices defined in Table 1 and used in Fig 7 & Fig 8.

stake-holder suffix	stakeholder	service used by	offer accepted by	use case steps suffix
a	application service provider (ASP)	u	u	e
u	end-customer (user)	-	-	-
p1	network provider 1	a	h	n1
p2	network provider 2	u	h	n2
h	clearinghouse	-	a	c

Table 1 - Clearinghouse use case stakeholders

The third column of Table 1 also summarises which stakeholder uses the service of each other stakeholder, as shown in Fig 7. The ASP offers its service interface to the end-customer on an end-to-end basis (fat arrow labelled application service). They both also use the service interfaces of their respective network providers (fat arrows labelled network service). The clearinghouse doesn't offer or use a service interface. It only has management (pricing) and charging interfaces.

Contractually, a clearinghouse might take responsibility for the network service it is retailing, but it only offers it by reference. If it does take responsibility for quality of service, it will clearly make best efforts to buy its services from network providers who can offer sufficient quality of service themselves. If there is a service degradation or failure, it simply intermediates between its customers and suppliers, on the one hand giving refunds and support, on the other requesting improvement or repair.

In Fig 7, service offers, O, are shown sitting on each interface between each pair of parties. These are the management information about services as distinct from the service interfaces themselves. The fourth column of Table 1 also summarises who offers and who accepts which offer. The network providers offer their pricing (O_{p1} & O_{p2}) to both end-customers and to the clearinghouse. In this case, these offers are ignored by the direct retail customers, only being taken up by the clearinghouse. Its business is to accept offers from multiple network providers and sell onward a combined offer of end-to-end service. So, in turn, the clearinghouse offers its pricing for the end-to-end service to either end-customer (O_h). Only one (the ASP) takes up this offer. It then makes its own offer (O_a) to its end-customers, which bundles any application service charges with those for transmission QoS from the clearinghouse.

The final column of Table 1 denotes each set of interactions with another letter that will be appended to each use case step. For instance, the steps relating to the provision of the left hand network provider's (p1) service to either the ASP (a) or the clearinghouse (h) are labelled n1 whether they occur within the domain of p1, a or h.

It is easier to describe this use case by first going straight to the **operational phase**, assuming the configuration has all been set up. Thus, steps 1n-10n, 1c-10c and 1e-10e will be described later. Fig 7 focuses on the operational phase, while Fig 8 shows the

©Copyright 2000, the Members of the M3I Consortium

whole scenario, including configuration and subsequent re-configuration through feedback. There is nothing in Fig 7 that isn't in Fig 8, but Fig 7 is less crowded and so easier to see what is going on initially.

As before, the **operational phase** starts at step 11, the user having accepted an offer at the application level (5e) and initiated the application (6e). In this case, the end-customer's use of the application service (11e) involves use of its own network service ($11n_2$) and bundled use of the ASP's network service ($11n_1$). Use of these three services is noted by four meter systems (the extra one is because the ASP replicates the network provider's metering in this use case).

There may well be application service charges applied to the end-customer. Thus, application metering system, M_{ap}, sends these records to the ASP's provider-side charging system, CA_{ap}. This meter might be monitoring the logs of a video server, or the time spent playing a network game, etc. The notification of any higher level charges for content or application service is not shown (outside the scope of the M3I project).

Measurement of and accounting for network service is as already described in earlier use cases. Each network provider measures its own service (M_{p1} & M_{p2}), passing the results ($12n_1$ & $12n_2$) to its respective charging and accounting system (CA_{p1} & CA_{p2}).

The ASP duplicates the measurements of its network provider (M_{ac} to CA_{ac} via 12c). This duplication is not essential, but is presented in this use case to show how a large end-customer (ASP) will often use its own charging and accounting system to apply fine market control to its network usage. Note, again, the appearance of the wholesale interaction pattern, where the ASP operates two logically separate charging and accounting systems, one to monitor its costs as a customer, and the other to monitor its revenues as a provider. Thus, this pattern isn't just confined to network wholesaling.

The end-customer doesn't do any network charging or accounting for itself as it has nothing to pay - all network charges are being covered by the ASP.

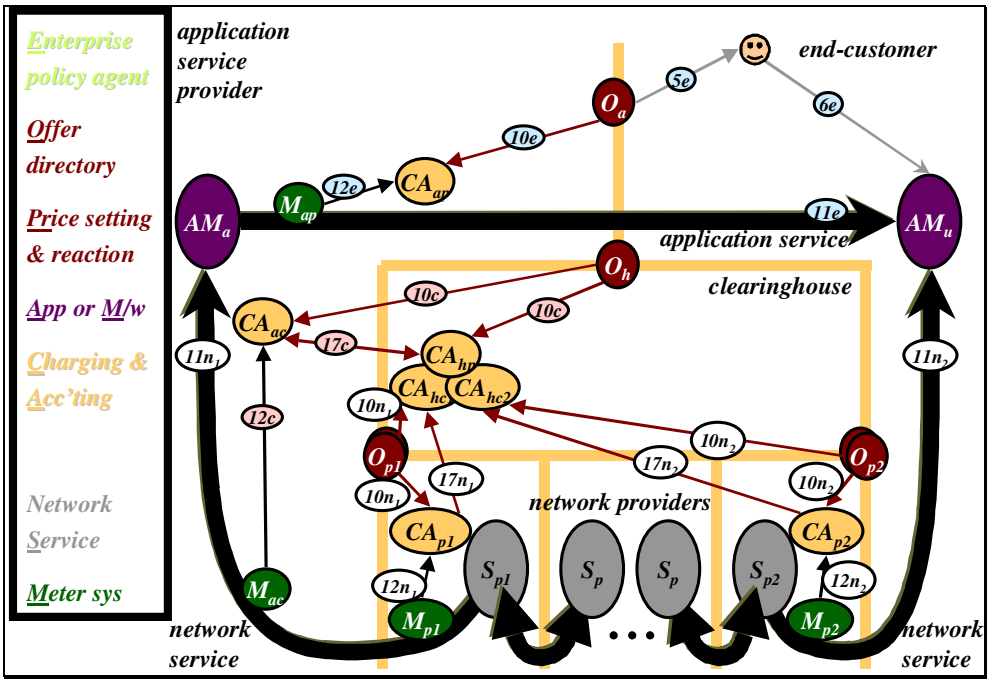


Fig 7 - Clearinghouse use case; operations phase

©Copyright 2000, the Members of the M3I Consortium

Note that the clearinghouse does no metering itself. It relies on receiving accounting records for each session from the two network providers ($17n_1$ & $17n_2$)[9]. It can also correlate these records with those for which the ASP thinks it should be charged (17c). The clearinghouse business relies on taking the risk that collusions between three other parties to falsify all their usage records for what is essentially the same flow through the network will be rare. In particular, the network providers have an incentive to over-report, and the ASP has an incentive to under-report.

The logically separate aspects of the clearinghouse's charging and accounting system are shown in Fig 7. There is one aspect for each provider for which the clearinghouse is a customer (CA_{hc1} & CA_{hc2} facing p1 & p2 respectively). Then there is one aspect for each customer to whom the clearinghouse provides its service (CA_{hp}). Unlike in previous wholesaling scenarios, the provider side is more coupled to the customer side. This is because essentially two parts of the same good (the session) are being re-sold as one, whereas in previous examples, there was the potential for adding value, or at least aggregation or disaggregation into qualitatively different goods.

We now briefly move back to what should have been the start of the use case, to cover the **configuration phase** (Fig 8). As already warned, we gloss over network provider configuration to focus on the clearinghouse and ASP. The clearinghouse functions in the figure are laid out sideways rather than keeping to the convention so far, of having high level policy at the top and low level operations at the bottom. This is because its customers are above, and its providers below.

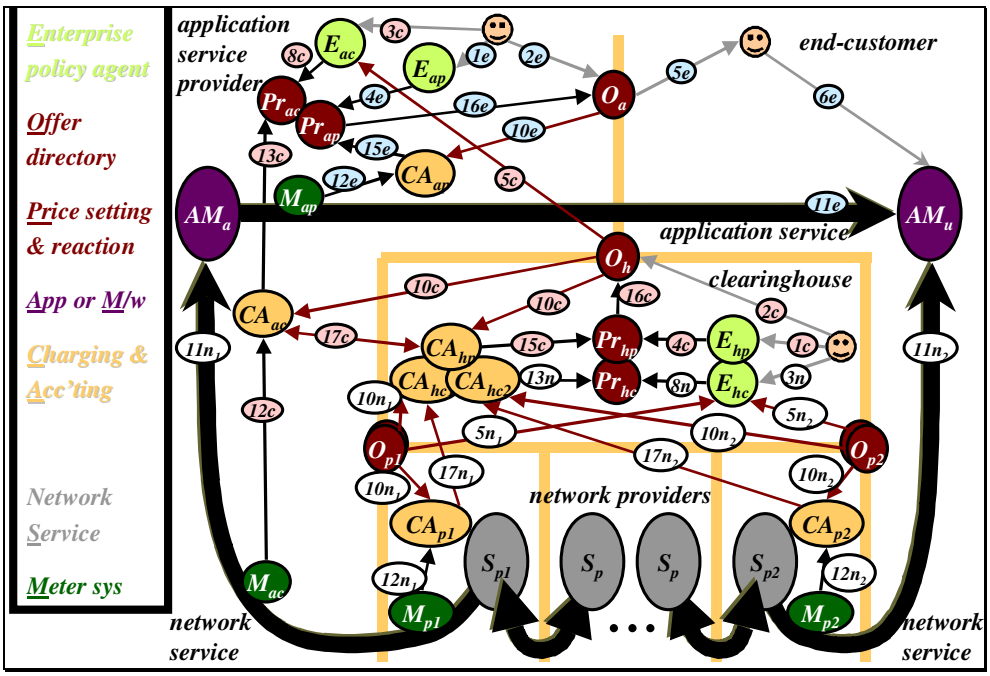


Fig 8 - Clearinghouse use case

The clearinghouse operator sets its provider side enterprise policy (1c) to control the prices it will offer for end-to-end service. At the same time it puts out its offer, O_h, (2c) with initial pricing. It then sets its customer-side enterprise policy (3n) to determine

[9] As before, this assumes the network providers are able to discriminate one session from another, which may not be possible with encrypted traffic (whether using RSVP [40] or off-line traffic metering). In this case, the only option for the clearinghouse would be to get its records from the two end-customers.

©Copyright 2000, the Members of the M3I Consortium

which network providers it is willing to wholesale, and any dependencies on the price they offer, given the prices it is offering itself. Typically, one would expect a clearinghouse to make a profit on everything it re-sells in general. However, it may make occasional deliberate losses when re-selling particularly expensive network service in order to present a uniform set of prices to its own customers. The loss on one provider might be compensated by a higher gain from the other provider combined with it in the same session. This 'win some, lose some' approach is a characteristic of any re-selling activity - it has already appeared in the interconnect and risk broker scenarios.

The clearinghouse's provider enterprise policy agent sets pricing policy (4c), while its customer-side agent listens to all the offers from network providers around the world ($5n_1$, $5n_2$, etc), accepting them if necessary (not shown). It passes those offers that are of interest (8n) to the price reaction function, which is tightly coupled to the price setting function, $Pr_{hc/hp}$, (in fact, it's only recourse if incoming pricing changes is to request a change in outgoing price). The tariffs of all its network providers (O_{p1}, O_{p2}, etc) and its own tariff, O_h, are all monitored by the various charging and accounting systems of the clearinghouse, its customers and its providers, to keep their view of current pricing up to date ($10n_1$, $10n_2$, 10c).

The ASP, similarly, sets its policy as a provider of application services (1e) and publicises its offer (2e). It might be a redundant step to set policy (1e) for setting the pricing (4e) of application services, if they are only offered at fixed prices. In this more likely scenario, pricing would just be set manually, directly, combining steps 1e & 4e. However, an increasing number of e-commerce goods are being offered at highly dynamic prices to compete effectively [7]. It would be necessary to set policy for the buying side of the ASP (3c), which would most probably scan many clearinghouses' offers (5c) to choose the best deal at any one time. A policy on how to react given the chosen deal would then be downloaded into the price reaction function (8c). For instance, an increase in network charges (13c) might trigger the ASP to decline access to one of its services when the network charges reduced the margin it would make on that product below a threshold. This effectively, turns dynamic network pricing into **application-based admission control**, which triggers when the network is too loaded to be able to meet application demand.

There is nothing particularly new compared to previous use cases in the re-configuration phase, where prices are regularly re-calculated based on demand feedback from the respective charging systems (15c & 16c, 15e & 16e). Similarly, reaction of the clearinghouse to the network charges from multiple providers is unchanged (13n). ASP price reaction has already been covered in the previous paragraph. The end-user's price reaction to application offers is assumed to be manual (unshown as internal to the end-user).

2.3.5 Further use cases

Further scenarios have been defined in the latest drafts of the M3I requirements specification. Use cases for these will be included in future issues of this document.

2.4 Applications overview

We have promised to introduce the architecture's building blocks and interfaces and methods to compose them. Use cases like those above always include the outside

©Copyright 2000, the Members of the M3I Consortium

stimuli on a system to give a feel for how it is used. However, we must now draw a line between what is inside and what is outside the market managed multi-service Internet architecture. Essentially there are three activities that drive a market managed network:

- selling
- buying
- using

The applications that support these three activities are defined as outside the architecture. The architecture will not contain any building blocks to implement these functions, but, in order to be useful, it clearly must provide interfaces to support them. That is, management interfaces for buying and selling, and, of course, the service interface for actually using the network. Thus, in the above use cases:

- The **selling** activity (deciding what to sell, deciding tariffs and price setting policies) is primarily **manual**, but the architecture legitimately supports it with directory interfaces for holding offers and provider enterprise policies.
- The **buying** activity involves the **customer enterprise policy agent**. This could be arbitrarily complex, and therefore also sits outside the architecture. It is nonetheless supported with interfaces for offering service to it and receiving offer acceptances from it. A directory interface is provided for it to create, modify and safely store its policies and the price reaction function offers an interface for its policies to be applied to.
- The **customer's applications or middleware** are the entities **using** the network service and hence are also outside the architecture that follows. Obviously, the service interface of the network is provided for their use.

Returning to the opening paragraph of the introduction, we stated our goal as to enable and encourage users of a self-managed Internet to use it 'however they want'. But how do we characterise 'however they want? In other words, "Are there more inputs to this self-managing system than our use cases have identified"?

Clearly the ability to buy bespoke quality of service is part of 'however they want' and we must offer interfaces for this. However there is still room to take a step yet further back. Instead of trying to offer a suitable service for what people believe they should be able to do now, let us consider what is necessary to encourage people to do *anything* with the network.

The key to 'however they want' is commercial openness. That is, to support people who wish to offer their own idea of the network service bundled into their own services. Thus, as well as considering how to service current needs, we must serve the creation of new needs. From this perspective it becomes important to offer management interfaces for two further classes of application:

- selling and buying Internet (not just network) services that use the network service
- selling and buying Internet sessions that use the network service

Thus, our building blocks will offer interfaces to support creation of complementary services and sessions, not just network service itself. We will further justify these choices in a later section where we try to define generic building blocks to represent high level classes of characteristic applications (Section 4.4).

©Copyright 2000, the Members of the M3I Consortium

2.5 Building blocks overview

Here we introduce the *main* building blocks, to put the following discussion on composition in context. All fourteen are fully defined in section 4. We gloss over some completely at this stage.

The reason we have so many building blocks is that this is effectively an architecture of architectures. It has to encompass a number of pre-existing network control, application support *and* charging architectures, all with very different assumptions and compositions. For instance: intserv [19], diffserv [20], sample path shadow pricing (SPSP) [22], TUD/ETH embedded RSVP charging [23] including by auction, BT diffchar [24], etc.

However, the building blocks are not just a pooling of all the unrelated bits of each architecture. They are an identification and distillation of the building blocks that are *common* to all the encompassed architectures. In presenting the building blocks, we deliberately give no indication as to how they should all be put together (because there are multiple ways). However, people familiar with any one of the specific architectures, will immediately recognise the bits of 'their' architecture[10]. Our aim is to enable this, but also allow more generic approaches.

2.5.1 Service operation building blocks overview

We define eight service building blocks that are necessary and sufficient to operate a networking service, including its control and charging operations. The first four deal directly with packets, and have to be embedded in the traffic flow, or at least a copy of part of it. Two of them offer potential interfaces into and back from a charging system.

- **Networking** is the fundamental service of forwarding and routing. Sticking loads of these building blocks end to end makes a big network.
- **Classifying**, **metering** and **QoS control** have always traditionally been used for network control. Classification simply identifies packets with common fields in a traffic stream (e.g. all packets to the same destination with the same diffserv code point). Metering generates **session characterisations** as a side-effect to the packet flow. These are a message payload type used extensively in communication between network control and charging building blocks. We define our terms exactly in Section 5.2.1 but for this overview it is sufficient to say that session characterisation is our generic term for number(s) that represent the rate or volume of load, burstiness, etc. of a session. In a charging context they are typically called **accounting records**, although this isn't a common term when they are used in a network control context. For instance, the QoS control building block uses session characterisations (metered load) as its control input to affect the flow of packets.

  Metering offers a potential interface into a charging system and the QoS controller offers a potential interface back again from a price-control system. This feature is used in Kelly's sample path shadow pricing (SPSP) [22], and some configurations of BT's diffchar [24]. Explicit congestion notification (ECN) marking effectively writes the session characterisation into the packet stream as a marking rate, in order to communicate it along the path.

[10] ...and probably assume they are *only* ever put together in the way familiar to them

©Copyright 2000, the Members of the M3I Consortium

The next five service building blocks are exclusive to the operation of charging systems.

- **Charge advice** simply applies a tariff (see below) to a session characterisation to produce a **charge record**, which gives advice of the charge for that session. It can be used for traditional billing, or it can be used in congestion pricing systems as a measure of the strength of control signal required to be fed back to control the flow rate. It can also be used to assess the cost of various alternative strategies without actually incurring any charges. Together with a tariff, charge advice is the fundamental building block connecting the network world to the charging world.
- **Distribution** ensures management messages are routed to their destination(s), while **aggregation** is used to reduce their volume, either merging details together across sessions (aggregation in space) or buffering details and combining them over the duration of a session (aggregation in time). For messages used in charging systems. Together both determine the workflow of control management messages in both space and time. That is, they control where messages are distributed to and how often, whether they are copied to stable storage, and how they are summarised.
- **Settlement** is simply the process of paying real money (and recording the fact). It determines the **context** of trust necessary for certain of the other building blocks.
- **Correlation** is a 'non-functional' building block not being essential for *correct* functioning of a system. However, any party may use it to ensure *secure* functioning of their system. It is used as a 'double check' on the operation of another party's system, perhaps by random audit, or by detecting unusual patterns of behaviour.

Every one of the above building blocks for operation of the service, except settlement, is associated with a corresponding **policy** information structure, which controls its behaviour. The various stages of configuration, described in overview next, give the opportunity to set these policies. This is how the behaviour of the whole system is controlled.

2.5.2 Configuration building blocks overview

At this overview level, configuration can be taken to mean either configuration or creation. Also, the same principles apply whether the service as a whole is being configured or just the scope of one session of the service. Therefore we use the term **configuration** for service or session configuration or creation.

The general scheme of configuration is to create **associations** between items of information about the service or session (such as the **policies** of the owner) and place them in a **directory**. Looking up any one item of information (e.g. the session title) returns whatever type of associated information is requested, which may in turn be looked up to find further associated information. The directory implementation is left to be defined in a particular design. For instance, it may be either distributed or centralised, and either based on soft or hard state. For instance, Web protocols can be used to implement a distributed directory of extensible mark-up language (XML) pages. The mbone session directory [16] is an example of a soft state directory.

The primary association is the **offer**, which service providers or session initiators use to advertise their wares. An offer associates all sorts of policies together, possibly along

©Copyright 2000, the Members of the M3I Consortium

with related offers that are required to use the service. The offer also typically includes the **tariff** levied for use of the service or session (see Section 4.2.3 for definition). This allows a customer to assess how costly the service or session would be for her particular type of usage. As time passes, the provider may decide to alter the pricing of the offer, based on demand, supply and competition. We define a **price setting** service building block to aid in this task, but it may involve considerable manual input.

Customers have to be able to locate offers that providers are likely to place in appropriate locations by using all the means available for advertising on the Internet. Once located, a customer can create a further association between herself and an offer, by returning an **offer acceptance**, which may include resolution of any number of policies, left available for customisation.

The policies in the offer and in its acceptance, can then all be used to configure the service (i.e. to configure all the operations building blocks) for the particular customer and offer. This includes using any **address allocations** to configure meters, etc. Associations of interest include those between a session's description in an offer, and policies such as its contractual details and the certified roles of those involved.

One further association worthy of particular note allows the QoS control building block(s) to be tailored to the context in which they are used. This context can be as specific as necessary, perhaps relating to the customer, the application and media type, the task being performed and the tariff. **Task reaction policy associations** can be stored in a directory and the policies then applied to a QoS controller whenever the associated context applies.

2.6 Interfaces overview

In our overview so far, we have described the *function* of each service building block, while only briefly highlighting some of its inputs and outputs. The full definitions of each service building block (Section 4) include an enumeration of all required inputs and expected outputs - that is the **interfaces**. Thus, a detailed list of all the interfaces required between the building blocks is best left until Section 5 after the building blocks are defined.

Interface definitions are one of the most useful parts of an architecture, therefore it is important to get the major separations of building blocks in the right places first, as this determines which interfaces are required. Then it is important to classify the interfaces to avoid specifying two interfaces as different where they may in fact be sufficiently similar for an abstraction to be created so that both can be defined as the same type of interface. To this end, interfaces may be classified by:

- **distribution** - whether an internal interface or an interface between distributed systems
- **interaction mode** - whether query-response, request-confirm, event listener, one-way etc.
- **message payload type(s)** and **parameter types** or the **type signature**
- **service primitives** or **methods**

A building block with an output of a certain type can legally interface with another building block with an input of the same type, as long as it is the same in all four of the above respects, which we now explain:

©Copyright 2000, the Members of the M3I Consortium

2.6.1 Interface distribution

In any particular scenario, certain interfaces will be between two[11] building blocks on the same machine operated by the same party. Other interfaces will cross machine and organisational boundaries. In both cases, an **application programming interface (API)** is required to each service building block. In the latter case, there will also be a protocol (most likely a number of layered protocols) between the two distributed systems and organisations[12]. There are two common approaches to defining these protocols:

- To define *all* the layers of **wire protocols** to be used at each end of the communication for the interface in question. In this case, the APIs to the protocol handler at each end *need* not be the same, as the protocol provides the common interface.
- To specify use of a **distributed processing environment (DPE)**, which provides an abstraction of the distributed nature of the interface, such that the API may be used at either end of the interface without concern for the wire protocol. At each end of a communication, a DPE creates a proxy of the other end that offers the defined API, such that interaction with the API appears local (in functional terms, if not in performance). The same API is therefore presented at both ends. A DPE uses its own generic wire protocol for all communications, the nature of which is hidden from the programmer. However, certain advanced DPEs offer 'selective transparency' of the communications mechanism, which leaves it hidden if it is not of concern, but allows the application programmer to tune its engineering performance if required.

Interfacing over a DPE is appropriate where the pace of change puts programming flexibility above the details of communications engineering, such as performance. On the other hand, the process of defining a standard wire protocol brings out all the engineering details of the communication and therefore tends to be used where performance is more important than flexibility.

However, the DPE approach is usually only practical between distributed systems under the control of a single organisational unit (or a tightly controlled group of organisations). Where one system has multiple interfaces to multiple other distributed systems in multiple organisational units (whether in the same organisation or different), it is necessary to agree on a common DPE for all systems. This is to avoid the system in the middle having to cope with multiple different DPE technologies (including licensing them). For globally interconnected systems, it can soon be seen that this task becomes impossible. Standardisation of the main elements of DPEs has been steadily improving over the years (starting with the wire protocol and gradually including the supporting services). However, gateways are often required between different DPE approaches and considerable problems remain.

For this reason, it is still fruitful to agree specific global standard wire protocols for specific interfaces among the parties specialising in that interface. In a globally interconnected set of systems, each system can be implemented with each agreed protocol on each interface. Of course, in practice more than one standard protocol

[11] An interface may well have many ends, not just two. If relevant, the reader is asked to assume 'many' wherever 'two' is stated.

[12] It is possible *but not common* to implement a bespoke protocol directly within the implementation of a service, in which case there is no API.

©Copyright 2000, the Members of the M3I Consortium

often ends up being agreed by different factions. Where more than one standard is available, either most organisations within a faction will only interface with others in the same faction, or some will implement both protocol standards to act as gateways between factions. Note that multi-ended protocols clearly have a stronger need for a single global standard, for similar reasons of co-ordination between multiple organisations.

For any one scenario, this architecture therefore recommends the use or creation of wire protocol standards for interfaces likely to cross organisational boundaries. Interfaces between distributed systems within an organisation may use a DPE unless the data volume makes detailed protocol engineering necessary for performance. General recommendations on technology agreed for use in the project, including the DPE, are in [33].

However, certain interfaces in the architecture are not central to the research interest of the M3I project. Therefore, even if definition of a wire protocol would be appropriate, if one is not readily available (either code is not in the public domain or it is more expensive than is warranted for the project), it will be appropriate to use a DPE under the APIs. Thus, for instance, internal interfaces within a charging and accounting system may be DPE-based.

In all cases, API definitions are necessary, whether between a building block and its protocol handler or between building blocks (whether or not a DPE is interposed transparently).

2.6.2 Interface interaction mode

There are three clear patterns in the modes of interaction across most of the interfaces in this architecture.

- The interfaces that support *configuration* invariably operate in the style where an issued configuration **request** is accepted and **confirmed**[13] (or failure reported).
  - An alternative to this has to be found where a single major event causes a need to configure large numbers of interfaces (e.g. a price change or context change affects many parts of the provider's systems, and every customer's system). Configuration interfaces that use the **event listener** style of interaction are more appropriate here, and reliability often has to be achieved by alternative means than acknowledgement, which would overload the source of the configuration event with an implosion of responses.
- On the other hand, the interfaces between the *operational* building blocks in M3I invariably involve a continuous flow of messages in one direction. There is no explicit request for each message, other than configuration of the system to request that future messages should be passed to a certain service building block. This interaction mode doesn't preclude acknowledgement for reliability, but this is usually achieved at a lower layer protocol (e.g. TCP). Thus at the application layer, such messages don't need a response and are called **one-way** messages. The overall pattern of initial configuration then one-way messaging is called the **event listener mode**. As discussed above, this encompasses sending to multiple destinations if

[13] Both query-response and request-confirm are client-server style interactions. Formally there is no theoretical distinction between them, the two styles simply indicate whether the significant payload is in the response to a query or in a request that typically configures a service and merely needs confirming.

©Copyright 2000, the Members of the M3I Consortium

required. These flows form the control loops that manage the main flow - that of Internet service - through the network.

- Only two M3I operational interfaces are exceptions to this pattern. They can be considered normal, but not regular, operation as they involve arbitrary, **query-response**[13] mode interactions. One supports arbitrary queries from humans on the current state of charges (e.g. to check bill status) or usage (e.g. to support a marketing campaign or to diagnose a fault). The other allows a human or agent to request hypothetical quotations to allow comparison of competing offers.

2.6.3 Message payload type(s) or type signature

In M3I, most interfaces are primarily concerned with moving a fair amount of management data around control loops. Services regularly output messages containing a payload that the client is 'listening' for. Thus the interfaces of most interest in M3I are primarily characterised by their **message payload types**. That is, whether the message carries an accounting record, a QoS control policy or a tariff etc. determines which protocol is chosen or defined to carry it. At this high level, it is also necessary to agree on the format of the payload itself, to ensure the information it carries can be extracted and used at the other end.

Where APIs are concerned, it is more usual to refer to the payload type, parameter types and exception types as the **type signature** of the interface. This, with the method and interaction mode, forms the definition of the API.

Of course, even a specialised management protocol is often designed to handle *multiple* types of payload. Therefore, in Section 5 we start by identifying families of payload types (abstract types). Then, more specific payload types are identified (sub-types). The aim is to try to design an efficient protocol for the whole family of payload types, but where efficiency is incompatible with generality, the option of designing a more specific protocol is left open.

Where interfaces are grouped into families, any pair of similar type interfaces may be of different sub-types and therefore not immediately compatible. However, an output interface that is incompatible with another input, but of the same family, may be able to intercommunicate through some intermediary, e.g. through a format translation gateway (**transformation** is therefore provided as a service building block). This motivates grouping interfaces into families.

Well-designed, low-level generic protocols can accept a wide range of message payload types. However, because most of the M3I interfaces are at a higher management (application) level (layered over one of these generic protocols) they are more concerned with the specifics of the management data transferred, such as its format. There is also little emphasis on the client-server style of interaction, where the client wants the server to do something or to supply some result. Thus the **service primitives (methods)** employed or the **other protocol parameters** are not the main concern, both usually being fairly trivial.

©Copyright 2000, the Members of the M3I Consortium

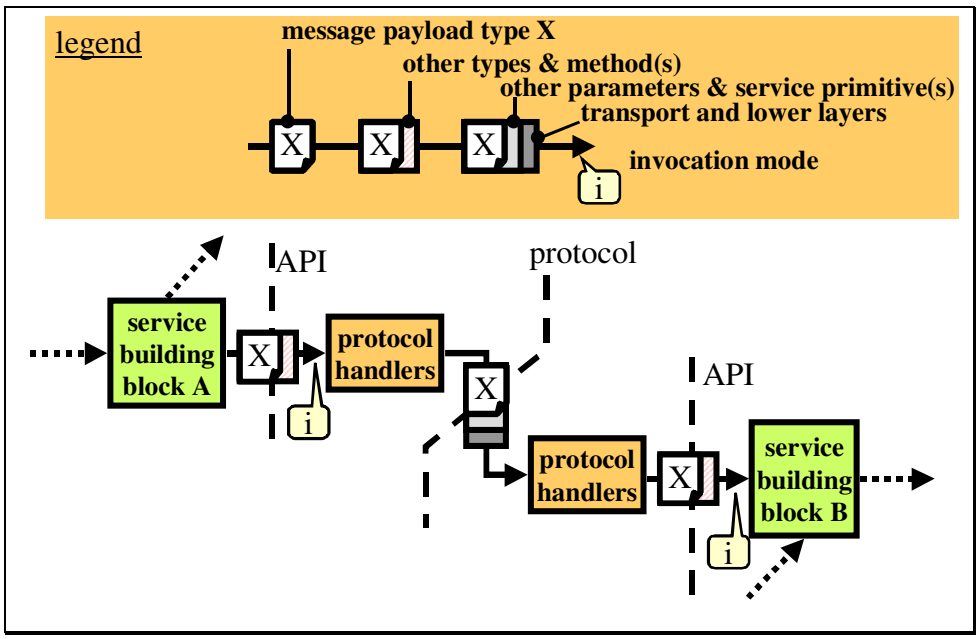


Fig 9 - Interface approach

To summarise, Fig 9 shows the approach to identifying interfaces diagrammatically. We focus on a distributed interface, rather than an internal one - that is, an interface that is distributed, in at least one use case. First the inputs and outputs for two building blocks A and B are defined in terms of the main high level aspects: the predominant message payload type, X, and the interaction mode, i. Where X and i match, it is likely that an interface can be defined between the two building blocks. However, the other details of the interface must, of course, match as well. In the context of M3I, we know from experience that aspects like the method and other supporting parameters are usually fairly trivial making this approach reasonable.

To define the wire protocol between the building blocks, first the format of the message payload will need to be defined. Then an application layer protocol is chosen or created that is suitable for the type of message payload. Beneath this, the choice of transport protocol will depend to a great extent on the interaction mode. A set of protocol handlers is then defined with the API as its upper interface and down through the relevant lower layer protocols to the network at its lower interface. Alternatively, if the wire protocol is of little concern, a DPE may be used.

Example:

Let us consider step 13 in the use case shown in Fig 2 (from the charging and accounting system to price reaction). We will show later that this interface is in fact between the charge advice building block within the charging and accounting system and the price reaction building block. The charge advice building block has an output interface with a message payload type of 'charge record' (denoted later by $) (X=$). It's interaction mode given in Section 4.1.5 is request-reply (i=r/r). From Section 4.1.6 price reaction has an input interface which matches - with X=$ and i=r/r. In Fig 3, this interface is shown as distributed and crosses an organisational boundary, therefore we need to define a wire protocol for it. It will be necessary to define or choose a message payload format that represents a charge record. The application protocol used might be the ETSI Tiphon open settlement protocol (OSP) or we may find this is inadequate and have to improve on it. The interaction mode of both ends is defined as request-reply.

©Copyright 2000, the Members of the M3I Consortium

Therefore the transport protocol is likely to be TCP. OSP is defined to run over TCP, so if OSP is adequate, this process fully defines the interface.

2.7 Interim summary of parts

So far this overview has built up a list of parts (characteristic applications, service building blocks and interfaces between them), from which to build the architecture for any scenario. Before discussing how to describe compositions of these parts, we pause for a brief interim summary.

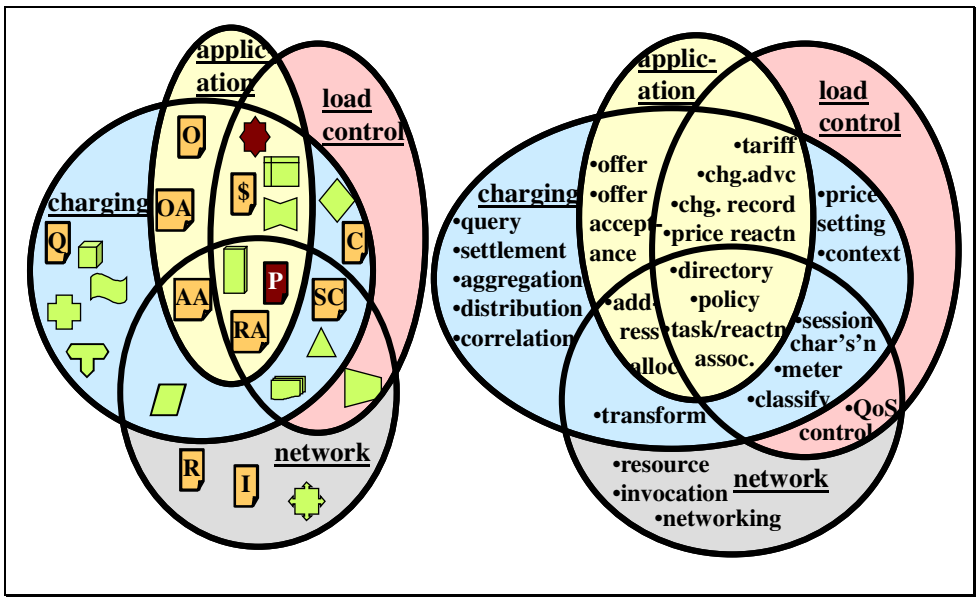


Fig 10 - Classification of main building blocks and message payload types

Fig 10 depicts the main building blocks and message payload types, classifying them by which of the four main aspects of a M3I they relate to. Their names are given on the right and the symbols used later in this document on the left. Note that nearly all the configuration building blocks are deliberately in the application domain, to avoid embedding any configuration assumptions into the infrastructure.

Over two thirds of the items fall into more than one category. This is because each item's role is ambiguous until it is included in a specific composition of the architecture for a scenario. We introduce the subject of composition next.

2.8 Compositions overview

The aim of the following technique for defining a composition is to be crystal clear about what is being described, but at the same time, to remain succinct.

Below we describe the technique using a diffserv service level agreement as a partial example scenario running through the whole of the rest of this section.

The purpose of using a common technique for all parts of the system - network, control, charging and application, is to allow the merits of proposals with their complexity in radically different parts of the system to be compared. For instance, diffserv appears simple until you come to the difficult bit: working out how to configure each core router to allow for each new policy on each border router. Our composition technique allows the essence of an architecture to be described, but it can also be extended to describe

©Copyright 2000, the Members of the M3I Consortium

the whole process of running the architecture as a business. Incidentally, we only use the easy bit of diffserv in our example!

2.8.1 Service components

As a first step, it is usually necessary to define a few re-usable collections of building blocks that we shall call **service components**. These typically represent the products the provider invests in to be able to offer the service, having chosen a particular architecture. Later we give some examples of such service components in the scenarios of interest.

However, the break-down into rudimentary building blocks is still justified, as communication between the service components is still essentially between the building blocks within them, therefore the same interfaces and protocols are still relevant. Of course, this also helps greatly when considering interoperation between architectures.

In our diffserv example, we build a **diffserv policer** from one classifier, and as many meters and QoS controllers as there are classes in the classifier policy, P_c. (Fig 11a - the symbols used are introduced in Section 4 and briefly in Sections 2.5 & 2.6 using Fig 10). We also build a **diffserv classifier** from a classifier and as many queues as there are classifications, each of which is assigned a share of available buffer and scheduling resource, R (Fig 11b). Finally we connect a diffserv policer component to each edge interface and a diffserv classifier to all the other interfaces to make a **diffserv edge router** component (not shown). Similarly, we connect a diffserv classifier component to every interface of a router to make a **diffserv core router** component (again not shown). Finally we connect a diffserv edge router component to a number of diffserv core router components to build a **diffserv network** component (Fig 11c - note for simplicity, we have only shown traffic flowing from one edge interface).

Other typical examples of networking components would be an intserv-enabled router or an intserv-enabled network. A typical charging system example would be a mediation system (built from distributors, transformers and aggregators). See Section 6 for more examples.

©Copyright 2000, the Members of the M3I Consortium

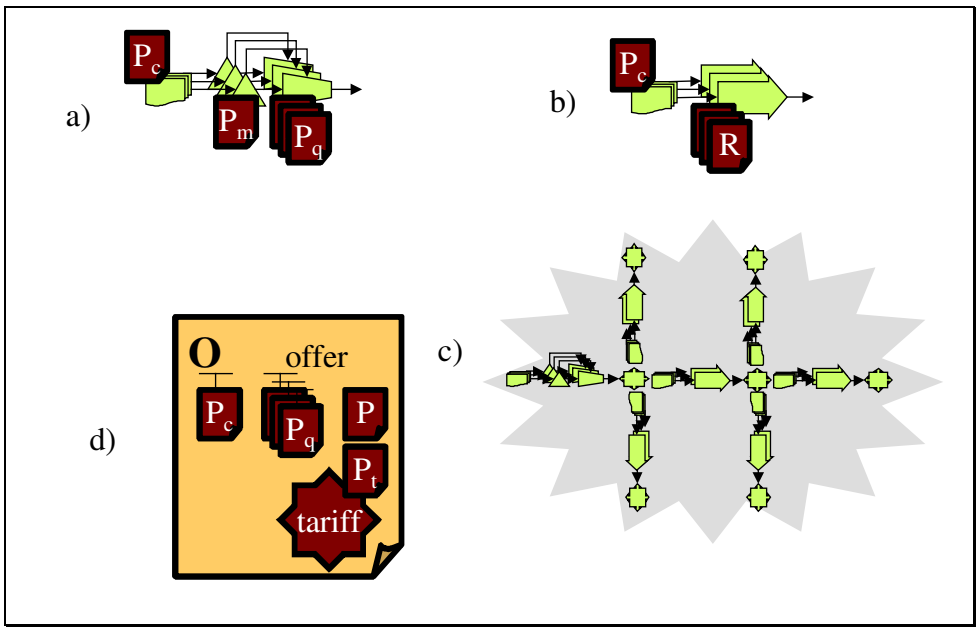


Fig 11 - a) diffserv policer; b) diffserv classifier; c) diffserv network and c) diffserv offer

Later, we will define a **service offer** more precisely, but the earlier overview of offers and a brief description of the diffserv offer shown in (Fig 11d) should suffice for this introductory note. From top left to right, our example **diffserv offer** includes:

- The **classifier policy, P_c** to be completed by the customer. The mappings that the customer wants between application protocols and diffserv code-points.
- The **QoS control policy, P_q**, for each code-point, also to be completed by the customer. That is, how much traffic the customer plans to send for each code-point, or a more detailed specification of its profile (e.g. for different times of day).
- The **service definition, P**. That is, the service level agreement that specialises the fundamental definitions of what service this particular customer can expect. Note this has no interface - it is merely a statement by the provider in this scenario.
- The **pricing, P_t**, for the tariff. This allows the customer to compromise what she asks for against what it will cost her. Again, there is no interface to change this in this scenario.

Having defined the main building blocks and service components in our scenario, we can now move on to define their composition. This, of course, requires an **architecture diagram** (not shown for this scenario) that shows the interfaces between the building blocks in each service component. However, this is insufficient to fully define the architecture. It is also necessary to specify the **granularity** of composition: by **deployment granularity**, by **role granularity** and by **event granularity**.

2.8.2 By deployment granularity

Deployment granularity *can* be shown with a diagram, although it is often difficult to highlight the granularity of entities of state with respect to logical contexts (e.g. that there is one QoS policy per application protocol per router interface). In most respects a table is less ambiguous.

Table 2 lists (some of) the service components of a diffserv architecture down the left hand side, and typical places and contexts into which the service components are

©Copyright 2000, the Members of the M3I Consortium

deployed along the top axis. The body of the table gives an indication of the deployment complexity of the different service components. For instance, the blob in the policer row means one policer is deployed per edge router interface, with for instance none on the interfaces of the same router that connect internally.

per: / service component	flow	task context	aggre-gation	user	applic-ation protocol	host	manage-ment system	router		domain
								edge inter-face	non-edge inter-face	
diffserv policer								●		
diffserv classifier									●	
diffserv network										●
P_c								●	●	
P_q					●			●		
R					●				●	
...										

Table 2 - Composition granularity by deployment

The columns include examples of logical contexts to the left and physical places to the right (divided by a thicker line).

Note that QoS control polices and resource specifications are both per application protocol and per router interface. In other words, there is a policy for each combination of protocol and interface.

The headings in the table are illustrative only. Headings should be chosen to give the detail required for the composition in hand. Clearly if two compositions are being compared, it helps to have the same headings for each.

2.8.3 By role granularity

Table 3 is similar to the left hand half of Table 2, in that it is classified by contexts, rather than places. However, the contexts we choose to list here are stakeholder roles along the top axis. The table then shows *who* is expected to operate each function or configure each information structure. Note that many of the roles have two aspects, one as a customer of the network service, and another as a supplier of the service heading the column. These are labelled 'C' and 'P' respectively.

A table like this is useful for analysing missing security requirements. It also allows redundancies to be spotted, for instance:

- many functions are repeated at both sides of a customer-provider interface, eg. in some configurations of diffserv, the customer shapes and the provider polices, leading to the possibility of the provider doing sample policing for instance;
- looking wider than diffserv, if price-based QoS control involves calculating a usage-charge, this doesn't need to be done separately from accounting in some configurations;

©Copyright 2000, the Members of the M3I Consortium

- price-based QoS control includes self-admission control so it could make a separate admission control system redundant - one of the project's research goals to establish whether this is so.

per:	End-cust-omer	End-user network provider		Access provider		Back-bone provider		Value-added service provider		App /content provider		Comms service provider		Charging function provider	
service component	C	P	C	P	C	P	C	P	C	P	C	P	C	P	C
diffserv edge router				●		●									
diffserv core router				●		●									
diffserv shaper			●		●		●				●		●		
P_c	(W)		W	R	W	R	W				W		W		
P_q	(W)		W	R	W	R	W				W		W		
R				W		W									
Svc def'n policy	(R)		R	W	R	W	R				R		R		
...															

Table 3 - Composition granularity by role
W = writable; (W) = writable but unlikely; R = readable; (R) = readable but unlikely

Note that for rows showing state in a component we can make the granularity table richer by identifying where state must be writable, and where it merely needs to be readable. This helps show the complexity of updating any item of information. The 'unlikely' status of the entries under the end-customer column is because diffserv is not really designed for this market. Certainly an end-customer wouldn't be expected to own a shaper, even if they were offered diffserv.

It is also often useful to represent much of the role granularity information diagrammatically, as long as more abstract roles are used, such as 'large customer' or 'network provider'. This approach is similar to that used in the earlier use cases.

2.8.4 By event granularity

Table 4 shows composition in the time dimension. It shows the possible different timescales over which different service components exist. Diffserv is a fairly uninteresting example in this respect, as everything other than load is stable from when a service offer is accepted to when it is terminated. If we had chosen a usage-based volume charging example, we would have been able to show a session characterisation being created for each new session detected - a characteristic of flow detail metering.

per: action	packet	reser-vation	sess-ion	query	addr. alloc'n	daily batch	price-chnge	billing period	offer accept-ance	svc life-time
set P_c									●	

©Copyright 2000, the Members of the M3I Consortium

per: action	packet	reser-vation	sess-ion	query	addr. alloc'n	daily batch	price-chnge	billing period	offer accept-ance	svc life-time
access P_c	●									
set P_q									●	
access P_q	●									
set R						●				
access R	●									
...										

Table 4 - Composition granularity by event

3 Definitions

Having introduced the whole approach, we now start the more comprehensive body of this document by defining our terms more precisely.

We use the term **service building block** for a minimum unit of function necessary for the architecture. Service building blocks may consist of more than one service element, where we have found that certain elements are always used together in the same configuration. Where certain building blocks are often but not always found configured together in the same way, we define such a super-minimal unit of function as a **service component** (we use the term without implying its more specific meaning from software engineering, i.e. it doesn't necessarily have a self-describing interface).

Fig 12 and Fig 13 collect together the legends for the rest of the figures in this architecture. The three symbols along the top of Fig 12 are the legend for this legend! They represent state in the system. From left to right, they are:

- **state** (represented by the sheet of paper with folded corner) - variables associated with a service building block (represented by the oval)
- **message** - that is state in transit such as a parameter or the payload of a message
- state with an **interface** to set or access it[14].

Any symbol for state (e.g. O for an offer) can also represent the same information when it is in transit, simply by showing it over an arrow (a message containing an offer in our example). All messages are shown in just one direction - that of movement of the state depicted. The interaction mode of a message is described in the text, so trivial request or confirmation messages[15] are not shown for clarity.

The meaning of each symbol for information (or state) in the lower half of Fig 12 will be introduced as we proceed.

[14] Strictly this is just one of the interfaces of the service building block, but it is shown attached to the state to emphasise its direct relationship.

[15] Only messages at the layer depicted are defined by the interaction mode. For instance, if a message is one-way at the application layer, any transport layer acknowledgement is not part of its interaction mode.

©Copyright 2000, the Members of the M3I Consortium

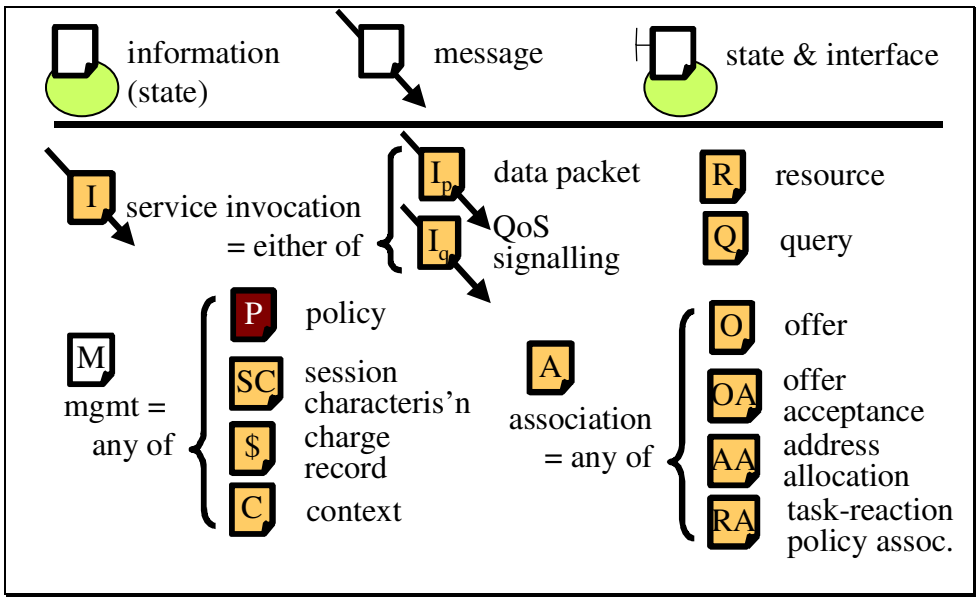


Fig 12 - Legend: state

A dark shaded symbol for information indicates it is potentially 'active'. In other words, it potentially contains logic and behaviour (e.g. script commands) rather than just static information as the lightly shaded symbols do. The white background to the symbol for management information simply indicates it is an abstraction for any of a number of types of state, some of which are static and others active.

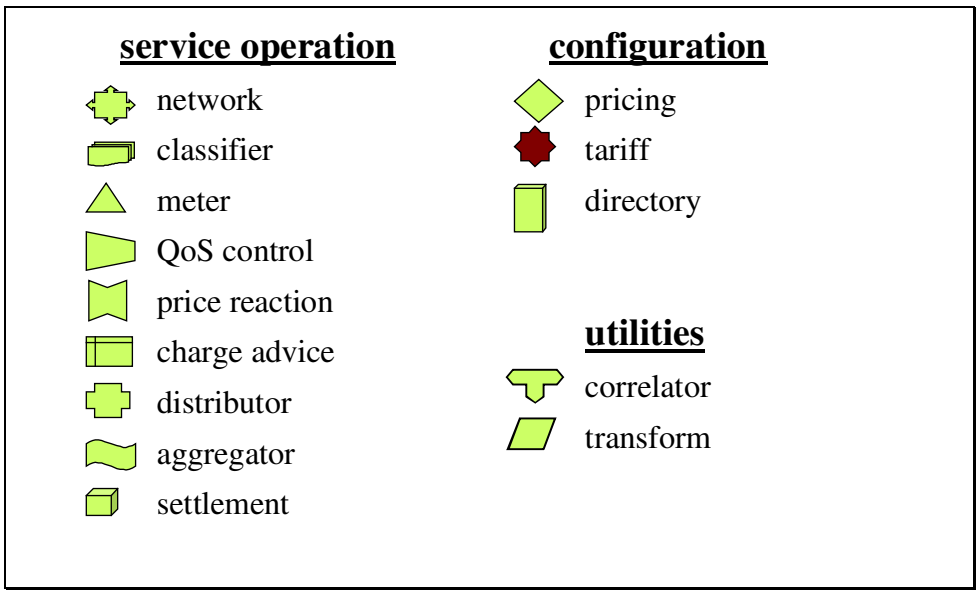


Fig 13 - Legend: service building blocks

Fig 13 is the legend of all the symbols for service building blocks (all the shapes that aren't like little documents), which will also be introduced as we proceed.

The service building blocks introduced in the next section are divided into those for **service operation**, then those for **configuration** (followed by a couple of utility building blocks that support operation). These are defined as follows:

©Copyright 2000, the Members of the M3I Consortium

- **Service operation** is defined as the flow of state between functions and general internal operation of those functions while the configuration state of those functions remains static.
- **Service or session configuration** is defined as the application of configuration state to functions in order to alter their future behaviour.

4 Building blocks

4.1 Network service operation building blocks

4.1.1 Networking

Perhaps surprisingly, the **networking** building block (Fig 14) fully describes every important aspect of the operation of Internet service. It consists of two service elements: **forwarding** and **routing**. Forwarding exists within the context of some **resource, R,** namely bandwidth, buffer space, and scheduling priority. The network service is invoked by sending **invocation messages, I,** to it. In more familiar terms, the invocations of immediate concern are just packets! But later we will broaden the context to invocations of a signalling protocol such as reservation protocol (RSVP) [3], so in general we will stick to the term invocation.

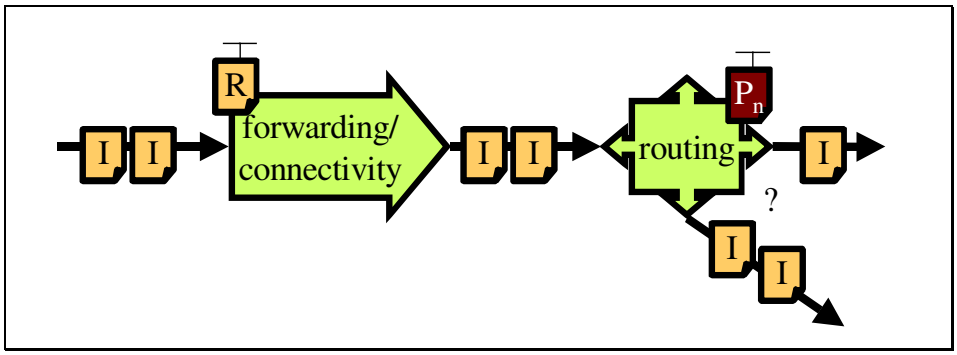


Fig 14 - Networking

The routing function simply decides into which output to forward each invocation based on routing **state, P_n**, residing in the router (multicast routing potentially forwards to multiple outputs). If connected to another networking service building block, this building block will invoke the onward forwarding service, then its routing service, and so on. Note that we model *all* routing state as 'policy'. This appears confusing as the term 'routing policy' is normally reserved for the information used to modify border routing to fit the commercial goals of each network provider. However, in effect, such border policy permeates every routing table on the Internet, by propagation.

We now dwell briefly on the concept of a packet as an invocation of a service. A packet consists of header and payload, where each field in the header can be used as a parameter to a function call. Thus a packet arriving at the routing function invokes the method route(dest_addr). That is, it invokes the route function on the destination address parameter in the packet. All the other headers are ignored, but copied through along with the payload to the next potential invocation, in case they are needed. Incidentally, the forwarding function requires no parameters: forward() just results in all

©Copyright 2000, the Members of the M3I Consortium

the parameters and payload being forwarded to the other end of the pipe at a rate determined by R.

Note that the packet triggers whatever function it encounters. It does not contain the rules for what is invoked or how it is invoked. For instance, the routing function could be altered to invoke route(dest_addr, DS_code_pt) to enable some form of QoS routing [4] that also takes account of the diffserv code point field [2] in each packet. The function determines the nature of its own invocation. Thus, by connecting together different building blocks, we alter the logical order of invocation of a sequence of pre-set functions.

The resource, R, available for forwarding doesn't necessarily represent the total resource available on a whole link into a router. It may be some partition of a link's resource. The building block described next classifies invocations into these partitions of resource. Thus, any one physical link into a router may be modelled by many of these networking building blocks in parallel. Strictly, each element of the building block has resources available for forwarding, however, it is sufficient to only model the bottleneck resource of the whole building block. Later, other building blocks will be introduced, some of which when put together in certain ways allow the resource to be set to match to the required rate of invocations, while others allow the rate of invocations to match the resource available.

Note that this building block models one direction of the (connectionless) service, but of course, a similar building block can be used in the other direction. Of course, there would be a forwarding building block on each input interface of the routing building block, not just one as shown. Also note that a pre-requisite for forwarding is connectivity. Although connectivity has value even if not used for forwarding, we bundle the two together for brevity.

4.1.2 Classifying

A **classifier** building block (Fig 15) separates a series of invocations into sets based on rules in the **policy, P_c**.

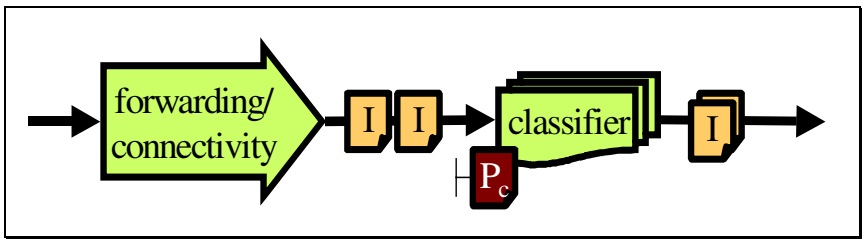


Fig 15 - Classifying

A typical classification policy (rule) might be to separate out all invocations that have common source and destination addresses and ports (flows). However, any classification policy that is possible is valid, in general termed a multi-field classification. Common classifications would be aggregations of flows between one set of addresses and another or a flow might be further sub-divided into sets with common diffserv code points.

The output from a classifier isn't necessarily a physical separation of the invocations into parallel queues, one per classification, as the figure might be taken to imply. The queues might be virtual - that is, the process implementing the classifier would merely

©Copyright 2000, the Members of the M3I Consortium

maintain a handle on which packet was in which classification, but the invocations might still appear in the same order as they arrived.

A classifier operates on a connectionless flow of invocations in one direction only.

4.1.3 Metering

A **meter** building block (Fig 16) is strictly passive. It doesn't change the invocations, or their order. It merely characterises the invocation load over time. The nature of the characterisation is dependent on **policy, P_m**. For instance it might sum the packet sizes, or report the mean or peak invocation rate. The essence of a meter is that it *sums* or *integrates* a characteristic of a number of invocations over time, introducing memory of the recent history of invocations. The history may be over a discrete time or a continuously moving average.

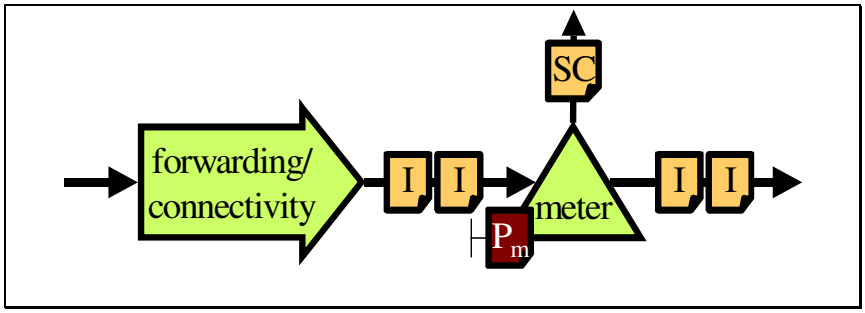


Fig 16 - Metering

The definition of a sequence of invocations for a certain time is a session (strictly it is still a session if the times are unknown, but still knowable, even if only in retrospect). Thus we term the output from a meter as a **session characterisation, SC**. In the field of charging, a more familiar term is an **accounting record**, but this isn't appropriate when this message is used for QoS control, so a more general term has been coined. However, we may use the term accounting record where appropriate. A meter may regularly report a sequence of intermediate SCs, which when accumulated sum to a characterisation of the whole collection of concatenated sessions. The term session is very flexible, which is why we have deliberately chosen it. A concatenation of sessions is itself a session. An aggregation of parallel sessions can also be defined as another session. Sessions can consist of other sessions to any number of levels of recursion. Note that the flow of invocations is not a session, as it is not bounded in time, which is a necessary property of a session.

A meter operates on a connectionless flow of invocations in one direction only. However, its characterisation output need not be connectionless. The question doesn't arise if the next building block is in the same address space (e.g. in a traffic shaper), but if the output is sent over a network (e.g. to a meter reader [5]) it might make sense to ensure message reliability, depending on the impact of loss, and the vulnerability to denial of service attacks.

4.1.4 QoS, rate, admission or access control

The **QoS control** building block (Fig 17a) takes a series of service invocations as input and controls the level of output dependent on an incoming session characterisation signal. Typically, this signal will have been derived from the flow of invocations being controlled, allowing the recent history of invocations to control their own current QoS.

©Copyright 2000, the Members of the M3I Consortium

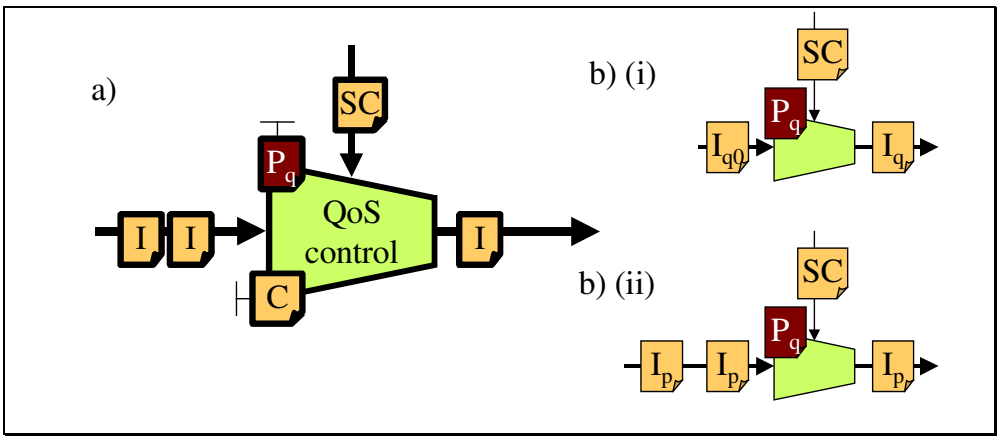


Fig 17 - QoS, rate, admission or access control

The nature of the control is open to wide interpretation. It may drop random invocations in order to maintain the desired rate [6] (admission control). It may delay (buffer) invocations until a possible later lull, thus smoothing the load. We even allow the building block to encompass behaviours where all inputs survive into the output, but certain output invocations are marked or designated to be placed in a lower priority virtual queue. Thus, we allow this building block to merely control the rate of the invocations it allows through unchanged. The remainder may be dropped, marked, or somehow treated in a less favourable way. Whatever the control algorithm, the strength of the effect of the session characterisation signal on the invocations is determined by **policy, P_q**.

Fig 17b shows how the QoS control building block is polymorphic. That is, its function depends on the type of its input. If the input invocations are themselves QoS signals (I_{q0} in Fig 17b(i)), the QoS controller outputs a modified QoS signal, I_q, that matches its QoS control policy, P_q. The motivation for this is described in [31], relating to where an application has delegated control of QoS to this controller, which can take account of the charges involved. On the other hand, if the inputs are packets, I_p, the *shape* of the output stream of packets conforms to the QoS control policy, P_q as in Fig 17b(ii). The QoS control policy, P_q, that controls this building block, may be a static QoS specification (e.g. and RSVP flowspec), or a dynamic specification to control the parameters of an adaptive algorithm. The exact nature of this building block is to be investigated in the price reaction task of the project [31].

We have taken further liberties with this building block, by generalising it to model an *access* controller. Our justification is that access denial is equivalent to zero rate output. That is, the policy, P_q, can be set to cut access altogether (or at least for certain types of invocations) dependent on **context, C**. For instance, the policy may be to cut access to all but best effort traffic if the customer account context runs into debt. Note, however, that some system compositions will place this building block under customer control, where it would be unlikely to be used for access control.

Whether under provider, or customer control, this controller is effectively a policy enforcement point (PEP), with decisions affecting it being communicated via its policy and context interfaces from policy decision points (PDPs), as in the policy-based admission control framework [6].

A QoS controller operates on a connectionless flow of invocations in one direction only and the control input is also likely to be unreliable, connectionless and unidirectional.

©Copyright 2000, the Members of the M3I Consortium

4.1.5 Charge advice

The **charge advice** building block (Fig 18) turns the characterisation of a session into advice of how much money would be charged, given a certain **tariff** with certain pricing **policies, P_t**. The result also depends on the **context, C**.

The interfaces on the tariff and its pricing policies relate to configuration, not operation, so will be described when we discuss configuring a tariff (see Section 4.2.3).

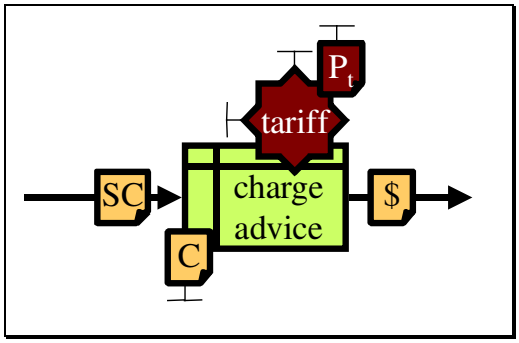


Fig 18 - Charge advice

The context, C, represents such things as who the customer is, what their current account balance is, what their current volume of business is with the provider, which other service offers the customer has accepted etc. Thus SC represents the facts about the product or service being charged for (including its own context such as the time of day), while C represents the facts about the parties involved.

Note that the charge advice need not alter the amount owing. Whether it does, purely depends on into which function it is input later. The charge advice function may merely be used to see how much various sessions would cost in various circumstances (effectively to create quotations). There is no implication on who is allowed to do this. Given the tariff, the customer can operate charge advice completely autonomously. For instance, if the customer is a corporate entity entitled to a volume discount, one user can calculate the charge for their current session in two contexts - one assuming their company will reach the volume required and the other that they won't. As long as the corporate as a whole can give its estimate of how likely each context is, each user can then weight the scenarios accordingly to estimate the likely charge for each of their sessions.

The building blocks discussed up to this point receive a regular invocation load in one direction, purely as a result of network activity. Charge advice also receives such a load, but it is the first building block we have introduced that expects to also be invoked arbitrarily. The invocation load is thus beyond the control of the network's inherent rate control mechanisms. In both cases however, it is likely to be used on a request-reply basis.

The outputted **charge record, $**, (sometimes itself called a charge advice) may include a reference to (or a copy of) the original session characterisation, SC, the context, C and the tariff. Note that the use of the $ symbol doesn't imply that money is output by this building block! $ merely contains a *description* of an amount of money.

©Copyright 2000, the Members of the M3I Consortium

4.1.6 Price/charge reaction

The **price reaction** service building block outputs the long-term control for the QoS control building blocks - the **QoS control policy, P_q**. It acts on behalf of the customer through the customer's **price reaction policy, P_r**. It responds to the current charge for a session, $, by signalling the QoS that will optimise the charge for future activity. It is guided in this, by its price reaction policy, which is discussed in Section 4.2.11 and in more depth in [31].

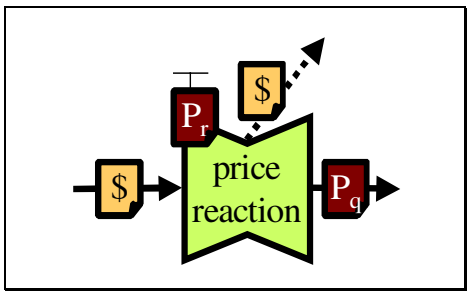


Fig 19 - Price/charge reaction

A high level switch in the price reaction policy determines whether price reaction for the session of concern should be automated or manual. If automated, a QoS control policy is output as above. If manual, the inputted charge record is simply forward onward (shown dotted), typically to some user interface code. Normally, it is bad practice to have software modules just forwarding messages to other modules. Therefore, if possible, the price reaction building block should have the charge advice stream re-directed at source. However, this forwarding facility is provided for the case where the charge record stream is arriving from a remote source without the capability to re-direct it to another port, for instance.

The charge record input is likely to be a one-way flow of messages that immediately result in a one-way flow of output messages to continually adapt the relevant session's QoS.

4.1.7 Distribution

The **distribution** building block (Fig 20) is broadly semantically similar to a routing element. It operates on **control messages, M**, which are simply defined as any of P, SC, C or $. **Policy, P_d**, defines to where different types of input messages are forwarded (including to multiple destinations). The policy effectively defines the control management workflow.

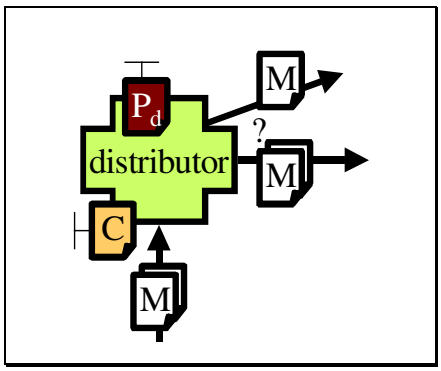


Fig 20 - Distribution

©Copyright 2000, the Members of the M3I Consortium

Policy, P_d, also controls what should happen depending on some aspects of context, C. For instance, if a customer's account is in debit, some messages might need to be routed to set admission control policy that would not at other times. Or if the customer has accepted tariff A rather than tariff B, the former might require a different frequency of meter messaging to the latter, so messages might have to be routed through an aggregation building block (see next). Policy can depend on context, but context never depends on policy, as context is factual.

Distribution can be thought of as a distributed processing environment (DPE). Just as with DPEs, distribution can be implemented on a centralised or distributed model. For instance, all meters could be configured to send their records to a central distribution system, or a distribution building block could be part of each meter, and configured to send the records directly to where they were required next. In both cases, the routing policy is determined centrally, but in the latter case execution of the policy is distributed.

This justifies distribution being a different building block from routing, because a distribution building block can use networking for forwarding, it doesn't need to forward itself. Effectively, a distribution service facilitates inter-process communication at the application layer, being built on top of routing at the network layer. Another difference is that, with distribution, message *destination(s)* are determined by the relevant distribution policy, P_d, for the *type* of message payload. In contrast, a routing element would expect the originator of the message to choose the destination (unicast), or each destination to declare which types of message payload it is interested in receiving (multicast). These models are possible, but not prevalent in distribution.

Policy, P_d, also determines which message flows follow which interaction mode. For instance, some might be polled, others might be set up as event listeners (either with reliable or unreliable communications), others might be simple unreliable 'fire and forget'.

4.1.8 Aggregation

The **aggregation** building block (Fig 21), like charge advice, has two modes of operation; one processing a regular flow of messages, the other responding to arbitrary queries. In its regular mode, this building block aggregates sets of similar message payloads (e.g. session characterisations or charge records). It can aggregate in *time* by buffering and forwarding on a message payload that might give the accumulated total over a longer session. It can aggregate across parallel streams of message payloads (in *space*) by merging detailed differences and summing quantitative values. The level and frequency of output is under the control of **policy, P_a**. In its arbitrary mode, the building block responds to queries about details of past input in order to formulate the requested aggregated reports.

©Copyright 2000, the Members of the M3I Consortium

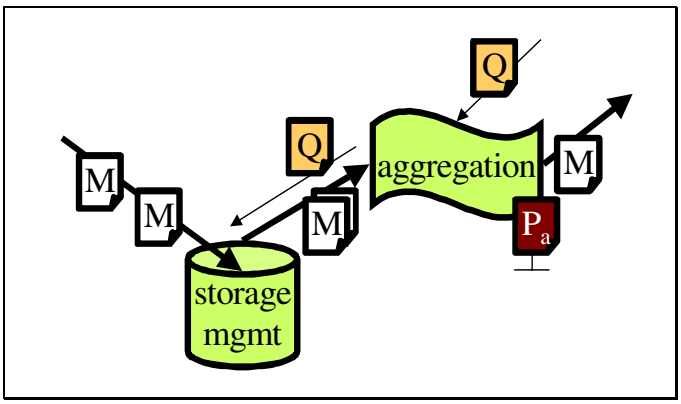


Fig 21 - Aggregation

The essence of aggregation in time is storage. Therefore this building block is the natural place to manage any required hard storage of control messaging for recovery after failures and disasters. The policy, P_a, also determines when records should be archived or expired. Having stored information, this building block is the natural one to manage later access to the records. Thus, in arbitrary mode, this building block forms the basis of a traditional data warehouse. Large databases tend to be slow, therefore, it is likely that the storage management will regularly separate off longer term records to a more batch-mode system, allowing the current records in the regular flow to continue to be processed in a lightweight fashion.

A typical pattern of use in regular mode would be for the distribution building block to send messages into the aggregation building block, asking for them to be returned for onward distribution of the aggregated result. The interaction mode for both inputs and outputs is likely to be reliable connection-oriented request-reply.

4.1.9 Settlement

It is important to include this final, very simple, building block in the set of service operation building blocks, despite it seeming to be one step removed. **Settlement** (Fig 22) is the commitment of a move of real funds between two parties. It is important in a market-controlled system, only because it alters the trust context, C, between the parties, which in turn controls the behaviour of some of the other key building blocks, such as admission control.

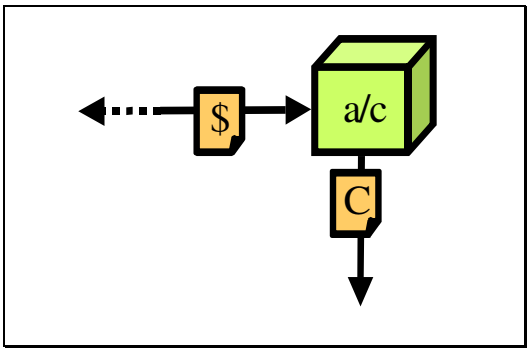


Fig 22 - Settlement

Note that this building block would most likely set the context for the access control building block.

©Copyright 2000, the Members of the M3I Consortium

4.2 Configuration building blocks

4.2.1 Directory

A heavily used building block for many aspects of configuration is the **directory** (Fig 23). A directory stores **associations, A**, between items of information. Many items may be included in each association. An example is given where **X** and **Y** are associated, each of which might be associations themselves. It operates in two modes, read and write. The figure attempts to show read mode on the left, and write mode at the top and the result of write mode appearing inside and at the bottom of the directory symbol.

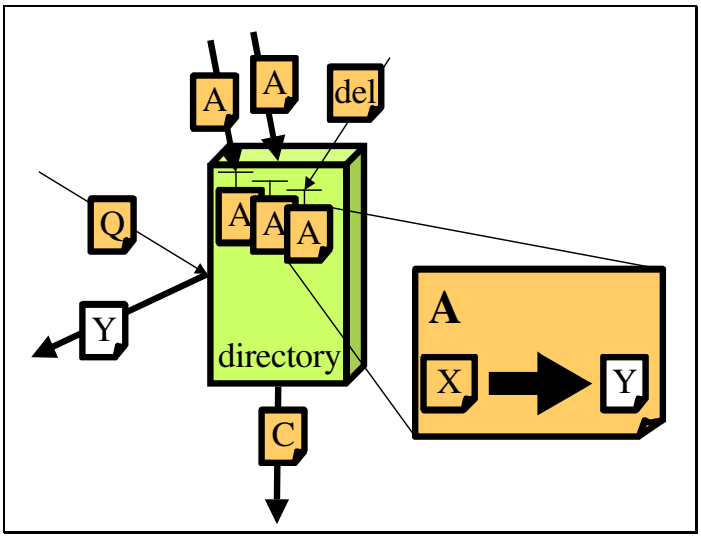


Fig 23 - Directory

Taking read mode first, given an **index term, X**, in a **query, Q**, that requests information of type typeof(Y), then Y will be returned.

Three possible methods are possible in write mode, shown from left to right as update, add and delete. Each write event may result in the generation of a **contextual event message, C**, if other parts of the system require to know the change of context brought about by the change to the directory. Directories may also contain security associations that can determine the security of the directory itself [15].

If a directory is implemented using hard state, it need not be as a centralised database accessed using LDAP [17] or X.500 [18] protocols (although this is common). The DNS is a well-known example of a distributed directory, although its write mode has primarily been through editing configuration files, until recently. Even a set of XML-based Web pages across multiple servers can be used as a distributed directory, particularly if a mixture of human and machine readability is useful.

However, a directory need not necessarily be implemented as a database of hard state. Soft state directories are also possible, such as the session directory used for mbone event announcements [16], where hard state partial directories maintained by contributing participants regularly send repeated directory entries to a multicast channel. From the point of view of a listener, all announcements are merged into one, soft directory. This directory exhibits all the same behaviours as the directory building block described above, including an implicit context message when an entry is added, changed or deleted.

©Copyright 2000, the Members of the M3I Consortium

It is clear from the above description that a directory read may be achieved either in event-listener mode (through the context message), or by request-reply.

4.2.2 Service definition

Starting from the very beginning (a very good place to start, even if it is half-way through the document), it must be clear what behaviour will result from an invocation of service. This is usually defined and stable over very long time-scales, mostly through the core Internet standards. However, a growing number include the word SHOULD, rather than MUST, for major new services (such as the behaviour of multicast packets). Thus, providers each set their own local **service definitions** for certain invocation behaviours (Fig 24) by adding policy statements to the **association, A**, between the language standardised for the invocation, and the resulting behaviour. These associations are typically in natural language rather than for machines to read. However, architecturally, they are associations that would sit well in a directory building block.

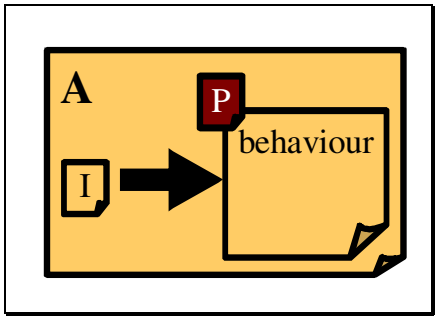


Fig 24 - Service definition

For instance, a service provider might declare connectivity availability targets, or general targets for the likelihood of failure to meet service level agreements. Or it might be declared that sending multicast packets into the network will result in silent failure. Service definitions might also define the exact meaning of otherwise brief statements in service or session offers (see next), particularly the implications of contractual terms.

Typically service definitions are created by the provider stakeholder role. One can consider service definitions to be long-term aspects of an offer (see next).

4.2.3 Tariff

A **tariff** is an algorithm that contains, at its simplest, a vector of **prices, P_t**, to be applied to a vector of session characterisations. More complex algorithms with considerable conditional semantics are also common. See section 4.1.5, Charge advice.

©Copyright 2000, the Members of the M3I Consortium

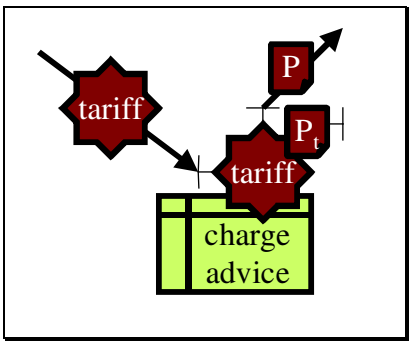


Fig 25 - Tariff configuration

As well as setting the prices in the tariff (Section 4.2.7), it must, of course, also be possible to set the whole tariff that charge advice uses, as shown at the left hand interface in Fig 25. Typically, the charge advice building block for a session will be configured with the tariff that is agreed for that session. As already explained, it is also possible to configure other, hypothetical tariffs in parallel to do comparative costings of a session (quotations).

The top interface on the tariff building block allows the internal structure of its interface to be interrogated (reflection). This allows, for instance, a list of its chargeable parameters to be extracted, which can be used to configure other parts of the system to be compatible with the tariff in use.

4.2.4 Offer

An **offer, O** (Fig 26), is an **association, A**, between various items of information and is therefore well suited to storage in a directory. However, it is also possible to send someone an offer, rather then wait for them to find it in a directory. An offer may be a mixture of natural language and machine-readable information (which may be separate or cross-referenced).

Offer information can be used to describe both services and sessions. It is a highly flexible mechanism for putting service components together into services or sessions without making any physical changes to the service components.

Considering first an offer of *service*, a provider will often offer multiple different service plans to cater for different types of customer with different patterns of usage. Each offer will contain a description of the service and contractual terms, which will include the obligations of the provider and of the customer [42]. Once available services are defined and described, sessions can be created in order to put the services to good use. A session description can define the use of multiple complementary services. It can define the roles required in the session, such as a clearinghouse or charging function provider, and who is assigned to them. Or it can allow participants to choose who they wish to assign to the roles. The difference between a service and a session is described in Section 2.1.

©Copyright 2000, the Members of the M3I Consortium

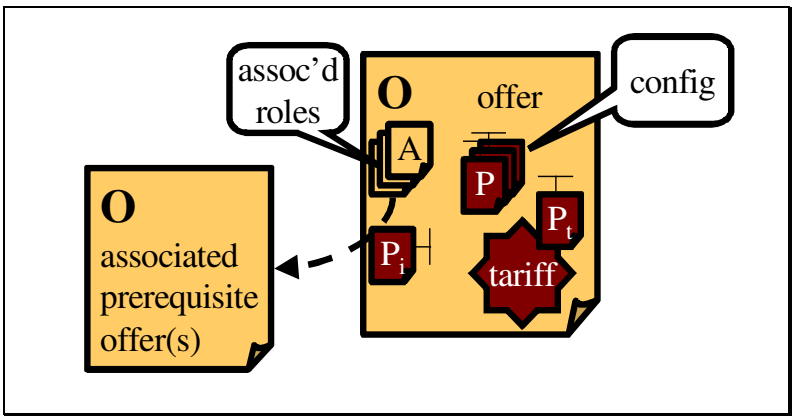


Fig 26 - Offer

An offer is a modular structure with the following major parts, some of which will be optional in each circumstance:

- Description
- Configuration necessities
- Associated roles
  - Associated pre-requisite services
- Contractual terms
  - Obligations of the provider
  - Obligations of the customer
    - Tariff

It is likely that the information necessary to decide whether the offer is even interesting will be made available separately from all the details, which would normally be cross-referenced for checking once a customer's interest has been caught. Thus, the top level description might be all that would initially be listed in a browsable directory listing.

The session description protocol (SDP) [11] is the closest 'standard' that is capable of extension for these purposes.

We now give a little more detail on each module of an offer:

- **Description**

  A (probably textual) description of the service or session

- **Configuration necessities**

  The particular settings to configure the customer's software with the correct address(es) to access the service and to access the management interfaces of the service, for things like price update information and possibly charge advice. These details are shown as various configuration **policies, P**, in the figure. Some are fixed and therefore have no interface. Others have an interface, either for the customer to make her choices, or for the provider to be able to change settings later.

- **Associated roles**

  This module includes **security associations, A**, to associate roles in the session with the identities of individuals or corporate bodies. These would most likely be signed by the session initiator to indicate delegation of trust for these roles in the

©Copyright 2000, the Members of the M3I Consortium

session. Roles nominated might include services such as the clearinghouse, the charging function provider etc. [14], but would also declare who was taking the session specific roles, such as initiator, floor-control and even the person who volunteers to pay for everyone.

- **Associated pre-requisite services**

  The figure shows optional reference(s) to services that are pre-requisites for the session. To be flexible, these references would have to be allowed by inclusion, reference or type. That is:
  - either the whole subsidiary service description could be included
  - or it could be referred to by address
  - or the type of service required could be defined, to allow each participant to choose their own compatible service.

  The inclusion policy, P_i, associated with each reference declares whether the terms of an associated service are overridden by the including offer, or whether the customer should refer to the terms in the associated service. For instance, the inclusion policy affects whether the cost of the daughter service is included in the overall tariff or whether it is additional, and who the cost should be settled with. This policy would also determine who was responsible for support, compensation, etc.

- **Contractual terms**

- **Obligations of the provider**

  The provider's obligations might include reference to the service definitions mentioned above.

- **Obligations of the customer**

  The primary obligation of the customer will be to accept the tariff structure for using the service, with a schedule of current prices. The terms may well include how much freedom the provider has to change prices in the tariff as time passes.

  The customer will most likely also be obliged to provide a means to pay the tariff, such as a credit card number, an e-cash account or to put a coin in a slot (assuming a kiosk).

  If someone else volunteers to pay, the customer will probably be obliged to limit the costs they cause to be incurred to some amount. For instance, if their network provider applies congestion pricing, they might be required to override their QoS controller policy with that of the payer.

  - **Tariff**

    The **tariff** should provide sufficient information for the customer or her machine to be able to predict the charge that would be applied to any service invocation before it is attempted. This is on the assumption that all the properties of the invocation can be predicted (e.g. for a network service tariff: duration, data volume, remote address, class of service or whatever parameters are priced). However, some aspects of the tariff might be openly declared as invocation dependent (e.g. for a network service tariff: congestion pricing or number of hops to remote location).

©Copyright 2000, the Members of the M3I Consortium

The pricing of the tariff would be set by **pricing policies P_t**. The tariff might include an interface for the provider to be able to change prices later.

Machine readability of all this information assumes a good degree more standardisation of service metadata than currently exists. Nonetheless, such information can be a mixture of human and machine readable by referring out to Web pages with links to services where necessary.

Again, it hardly needs saying that a service offer can only be made by the provider stakeholder.

4.2.5 Offer location

Once a provider has declared its service offer(s), or a session has been described in an offer, customers need to be able to **locate** them and choose between them. Any of the usual interaction modes for advertising are likely to be used during this phase, such as browsing, recommendation by reference in e-mail from friends, junk mailings, Web 'push', multicast channel advertising etc. The interaction mode to access an offer once it is located will most likely be request-reply initiated by the customer, although this might be with a locally cached copy of the top level of the offer as is the case in the mbone session directory. Updates to the offer, and in particular the pricing, are likely to be accessed in an event listener mode, such as by joining an announcement channel perhaps using multicast, Web 'push', e-mail list or whatever. Listening to such a channel may be a condition of the service. Fig 23 shows a rather generic summary of all these offer location possibilities, by just the interaction as a directory look-up. Given transport flexibility is clearly important, the only likely standardisation possibility would be to register a service offer payload type for use by a number of payload transports (HTTP [8] & SMTP [13] use MIME payloads [12], SAP [9] & SIP [10] use SDP [11], RSVP [3] defines its own payload extension capabilities).

4.2.6 Offer acceptance

The service offer may require the customer to accept it explicitly. This may be because the offer includes choices the customer has to make, such as friends and family addresses etc. Alternatively, the customer may be warned that using the service involves acceptance of the offer implicitly, *thus making the offer acceptance structure below redundant*. This latter case might be appropriate if the customer already has access to the offered service through acceptance of an earlier offer, and this new offer simply adds a new service feature (e.g. multicast or reservations).

©Copyright 2000, the Members of the M3I Consortium

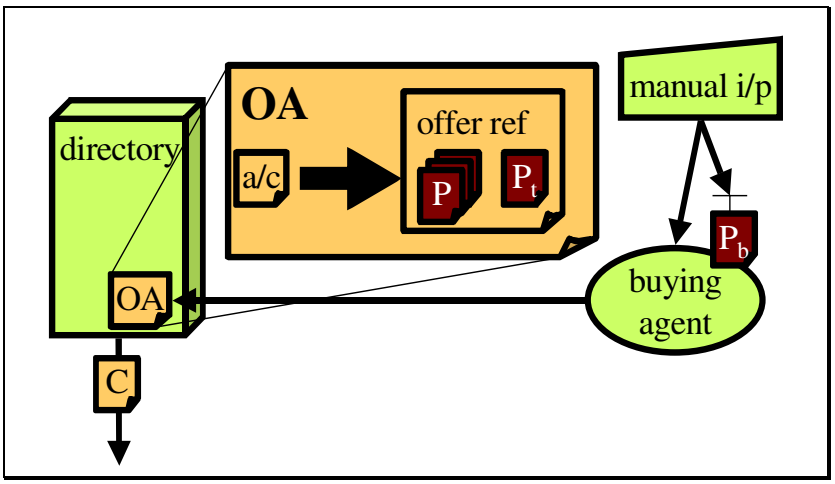


Fig 27 - Offer acceptance

Fig 27 shows the case where there is an explicit **offer acceptance, OA**. In fact, it goes further showing automated offer acceptance, where a customer's buying agent has been programmed to search for optimal service offers that match **policy, P_b**, much as e-commerce 'shopbots' do on the Web. Having found an optimal offer, it may require user confirmation before sending the acceptance to the address of the directory given in the original offer. The acceptance is essentially an **association, A**, between the accepting customer and a reference to the offer. The customer is shown referring to herself by her customer account in the figure. The acceptance also includes the completed customer policy details, such as the chosen payment method, the chosen friends and family addresses etc. These are shown wrapped into the acceptance as a set of fixed **policies**, denoted **P**, which set the customer **context, C** output from the directory.

4.2.7 Price setting

The role of the **price setting** building block (Fig 28a) does *not* encompass controlling all elements that affect the *charge from* a tariff. It only sets the *price in* a tariff. As an example, the tariff may be constructed so as to make the charge dependent on congestion marking rate by including a price per mark. An element that controls marking strategy does not set this *price;* it only determines the resulting *charge*. On the other hand, our price setting building block sets the price and, as a result, affects the charge. Price setting is therefore a higher level of control.

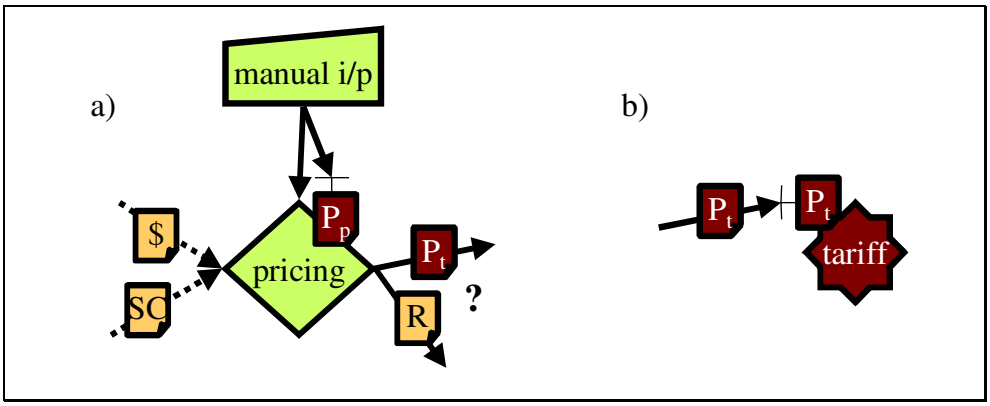


Fig 28 - Price setting

©Copyright 2000, the Members of the M3I Consortium

There are two different classes of price setting function:

- One sets a nominal (internal) price of a resource to enable an internal market in resource management for a network.
- The other sets customer prices within the each of the tariffs offered to customers as part of their service plan.

Both may exist together, with the latter very probably feeding internal pricing information to the former, for it to set external pricing. Customer pricing, on the other hand, might be a completely manual process, with internal pricing being fed to manual decision makers. Then again, with congestion marking, there may be no need for internal pricing, as internal resource prices are directly exposed to customers. All that is then necessary is to set the external price per mark; a process which will need background information on general resource utilisation. Internal price setting is likely to be highly distributed, with every network resource monitoring itself and reporting its internal price. External pricing is likely to be centralised, with information on demand for each service plan collected together from every charging and accounting system in order to work out the best external pricing strategy. Thus, a price setting building block will need to be able to accept either usage records (session characterisations) or charge records.

Thus, the figure shows how a pricing building block has two options for inputs (shown dotted) in order to make its decisions. Either it will use **charge record** inputs**, $**, or **session characterisation** inputs, **SC**. Recall that $ includes a reference to the session characterisation, as well as the charge for it. Thus, $ describes both revenue and cost, making it ideal for *external* pricing decisions. On the other hand, SC contains no assumptions about the current price, so is more useful for internal pricing decisions. Note that the 'session' of interest may well relate to all traffic flowing through a resource, rather than some finer-grained session. Indeed, $ may arrive via the aggregation building block, and relate to a whole network of resources. $ may be a characterisation of current load, or it may represent a customer's bid for a session in an auction.

A price-setting decision is generally designed to maximise profit, therefore it cannot be assumed that as load increases, **price, P_t**, should automatically be increased to back it off. The other action available to the pricing building block in the figure is to make a request to increase the capacity available (**resource, R**). This may be possible instantaneously (e.g. by entering into a bandwidth auction), or may involve considerable delay while extra physical resources are deployed. The **policy, P_p**, of the price setting function itself is the provider's primary means to decide between the various courses of action available. For instance, the policy may set the delay before raising the price at times of unplanned congestion, so that customers experience degraded quality rather than increased pricing. Similarly, it may set how the price should behave after prolonged periods of light load. Thus the price setting policy must be continuously taking account of the action of competitors, as it essentially determines where the provider wishes to sit in the marketplace [7]. Hence we have shown the definition of price setting policy as the provider's 'top level' where their only **manual input** is required. In fact, we have also shown the option of manually controlling individual price changes, or at least confirming recommendations of the price-setting function.

©Copyright 2000, the Members of the M3I Consortium

Being, higher level, external price setting decisions are likely to be made on longer time-scales than other ways available to the system to cope with load fluctuations, which are likely to themselves assume pricing is stable. Examples of likely shorter term strategies are re-routing, borrowing resources from lower classes of service, dynamically deploying spare capacity or increasing congestion marking rate (assuming congestion marks are priced).

This building block is discussed in detail in [35].

Fig 28b show the result of setting prices - the pricing policy, P_t, in the tariff is configured. There may not be a direct interaction from the price setting building block on the left to the tariff on the right. The new pricing may reach the tariff indirectly through a tariff directory or other intermediate stages.

4.2.8 Address allocation

Having set up all the arrangements at the application level, we now drop down to the interfaces more directly related to the network again. The first information necessary to create context in which to use the network is a simple **address allocation, AA** (Fig 29). Note that we can use this association for allocating any name or address, not just network addresses.

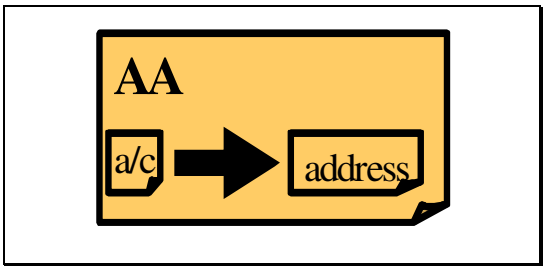


Fig 29 - Address allocation

The allocation is an **association, A**, between an account and an address. We deliberately choose the term account, rather than identity, to cater for anonymous customers, as we only need creditworthiness, not necessarily identity. We assume that the address allocation is stored in a directory (see earlier). An allocation event will then output a **contextual event message, C**, which will need to be distributed to any part of the system that uses addresses or names to identify who is liable to pay etc.

4.2.9 Classifier and meter configuration

A typical event triggered by an address allocation would be to **configure classifiers and meters** to monitor activity by the user of the address. Similar mechanisms could be re-used for unrelated purposes, such as general network management.

©Copyright 2000, the Members of the M3I Consortium

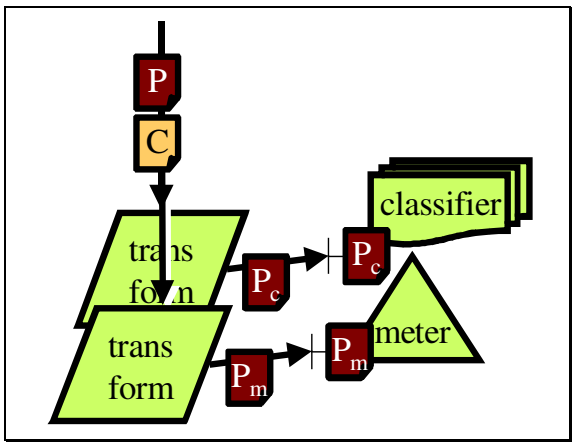


Fig 30 - Classifier and meter configuration

In Fig 30 we show the contextual message payload warning of the new address allocation being transformed into a form of rules language suitable to configure the classifiers and meters.

Also shown as an input is the policy, P, which configures which parameters require measurement for the tariff(s) in operation, and other configuration information, such as how long it should be between reports.

4.2.10 Task-reaction policy association

In order to configure the price reaction building block, it is necessary to use a policy that is relevant to the flow of invocations being controlled. With the integrated services architecture [19], this is directly under the control of the application author, which makes considerable sense. However, with the initial differentiated services proposal [20], this policy was expected to be set by a network manager for a corporation, based on such characteristics as the application protocol in use. This is likely to be a poor indicator of the required policy [21]. It is also useful to be able to add QoS to applications not originally written to take advantage of it. Therefore we need a general mechanism to associate the context of a flow with a policy to control it. This is our justification for introducing the **task-reaction policy association, RA**, in Fig 31.

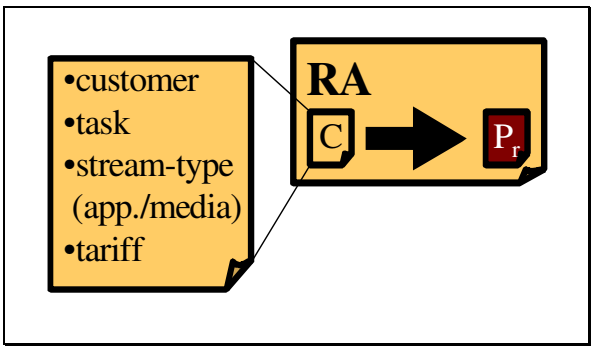


Fig 31 - Task-reaction policy association

The **context, C**, of the flow is defined by who the customer is, their task, the type of stream in terms of application and media and the tariff being applied. All this contextual information determines the **price reaction policy, P_r**.

©Copyright 2000, the Members of the M3I Consortium

This **association, A**, would typically be stored in a directory. If an exact match were not available for a particular context, it might be possible to find the nearest match available. Of course, it would be necessary to allow new policies to be created for new contexts.

4.2.11 Price reaction policy configuration

Assuming the repository of task-reaction policy associations introduced above, Fig 32 shows the **price reaction** building block being configured with the **policy, P_r**, relevant to a particular **context, C**, by looking up the **task-reaction policy association, RA**, between the two in a **directory**.

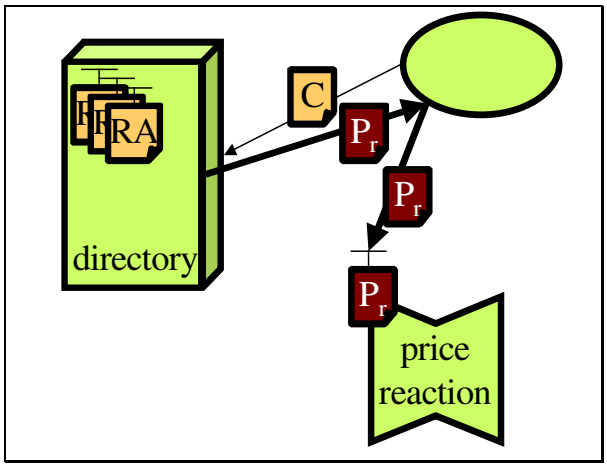


Fig 32 - Price reaction policy configuration

Note that the unlabelled function in the figure that initiates the directory lookup and consequently configures price reaction with the returned policy may be any arbitrary function.

4.2.12 Distribution and aggregation configuration

For completeness, Fig 33 shows the **distribution and aggregation functions being configured** by application of the relevant **policies, P_d & P_a**. The configuration of the distribution function also depends on the current **context, C**.

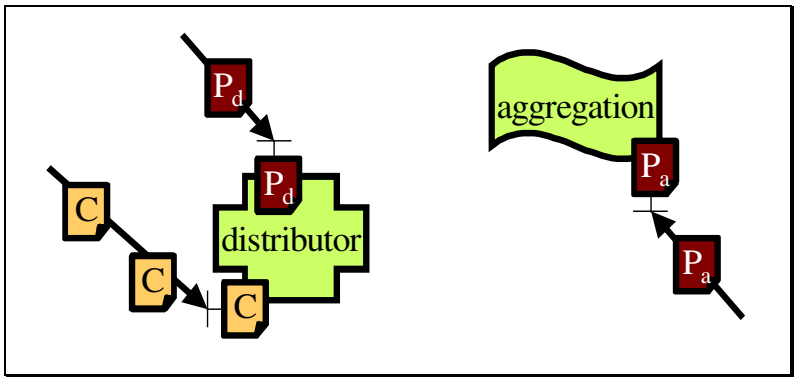


Fig 33 - Distribution and aggregation configuration

Note that a valid policy for the distribution function would be to filter incoming session characterisation message payloads, to identify certain of them as changes of context, and present these to its own context setting interface.

©Copyright 2000, the Members of the M3I Consortium

4.3 Utility building blocks

4.3.1 Correlation

The **correlation** building block (Fig 34) is likely to be used primarily as a security defence. It operates on **management control** message payloads, **M** (defined earlier). It takes at least two inputs, either in time or space and gives an output **context message, C**, that is the result of whether the inputs correlate. The type of correlation is a property of the particular building block employed. It might simply compare two inputs and output whether they are similar or identical. This could be used to audit a set of measurements against a sample control. A significant number of false matches might be used to trigger access denial. A more sophisticated correlation function might look for oddities in the patterns seen in a stream of inputs. Such a function might be used as an intruder detection system (IDS).

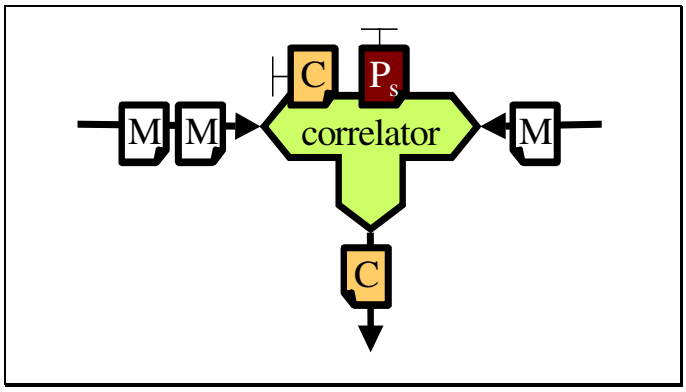


Fig 34 - Correlation

The strictness of the correlation test is under the control of **security policy, P_s**. The behaviour of this policy may also depend on the current **context, C**.

4.3.2 Transformation

Our final building block, is a simple transformation gateway that caters for the need to change control message payloads, M_1, into output control message payload, M_2, of another format or type.

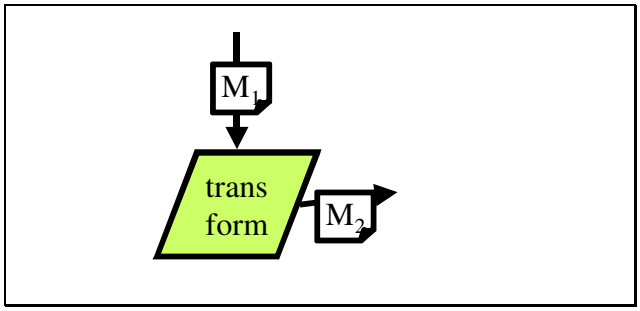


Fig 35 - Transformaton

4.4 Applications

We have now completed the descriptions of all the building blocks that feature *inside* the M3I architecture. We now return to the applications *outside* the architecture that we started to discuss in the applications overview (Section 0). On the theme of building

©Copyright 2000, the Members of the M3I Consortium

blocks, we shall now attempt to further characterise some high level types of application, so that we can define a few characteristic application building blocks that represent generic application types. This is not at the level of data, file-transfer, real-time or whatever. We are thinking more at the level of **selling**, **buying** or **using**, as introduced in the overview, but we will now get slightly more specific, by distinguishing between applications for:

- selling and buying Internet (not just network) services that use the network service
- selling and buying Internet sessions that use the network service

Services are potential sessions. They represent investment in a particular idea of what others will find useful. Sessions represent actual use of services. Both selling services and selling sessions can either involve creating more of the same, or inventing new types of service or session.

Why choose these two particular things? For two reasons: firstly to show that 'however they want' includes new patterns of use as yet unimagined. And secondly to remind ourselves that, not only do we have to provide the network service interfaces, but we have to provide the interfaces to the network support systems. But further, not only interfaces to the support systems to manage current sessions, but also to allow the network service to be included within new services and sessions, without making manual arrangements with the network provider. These last interfaces are the ones that allow commercial openness.

So we can now show (Fig 36) the regular, service interfaces of the current Internet, with extra management interfaces for six characteristic application building blocks. Note that the M3I 'cloud' is intended to imply the systems of both providers and customers, not just the former. The applications around the outside of the M3I are the applications that will be *using* the M3I building blocks introduced in Section 4. These are the independent variables - in effect, the building blocks we can't predict. As such, they are part of the architecture, and must be built or modelled to test the architecture. However, we can only invent scenarios to pretend we know what they will be like, we cannot really know. Below we try to give a flavour of the sorts of things these applications will do (each may be more than one application):

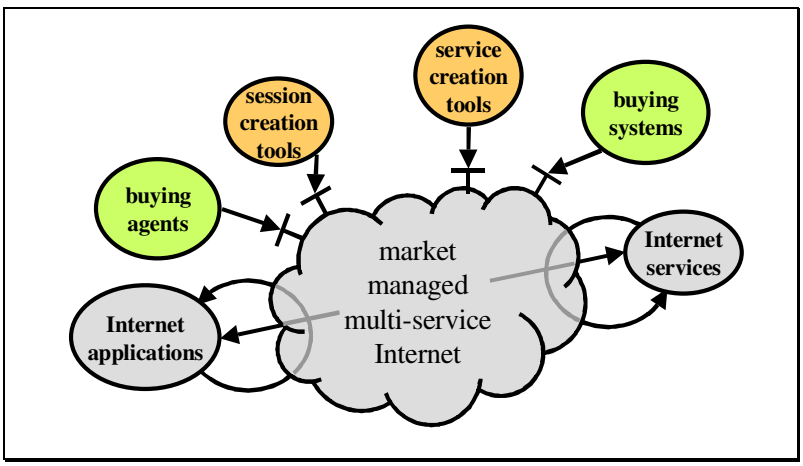


Fig 36 - Applications of an M3I

Internet applications (run by end-customers) and **Internet services** (run by Internet service providers):

©Copyright 2000, the Members of the M3I Consortium

- Applications and services that use the network service, require and specify QoS and generally load the network in relatively unpredictable patterns. Note that applications and services don't only communicate with each other. Some applications communicate with other applications, and some services communicate with other services (represented by the loops in the figure).

Buying agents (for end-customers) and **Buying systems** (for Internet service providers) each cover similar roles:

- Look after the commercial interests of the customer stakeholder
- Look for competing products and services, and compare charges
- Allow the setting of spending budgets and check spending against them
- Allow the setting of task-related policy over automatic spending
- Decide when to settle.

Service Creation tools (run by potential Internet service providers) and **Session creation tools** (run primarily by end-customers)

- Advertise the existence of new services or sessions
- Declare how to use the services or sessions
- Declare and certify the commercial arrangements for new services or sessions (e.g. offering to pay for the end-customer's network usage in return for payment for a bundled product or service)

Our intention is that network service creation should be no different from creation of any other service. Thus, if a network provider decides to offer a new service, or service plan, it would use a service creation tool to advertise itself. Obviously, a service creation tool only does the easy bit - declaring the service. The difficult bit comes before this. It is still necessary to make all the investment in network resources, think up a neat tariff, decide on the strategy for dealing with competition, demand and supply fluctuations, set routing policy, set future pricing policy (Section 4.2.7) etc.

Thus, one would imagine there were a need for a provider management agent to balance the customer management agent. In the present architecture, this function is enacted by a combination of the price-setting building block and manual input, as the industry is a long way off automatically deciding what are good business ideas and creating them without human input!

4.5 Clarifications

Below we make some points for exactness, which we felt would have broken up the flow if they had been mentioned where they were most relevant. They are in no particular order.

We have not intended to imply that all service elements of a certain type are identical, the only differentiation being based on policy. For instance, two QoS control building blocks might have completely different algorithms and semantics, whatever policy they are configured with. Our choices of building blocks are strictly *classes* of building blocks that may be sub-classified differently once we move from architectural abstraction to concrete design.

It may have been noticed in the diffserv shaper component example that multiple outputs were suddenly multiplexed into one building block. We decided we would make

©Copyright 2000, the Members of the M3I Consortium

this quite legal, rather than introducing a multiplexing building block, because the separate flows from a classifier are often virtual flows, made by creating a handle for each packet in a real queue simply to classify it, and without changing the order in the real queue. Thus, multiplexing simply involves forgetting the handles, which hardly needs another building block.

Also, in the same example, a single policy was used for multiple meter building blocks. As each meter was measuring the same characterisation of the flow given to it by the classifier, this made sense. This can either be defined in a component diagram, or in a deployment table, which would make clear that the meter policy was per hop, not per class.

When we get round to composing an RSVP invocation, it will be clear why we called packets invocations. The reasoning was to ensure we could route signalling as well as data packets. The best way to do this is to treat all packets as invocations, and just separate out the signalling in a classifier, so that it invokes something different than routing (just as it's done in the real world, of course). We also wanted to be able to introduce different routing for different types of invocations [25].

Note that a routing advertisement is semantically identical to a service offer. It can include border policy, and could, in the future, include, or refer to, a price. The routing information base is semantically equivalent to the directory used to store accepted offers and their associated policies. Whether a router decides to accept an offer depends on its cost (cf. price) compared to competitor's offers. This is modelled by the buying agent. The border policies of the router deciding whether to accept a route advertisement, are equivalent to the buying agent's policy.

Settlement can be fairly decoupled from the rest of the system. What coupling there is, is often quite subtle. Often, the provider will alter its access control rules dependent on the settlement record of the customer. This is sometimes explicitly declared, but it is often implicit or even secret. In the implicit situations, the customer often learns the rules from how the service surround then the service itself degrades if they don't pay their outstanding debts. For instance, a 'bill' can't be assumed to be a direct trigger of payment. It is a (big) hint, that payment is expected. The customer can find that paying less than the whole bill is sometimes still sufficient to keep the provider happy. In other words, 'pay as you fancy' with explicitly declared penalties is often the coupling that needs to be composed in order to model settlement.

5 Interfaces

Our set of building blocks numbers fourteen, plus six that characterise applications external to the system. We now list each type of interface between them using the approach in Section 2.6. We identify internal APIs in italics followed by (A). All other interfaces will require a wire protocol.

We will split the discussion into operation then configuration, as we did for building blocks.

5.1 Network and lower layer interfaces

Network service invocations, I, represent **IP packets, I_p**, or **QoS signalling, I_q (e.g. RSVP [3]**). **Setting resource, R**, if it is possible to do remotely, is done using protocols

©Copyright 2000, the Members of the M3I Consortium

specific to the link layer in use. Setting **routing policy, P_n**, and propagating it throughout the Internet, is done with standard routing protocols.

None of these protocols are expected to be altered by the M3I project. Therefore, we will leave these out of our deliberations on interfaces.

5.2 Service operation interfaces

5.2.1 Session characterisation interface family

<table>
<tr><th>input to

output from</th><th>inter-action mode</th><th>QoS ctrl</th><th>Chg. adv-ice</th><th>Dis-trib-utor</th><th>Aggre -gator</th><th>Pric-ing</th><th>Price react-ion</th><th>Cor-relat-ion</th><th>Trans -form-ation</th><th>App-licat-ion</th></tr>
<tr><td>Meter</td><td>oneway</td><td>SC1 (A)</td><td>SC2 (A)</td><td colspan="2">SC3</td><td>- SC5</td><td>-</td><td>SC3</td><td>?</td><td>SC3?</td></tr>
<tr><td>Distributor</td><td>oneway</td><td>-</td><td colspan="3">SC4</td><td></td><td></td><td>SC4</td><td>?</td><td>SC4?</td></tr>
<tr><td rowspan="2">Aggregator</td><td>oneway</td><td></td><td colspan="3"></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>query-response</td><td colspan="9">-</td><td>SC6</td></tr>
<tr><td>Price reaction</td><td>oneway</td><td colspan="9">-</td><td>$1?</td></tr>
<tr><td rowspan="2">Charge advice</td><td>oneway</td><td colspan="2">-</td><td colspan="5">$1</td><td>$1?</td><td>$1?</td></tr>
<tr><td>query-response</td><td colspan="9">-</td><td>$2</td></tr>
<tr><td>Transformat'n</td><td>?</td><td>?</td><td>?</td><td>?</td><td>?</td><td>?</td><td>?</td><td>?</td><td>?</td><td>?</td></tr>
</table>

Table 5 - Session characterisation interface family

We assume at most five session characterisation interface types, SC1-6, and at most two charge record interface types, $1-2.

SC interfaces carry at least the following:

- session scope (addresses etc)
- measurement start & stop times
- parameters measured (by name or reference)
- usage counters for the parameters measured

They may also include an account identifier, authentication mechanisms, usage report identifier, report send timestamp, meter identifier and an of course an extensibility mechanism.

Examples of implementations of implementations of protocols that fit the SC interface are those used by cisco netflow and the open source NeTraMet which complies with the IETF real-time flow measurement experimental RFCs [5].

SC1 & SC2: These two interfaces are likely to be internal APIs only - unlikely to require a wire protocol. It is hoped to design them as one API. The meter-QoS control interface (SC1) will operate at approaching per-packet granularity and therefore have to be highly optimised within a single address space. Similarly, if a meter's output is plugged directly into charge advice without any mediation (SC2), it is likely that this will be as

©Copyright 2000, the Members of the M3I Consortium

part of a tight, price-based QoS control loop with similar performance requirements to SC1.

SC3 & SC4: These two interfaces take data from mediation systems rather than meters. Many use-cases require them to be distributed, therefore a wire protocol is required. Again, it is hoped to design these two as one interface and therefore one protocol. Ideally the API should also be the same as SC1 & SC2.

SC5: This interface transfers accounting records directly between mediation systems or meters themselves and price setting. It would be necessary where price setting is distributed, perhaps only being used to determine an internal price. The M3I Pricing Mechanisms Design [35] discusses the use of this interface and wire protocol in more depth.

SC6: All the above interfaces process a one-way flow of messages. This interface and wire protocol allows applications to make arbitrary queries on mediation systems for arbitrary accounting records. It carries similar information to SC3 & 4, but is likely to need to be more extensible and perhaps, as a consequence, less efficient. It *may* be possible for this to use the same protocol as SC3 & 4.

Charge advice ($) type interfaces carry at least:

- an amount of money
- a currency
- a reference to the relevant session characterisation (accounting record).

The currency may not be a real national currency, e.g. special drawing rights (SDR)[16], e-cash or 'test'. The accounting record may be included, rather than referenced.

The difference between **$1** and **$2** is that the former are regularly fed out of the charging system as service operation proceeds, while the latter are in response to arbitrary queries. Therefore, $1 must be optimised for performance, while $2 will need to be highly flexible, essentially being the result of an arbitrary database query. $2 would most likely be a standard format for database reports.

The charge advice interfaces are included with the session characterisation interfaces as they do essentially the same job, but add extra information (the charge). Therefore, there is a possibility that the two types may be derived from a single interface and possibly a single protocol.

5.2.2 Policy interface family (operation)

input to output from	inter-action mode	QoS ctrl	Net-work-ing
Price reaction	request-confirm	P_q or I_q	I_q
Application	request-confirm	P_q or I_q	I_q

Table 6 - QoS control policy interface family

[16] An international basket of currencies used in many international markets, including communications interconnect.

©Copyright 2000, the Members of the M3I Consortium

The contents of a QoS control policy are discussed in [31]. It may either re-use existing static QoS signalling or define the parameters of a dynamically adapting QoS controller. Although no use cases have been proposed where this interface is distributed, it seems possible that this interface may well need a wire protocol definition. However, the API will need to be highly efficient, as it could well be implemented as a system call to the operating system kernel in the future, if not before.

Note that standard RSVP signalling, I_q, is available as a wire protocol (and API), if a static QoS policy is sufficient. RSVP may be extendable for dynamic QoS adaptation policies.

5.3 Configuration interfaces

5.3.1 Policy interface family (configuration)

input to / output from	inter-action mode	Offer direct-ory	Chg. adv-ice	Price react-ion	Buy-ing agent
Offer directory	listener	T1 & P_t1			
	query-response	T2 & P_t2			
Price setting	request-confirm	P_t1 & P_t2			

Table 7 - Tariff interface family
T = tariff protocol

The issues concerning a tariff protocol and API and a pricing protocol and API are discussed in [35].

Ideally a tariff should be allowed to be an arbitrary algorithm, which is feasible because the edge pricing principle dictates that a tariff's primary use should only be for locally connected customers. Therefore an ISP could supply the software to understand its own proprietary tariff description protocols. However, at the same time a tariff may need to be understood across the wider Internet, for example by global clearing houses. Therefore, at least the interface to a tariff (described in Section 4.2.3) will need to be standardised and there will probably be a need for a standard tariff description protocol or protocols.

TUD's previous work referred to in the above reference embeds pricing infromation in an extension to RSVP. BT's previous work [36] establishes a rudimentary wire protocol for announcing tariffs and listening for updates to read them off the wire. It also defines a Java API for one type of tariff payload.

If a tariff is arbitrary, its changes to its pricing policy (i.e. changes to single coefficients of the algorithm) will also have to be arbitrary. The BT work above uses the same protocol and API to transmit pricing changes to a tariff (P_t) as it does to transmit tariffs.

input to / output from	inter-action mode	Price react-ion	Buy-ing agent

©Copyright 2000, the Members of the M3I Consortium

input to / output from	inter-action mode	Price react-ion	Buy-ing agent
Buying agent	request-confirm	P_r1	-
	event listener	P_r2	-
task-reaction association directory	query-response	-	P_r

Table 8 - Price reaction policy interface family

The contents of a price reaction policy are discussed in [31]. A use case has already been proposed where the receiver is given the sender's price reaction policy, therefore, these policy interfaces must have both an API and wire protocol defined. It is possible P_r1 could be carried by a SIP-like protocol and P_r2 by a SAP-like protocol.

The query-response mode is likely to use a standard directory protocol, such as LDAP to transfer the same payload.

input to / output from	inter-action mode	Class-ifier	Meter	Trans-form-ation
Tariff	listener	-		P
Offer acceptance directory	listener			OA
Addr. alloc'n directory	listener			AA
Transformat'n	listener	P_c	P_m	-

Table 9 - Classifier & meter policy interface family

A classifier policy and a meter policy consist of rules for the classifier's filter and for the granularity of what the meter measures - that is, what it measures, what it doesn't measure, any logical conditions on what is measured (e.g. only measure the packet size for outgoing packets with non-zero diffserv code point). The RSVP standard describes ways to define a broad range of session scopes [3]. The real-time flow measurement (RTFM) working group of the IETF has defined the simple rules language (SRL) [43] as a language to define rules for a meter. It is based on the Berkeley packet filter (BPF) interface.

Classifiers and meters are regularly controlled remotely. Therefore, both P_c and P_m need to have wire protocols specified.

In order to automate the creation of classifier and meter rules, their origin can be traced to the service offer that usage is being measured for. Thus, if the service offer is described electronically, P_c and P_m can theoretically be derived from the contents of the offer, in particular the offer acceptance if it exists. This should include (possibly by reference) the tariff, any address allocations, and other configuration information, which should even allow the meter needing configuration to be identified.

©Copyright 2000, the Members of the M3I Consortium

Thus, as a general case, it should be possible to write a transformation algorithm to convert the format of the contents of an offer acceptance into classifier and meter rules. This is discussed further in Sections 6.1.3 and 6.1.4 on the configuration component, including the idea of reflection on the tariff interface.

input to / output from	inter-action mode	Distr-ibutor	Aggr-egat-or
Offer acceptance directory	listener	P_d	P_a

Table 10 - Distributor and aggregator policy interface family

We assume that, on acceptance of an offer, sufficient information will be available in the acceptance to configure the mediation systems (**distribution, P_d, and aggregation, P_a**) for the new customer. This interface is likely to be distributed and will need to interact on the event listener model for efficiency. This is discussed further in Sections 6.1.3 and 6.1.4 on the configuration component.

There remain three very high level policy types for which we have no specific interface or protocol: the price **setting policy, P_p, the buying policy, P_b, and security policy, P_s**. These policies are applied to the price setting, buying agent and correlator respectively. However, these are likely to be highly proprietary and set through the user interface of these very high level building blocks.

5.3.2 Context interface family

input to / output from	inter-action mode	QoS ctrl	Chg. adv-ice	Dis-trib-utor	Cor-relat-ion	Task-react-ion assoc
Settlement	listener	C1				-
Correlator	listener					
Buying agent	request-confirm	-				C2

Table 11 - Context interface family

5.3.3 Association interface family

Throughout the configuration building blocks section, the following types of associations were introduced: offer, offer acceptance, address allocation, svc definition, task-reaction policy assoc.

We presume that all these types of associations will be delivered into and out of directories and therefore will most likely use a standard directory protocol, such as LDAP [17].

©Copyright 2000, the Members of the M3I Consortium

6 Compositions

Below we take the approach already described in Section 2.8, by defining re-usable components, then defining how they are composed to build systems that will support each of the use cases of Section 2.3.

6.1 Components

Table 12 is a list of the sub-systems that appear in at least one of the use cases in Section 2.3 with the symbols used for them. The third column identifies whether the item that implements that sub-system is:

- A: an arbitrary Application outside the scope of the M3I infrastructure
- B: itself a Building block
- C: a Component to be built from building blocks
- '+' denotes a distributed arrangement of many instances of building blocks and/or components.

Use case symbol	Use case sub-system	A/B/C ?	Service building block(s) or component
E_p	Enterprise policy agent (selling)	A	Arbitrary application (directory?)
E_c	Enterprise policy agent (buying)	A	Arbitrary application (agent supported by directory?)
AM	Application or middleware	A	Application (arbitrary)
O	Offer directory	B	Directory
Pr_c	Price reactor	B	Price reaction
Pr_p	Price setting	B+	Price setting
Q	QoS manager	C	QoS manager (QoS control, meter and classifier)
CA_c	Charging & accounting (end-customer grade)	C	Mini charging & accounting
CA_p or CA_c	Charging & accounting (provider grade)	C+	Charging & accounting (mediation components plus charge advice & correlator building blocks)
		C	Mediation (configuration component and distributor, aggregator, and transformation building blocks)
		C	Configuration (transformations)
M	Metering system	C	Meter system (meter & classifier)
S_p	Network service	B+	Networking service

Table 12 - Service building blocks for use case sub-systems

©Copyright 2000, the Members of the M3I Consortium

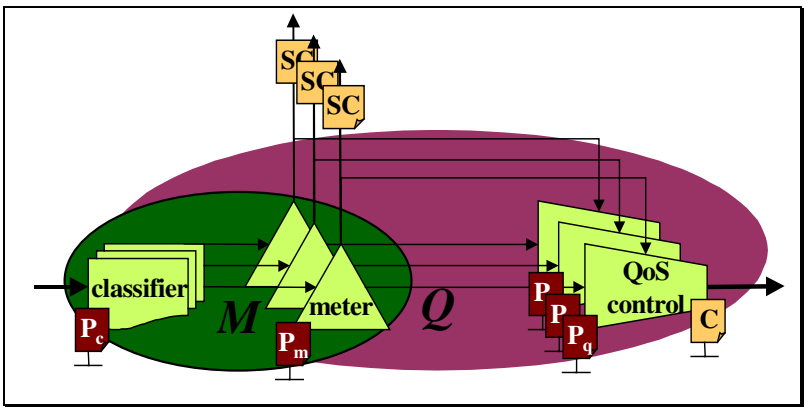
The fourth column, gives a description of the item that implements each use case sub-system. For applications (A), an attempt is made to characterise the type of application. For service building blocks (B) the specific building block from Section 4 needed to build that sub-system is declared. For components (C), the name of the relevant component we are about to define is introduced, and if space allows, a list of the building blocks of which it consists.

Note that names of the sub-systems used in the use cases and names of components are in some cases only subtly different from those of related building blocks. For instance, the QoS *manager* that appears in most use cases and the QoS *manager* service component that implements this is deliberately named differently from the QoS *controller* building block, which is just one part of it. More subtle still, we distinguish the term 'meter' from 'meter *system*'. We mean the service building block by the former, but the latter is a term reserved to mean the component consisting of a meter and a classifier, which we introduce next.

Further note that the provider grade charging and accounting system is potentially highly distributed and therefore smaller components need to be defined, from which to build it.

6.1.1 Meter system service component

A meter system service component (M in Fig 37) is a very simple composition of just a classifier and one meter per classification. A meter system service component exposes all the interfaces of its underlying building blocks, except the internal interface between the classifier and the meter, across which traffic flows.


SC
SC
SC
classifier
M
meter
Q
QoS
control
C
P
P
P

- P_m in
- P_q in
- C in
- data flow in
- data flow out
- SC out

There is, however, no SC input interface, as this is internalised. Note that the context interface is missing from the customer QoS control building block, Q_c, in Fig 38, as it is irrelevant for an end-customer. A provider QoS controller would include this additional interface (for access control, as discussed in Section 4.1.4).

6.1.3 Mini charging & accounting service component

Fig 38 shows four of the use case sub-systems listed in Table 12 in a typical setting on a single customer host such as in the dynamic price handler or guaranteed service provider scenarios (M3I requirements [14]). For comparison, the inset bottom left of Fig 38 shows the sub-systems as drawn earlier when describing the use cases for these scenarios (Fig 2 on p12 or Fig 5 on p19). The rest of the figure mirrors the same layout as the inset, but shows how one of the sub-systems (Pr_c) is in fact a simple service building block, while the other three[17] are components consisting of building blocks, shown within them. The interfaces of each building block similarly map to the numbered steps from the use case in the inset. To aid visualisation of these mappings, the step numbers are repeated on the detailed component diagram. However, a few more configuration interfaces appear in the detailed component diagram than were shown in the use case diagram. This is because it was stated at the start of the use case discussion that the use cases wouldn't cover all aspects of configuration, only those relevant to market control.

[17] In this case, the meter sub-system is implemented by re-use of the meter sub-system in the QoS manager, so there are really only two other components in the diagram.

©Copyright 2000, the Members of the M3I Consortium

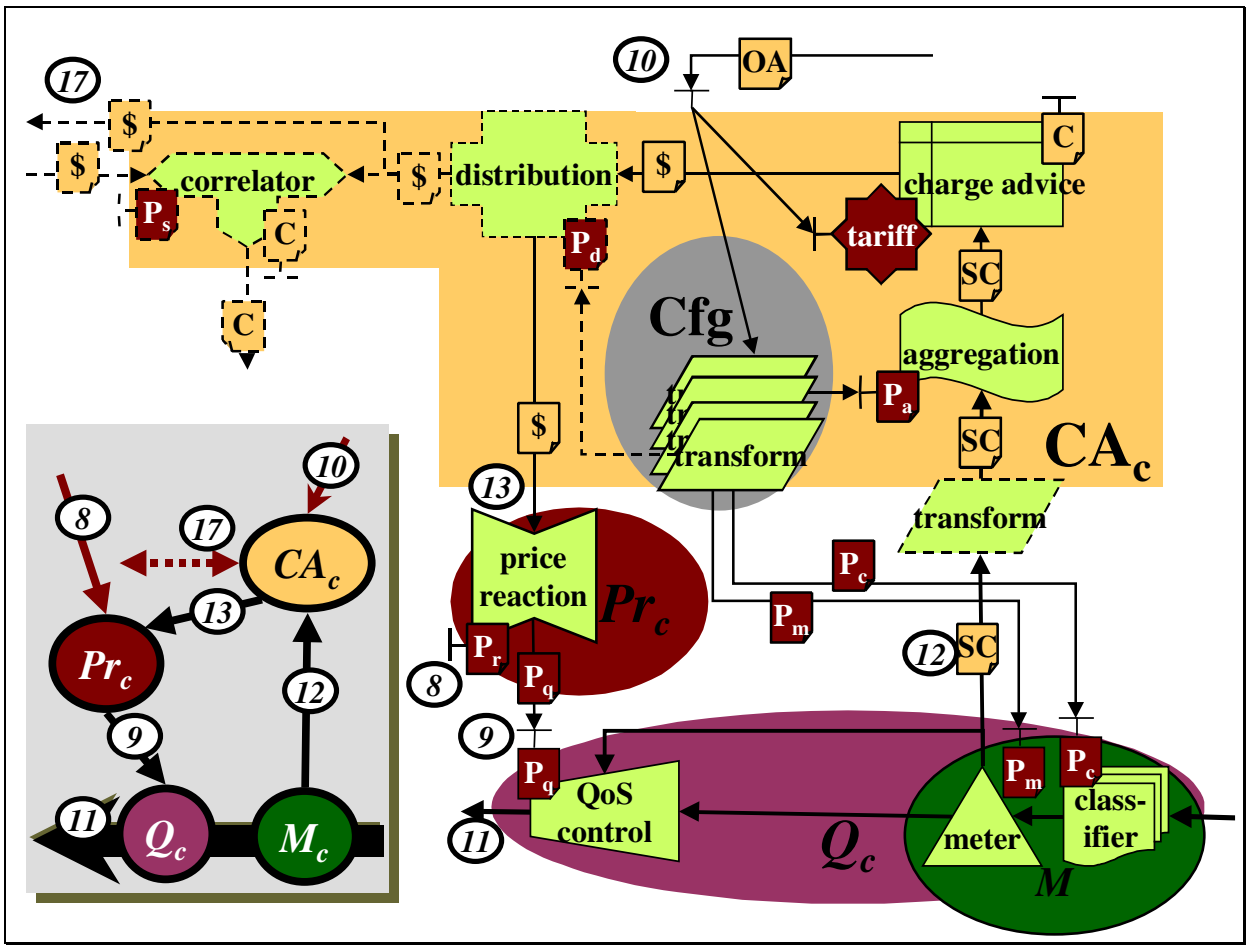


Fig 38 - End-customer service components

We will call the new service component in this diagram 'mini charging and accounting'. It implements a customer-side charging and accounting system, CA_c, on a single host. It would implement the **risk broker's** customer-side charging and accounting system in the **guaranteed service provider** scenario or the **end-customer's** in the **dynamic price handler** scenario.

The main function of this component is to take session characterisations from the QoS manager component and calculate what charge will result. This is then fed less regularly back to the price reaction function which re-calculates the target QoS policy to apply back to the QoS controller. Optionally (shown dashed) the charge can also be compared (reconciled) with the charge advice of the provider using the correlation function.

The charge is only calculated as often as the QoS policy needs to be re-calculated, or also whenever the provider calculates the charge, if reconciliation is enabled. In the meantime, the aggregation building block buffers the input, accumulating parameters together as appropriate.

Technical parameters declared or referred to in the offer acceptance (OA) are used to configure a large part of the charging and accounting component, as well as the engineering aspects of the QoS manager. For instance:

- The aggregation period is determined from the offer acceptance, which declares how often charge advice will be repeated. Aggregation period also depends on the

©Copyright 2000, the Members of the M3I Consortium

frequency of QoS policy re-calculation which is controlled from the price reaction policy (e.g. the kappa time constant from congestion pricing theory).
- The address of the provider's accounting system will be in the offer acceptance if it is intended to be possible to send charge advice for reconciliation. This will be used to configure the distributor with the correct remote address.
- The session scope and meter granularity are configured into the classifier and meter by reflecting on the tariff interface, which is referred to in the offer acceptance (see Section 4.2.3).

All this is achieved with a set of transformations that convert the information formats in the offer acceptance into suitable policy formats to configure each building block. This set of transformations is itself shown as a new service component (Cfg), which may be implemented separately from the charging and accounting system in other distributed scenarios.

Note that an optional transformation can be interposed between the mini charging and accounting component and the meter system. This allows the two components to be sourced from different suppliers, who might use different session characterisation (usage record) formats. Various suitable transformations might be supplied with the mini charging component, to be activated if required at installation time.

Examples of compositions that include this mini charging and accounting service component are given in [31]. Examples are given for both intserv and congestion marking as the QoS technology.

6.1.4 Mediation service component

It has already been hinted that a full, provider-grade charging and accounting system involves considerable distribution in the design. Many different distribution options are discussed in the M3I charging and accounting system design [41]. Therefore, we now introduce a service component that can be included in any of these designs to ease their distribution. The mediation service component shown in Fig 39 takes care of most of the routine technicalities of a charging and accounting system, both during operation and configuration, only leaving the skilled task of setting configuration and enterprise policy to operate it.

A number of mediation systems are available in the market, all of which tend to provide these functions as a minimum. But what we mean architecturally by a mediation *service component* is defined by the building blocks we describe here, *not* by any particular choice of product features in the market.

Our mediation service component operates on a stream of management messages, M. Typically these will be accounting records (session characterisations), but it is also common to mediate charge records. It is even feasible to distribute other management message types, such as context change messages or policies via this mediation component. The component offers five related functions for processing these messages:

- Transformation of the format of incoming messages (e.g. M_1 to M_2 in the figure). This allows heterogeneous standards for detail record messages to be supported.
- Aggregation of messages, both by buffering then accumulating in time within a session, and by aggregation across sessions (see Section 4.1.8). This reduces the

©Copyright 2000, the Members of the M3I Consortium

downstream message traffic load compared to the upstream source load (e.g. M_2 to M_3).

- Storage management of records to protect against system failure (also see Section 4.1.8). The skill in configuring this component is to get the balance right between hard and volatile storage, storage space requirements, throughput and disaster recovery protection.
- Distribution of messages (e.g. M2 is passed straight to storage management, messages of type M3 go to two destinations, while those of type M4 only go to one).
- Automated configuration of both the internals of the mediation component and of related external components. For this, we re-use the configuration component (Cfg) already introduced for the mini charging and accounting system above). For instance, the mediation system might be configured to poll an external meter system. But alternatively, it might configure the external meter system with the required frequency of reports and give it the address to which to send the regular reports.

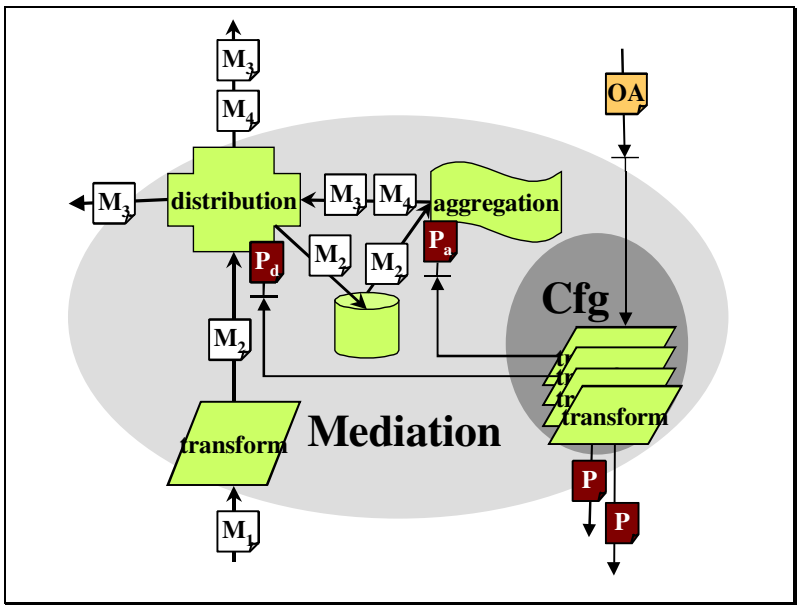


Fig 39 - Mediation service component

6.1.5 Charging and accounting system

Having defined the mediation component, we can now show the architecture of a provider-grade charging and accounting system, CA_p. This system architecture would be suitable for any of the provider-side charging and accounting systems in any of the use cases in Section 2.3 based on the scenarios in the M3I requirements specification [14]. Comprehensive details on the design of charging and accounting systems can be found in the M3I charging and accounting system design [41]. We call it a system, rather than a service component, as it is most likely distributed. However, it is feasible in some scenarios that all parts of it would reside on a single machine (see [41]), in which case service component would be a valid description.

Fig 40 shows the charging and accounting system in context with a metering system component. From this figure it can be seen that this charging and accounting system mostly consists of mediation components. In fact, it is effectively very similar to the mini charging and accounting service component described earlier. However, the whole internal layout of the mini component was optimised for working on a single host,

©Copyright 2000, the Members of the M3I Consortium

whereas this architecture is designed to defend against the vagaries of system distribution, particularly against partial failure (by regularly copying to hard storage) and against the performance penalty of remote message passing.

The rationale for the internal design is no different to that given for the mini charging and accounting system in section 6.1.3. In particular, that section should be referred to for discussion on the configuration component. Section 6.1.4 should be referred to for discussion of the mediation component.

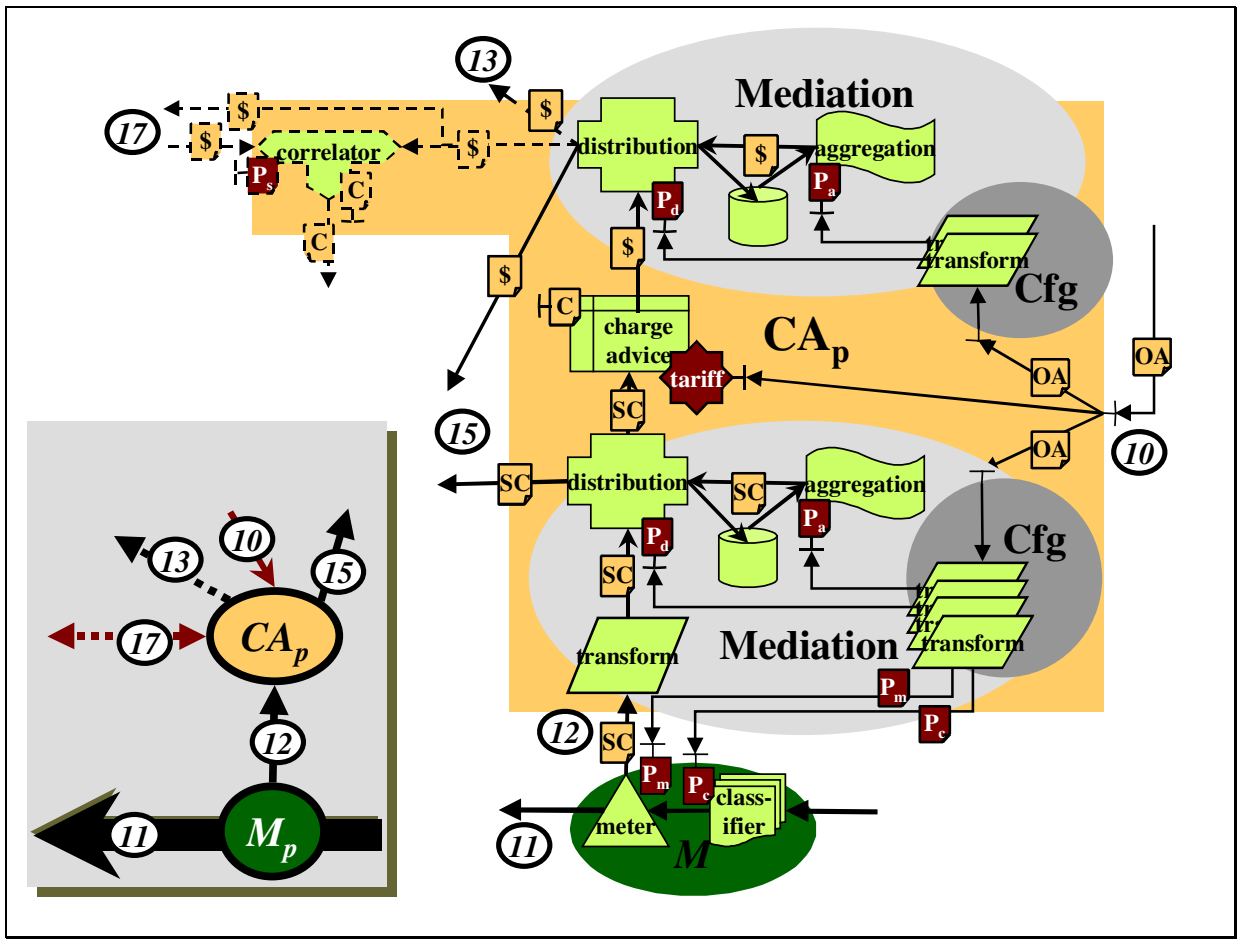


Fig 40 - Charging and accounting service component

As with the mini charging and accounting system, for comparison the inset bottom left of Fig 40 shows the use case sub-systems that map to these components as they are shown in Fig 2 - Fig 8. The rest of the figure mirrors the same layout as the inset, with the use case step numbers repeated against the relevant building block interfaces for ease of reference. Note that the detail of both the technical configuration interfaces, and the context interfaces was not discussed in the use cases.

The interfaces mapping to steps 13 and 17 are both shown dashed, showing that one or the other, but not both should be present. Some use cases had CA_p delivering charge advice directly to the customer's price reaction sub-system (step 13). Others reconciled charge records with the customer's own charging and accounting system (17), which allowed charge advice to be delegated to that system. Either option is possible, but both are not necessary together.

©Copyright 2000, the Members of the M3I Consortium

Note that both usage records and charge records are delivered to the price setting function (step 15). In some scenarios one or the other would be sufficient (see discussion in [35] and Section 4.2.7).

This charging and accounting system layout has been targeted as suitable for the provider side, CA_p. In fact, in corporate environments where a distributed system is required it is just as suitable for the customer-side, CA_c. All that is different is that charge records are always reported to the price reaction function (13) instead of the price setting function (15).

6.2 Scenario compositions

We have now defined components to allow us to build a larger system. In particular, we have components for a mini charging and accounting system for end-customers (if required) and a provider-grade charging and accounting system.

6.2.1 Dynamic price handler with explicit congestion notification (DPH/ECN)

We now show all the building blocks and components to instantiate one of the more general use cases - the dynamic price handler (Fig 41). By comparing this with the use case from which it is derived, it can be seen that the parts of the system beneath policy control (which term includes offer handling) form the bulk of the *architectural* complexity. The policy control elements that are within the M3I infrastructure are primarily directory based. Applications, middleware and agents outside the M3I boundary may become arbitrarily more complex and sophisticated in the future, but this should not require a change to the underlying infrastructure.

The only components we have been unable to show without their internal building blocks in Fig 41 are the two charging and accounting systems. The internals of each of these are shown in Fig 38 & Fig 40. This proves that a large part of the architectural complexity is in these two systems[18]. We will now briefly discuss the main points about this figure.

Firstly, the circled numbers are included as reminders of the steps taken through the original use case defined in Section 2.3.1, which will need to be kept in mind in conjunction with this discussion. Secondly, it should be noted that the same caveat applies to this diagram as did to the original use case discussion - configuration details are confined to those directly concerning market control. Technical configuration is omitted for clarity. The internals of the charging and accounting systems implement most of this technical configuration, which has already been discussed at length in the section above on components.

On the customer side, the only new information is the enterprise policy agent shown with its supporting directory of associations between tasks and price reaction policies. This is as introduced in Section 4.2.11. The rest of the customer side is exactly as in Fig 38, but with less detail.

On the boundary between the customer and provider, the offer directory has been instantiated as two directories. One for the offers from the provider and one for offer acceptances from customers. The rationale is that the offer directory may be a global

[18] However, in other scenarios, the price setting would not be centralised, leading to this being another focus of architectural complexity.

©Copyright 2000, the Members of the M3I Consortium

directory (or set of directories) including offers from many providers. However, the offer acceptance directory is specific to one provider, and contains at least one acceptance per customer (or zero if an offer can be accepted implicitly by use of service). Note that, in this scenario, the offer acceptance directory doesn't include copies of the original offers. Rather, offers are included by reference. Thus, when pricing is changed in the original offer, it must be clear by version control whether this implies the price of the accepted has changed or whether the original price still applies. Both are valid options, depending on the contract. The former case requires a mechanism for customers to be notified of the price change. Price communication is discussed at length in [35].

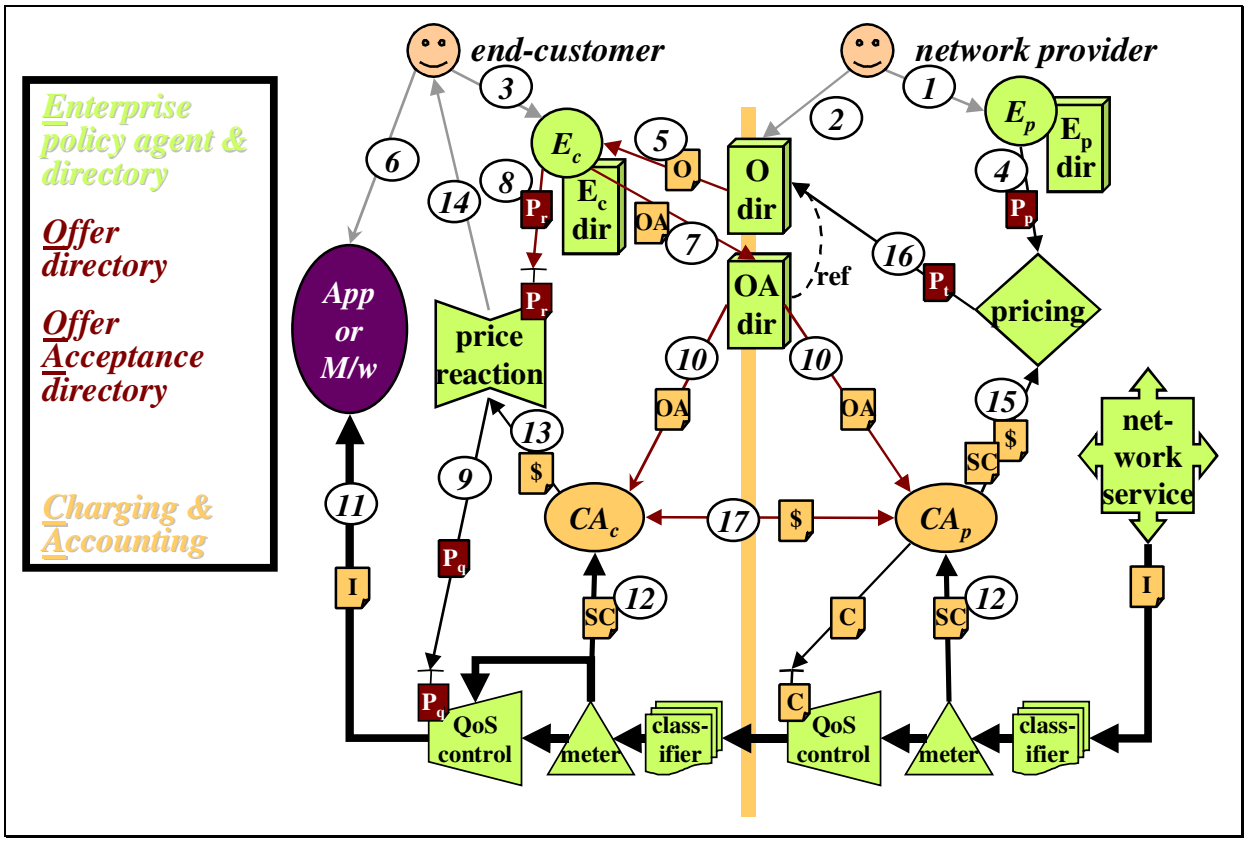


Fig 41 - Dynamic price handler composition

On the provider side, the charging and accounting system is much as already discussed above. The only addition is the interface between the charging and accounting system and provider QoS control. This merely shows the access control mode of the QoS control building block. If an event occurs that causes the provider to, for instance, lose trust in a customer, the context change in the charging system will trigger an access control action here.

Moving up the figure, note that a centralised pricing function is used in this scenario. It is assumed that congestion marking algorithms are identical on all this provider's routers and the price per mark is the same across the whole of the provider's domain. This function merely sets this price per mark. Edge pricing is assumed to include the interconnect charging of onward network providers. Whether this is a sound business scenario is for investigation within the project.

Finally, the provider's enterprise policy agent is essentially a graphical user interface that allows the provider to enter new tariffs and their associated price setting policies

©Copyright 2000, the Members of the M3I Consortium

for storage in the supporting directory. As already discussed, initially this agent is not expected to contain much, if any, intelligence. Rather it is designed to capture the intelligence of the marketing department of the provider.

6.2.2 Other scenarios

Future issues of this document will include building block/component diagrams for each of the M3I requirements scenarios, initially focusing on those to be built and/or tested in the project. Further scenarios are defined in the requirements document that are purely to test the architecture 'on paper'. Diagrams for each of these will also be included in future issues.

However, the DPH/ECN scenario above is a strong guide to the instantiation of many of the other use cases in Section 2.3. In general, the use case sub-systems map directly to the components and building blocks listed against them in Table 12. However, such an approach cannot be applied blindly. There is much work still to do, particularly in understanding the architectural aspects of the various price setting distribution options. Also, the architecture for offer acceptance has to be very different where the offer is tailored to each service invocation, for instance in the scenarios where service is invoked in response to an auction using RSVP.

6.3 Deployment engineering

Future issues of this document will include details of the architectural aspects of the engineering deployment of the various building blocks and components used for each scenario. The approach discussed in Section 2.8 will be used, with diagrams supported by tables specifying deployment granularity, role granularity and event granularity.

7 Limitations & further work

An architecture not only helps the mind form higher level concepts, it invariably limits the mind to exclude concepts that may be valid, but don't fit the architecture. The approach taken has been deliberately designed to allow building blocks to be composed in any valid way imaginable. However, if the building blocks are too coarse, or if the composition approach is too limiting, the architecture will not stand the test of time.

7.1 Status

Although informed by background work from all partners, this architecture is the first release. It was planned to work towards a first internal issue in the first 6 months of the M3I project to allow design work to proceed. However, it is also recognised that, as the project proceeds, new understanding will need to be fed back into the architecture. Hence the first public release is scheduled for near the end of the last year of the project. If the architecture proves not to need major changes, this date may be brought forward.

Before listing some of the specific known issues, we must note that the whole architectural approach is experimental, having been invented on the fly to deal with the problem of reconciling fundamentally disparate architectures. For instance, the functions in a traffic shaper that sit in one box controlled by one stakeholder in the diffserv architecture are spread across a whole network path in the sample path

©Copyright 2000, the Members of the M3I Consortium

shadow pricing approach, where they are co-operatively controlled by all stakeholders. Not only did we have to encompass wildly differing network control architectures, but also radically divergent charging proposals. In the TUD/ETH embedded RSVP charging approach, pricing decisions and charging are distributed across routers, while in one incarnation of BT's diffchar approach, charging primarily runs on customer hosts, with pricing decisions taken centrally, but modified to local conditions on each router. Therefore, there is a risk that the approach will prove *too* flexible, and that instead we will have to adopt a more traditional, less demanding approach that ties down some of the flexibility.

As well as the above more general issue, the following are particular issues recognised in the current state of our understanding:

- The composition technique needs developing. Currently it allows a scenario to be described succinctly and exactly, but it is less good at defining a class of scenarios, where there is flexibility in multiple dimensions. There may be scope for other dimensions, such as not just granularity by each context, but also each type of context - e.g. not just per flow, but per flow type, not just per application, but per application type etc.
- More exact discussion is required on interaction modes. There may be scope for this becoming another dimension of composition.
- Further work is required to define the rest of the M3I scenarios, e.g. those concerning bandwidth auctions and multicast.
- The scenarios have always been described one interface at a time. They need exercising as a whole in order to check for latent undesirable interactions.
- It is impossible for the network provider to identify an application sessions when the traffic is encrypted. If different session will be paid for by different parties, it may not be sufficient for the users to be the only parties who can say which traffic is in which session.
- The price setting building block requires further development. It may need to be two different building blocks for internal and external price setting. Better understanding of the distribution of the price setting function is also required.
- Security concerns are considered throughout the architecture, both in terms of message security and structural security (e.g. duplication of functions across parties with opposing incentives). However a formal security analysis has not been undertaken. To be done properly, engineering and technology choices would have to be defined, so that the impact of underlying system security could be taken into account.
- The role of the risk broker is not clear for sent traffic.

8 Conclusions

An architecture for a market managed multi-service Internet has been defined. As few assumptions as possible have been made on the business models it must support. Commercial openness has been achieved by abstracting the market control and charging aspects away from the network infrastructure. The only assumption this requires is that flexibility and commercial uniqueness are more important than technical efficiency. However, it is intended that technical efficiency will remain as high as is possible without sacrificing flexibility.

©Copyright 2000, the Members of the M3I Consortium

The tensions between requirements identified in the M3I requirements specification have mostly been addressed by allowing customer choice wherever a tension exists. Further work, particularly that identified in the previous section is necessary. But the building blocks and interfaces between them that have been defined so far form a self-consistent set with none dangling, other than those deliberately intended to be for human input only. Thus, we are a step closer to the goal of self-managing the Internet with a market. The wider goal of the project is to assess whether admission can be controlled by the end systems through a market mechanism, rather than directly by the network. This architecture brings the project a step closer to being able to investigate this subject.

9 Acknowledgements

Many contributors to the discussions in the M3I consortium, but especially Martin Karsten (TUD), Burkhard Stiller (ETH), Panayotis Antoniadis (AUEB), Mike Rizzo, Jérôme Tassel, Kostas Damianakis, Steve Rudkin, Ben Strulo, Trevor Burbridge, Gabriele Corliano, Kennedy Cheng and Francesco Manganotti (BT)

10 References

[1] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, L. Zhang, "Recommendations on Queue Management and Congestion Avoidance in the Internet", IETF RFC 2309, Apr 1998 <URL:rfc2309.txt>

[2] K. Nichols (cisco), S. Blake (Torrent), F. Baker (cisco), D. Black (EMC), "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers" IETF RFC 2474, Dec 1998, <URL:rfc2474.txt>

[3] R. Braden, Ed., L. Zhang, S. Berson, S. Herzog, S. Jamin, "Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification". IETF RFC 2205, Sep 1997, <URL:rfc2205.txt>

[4] Quality of Service Routing (qosr) IETF working group charter <URL:http://www.ietf.org/html.charters/qosr-charter.html>

[5] N. Brownlee, C. Mills, G. Ruth, "Traffic Flow Measurement: Architecture", IETF RFC 2063, Jan 1997 <URL:rfc2063.txt>

[6] R. Yavatkar, D. Pendarakis, R. Guerin, " A Framework for Policy-based Admission Control", IETF RFC 2063, Jan 1997 <URL:rfc2753.txt>

[7] Amy R Greenwald, Jeffrey O Kephart and Gerald J Tesauro (IBM), "Strategic Pricebot Dynamics", In Proc ACM E-Commerce, EC'99, 3-5 Nov 1999, <URL: http://www.ibm.com/iac/ec99/>

[8] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, " Hypertext Transfer Protocol -- HTTP/1.1", IETF RFC 2616, Jun 1999, <URL:rfc2616.txt>

©Copyright 2000, the Members of the M3I Consortium

[9] Mark Handley (ACIRI), Colin Perkins & Edmund Whelan (UCL), "Session Announcement Protocol", IETF Internet draft - work in progress, Mar 2000, <URL:draft-ietf-mmusic-sap-v2-06.txt>

[10] M. Handley, H. Schulzrinne, E. Schooler, J. Rosenberg , " SIP: Session Initiation Protocol", IETF RFC 2543, Mar 1999, <URL:rfc2543.txt>

[11] Mark Handley, Van Jacobsen, "SDP: Session Description Protocol", IETF RFC 2327, Mar 1998, <URL:rfc2327.txt>

[12] N. Freed & N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", IETF RFC 2046, Mar 1998, <URL:rfc2046.txt>

[13] Simple Mail Transport Protocol (SMTP) - various IETF RFCs

[14] Ragnar Andreassen (ed) (Telenor R&D), "Requirements specifications; Reference model", M3I Eu VthFramework Project IST-1999-11429, Deliverable 1, Jul 2000, <URL:http://www.m3i.org/private/>

[15] Vesna Hassler (TU Vienna), " X.500 and LDAP Security: A Comparative Overview ", IEEE Network, Vol 13, No.6, pp54--64, 1999.

[16] Mark Handley (UCL), "On Scalable Internet Multimedia Conferencing Systems", PhD thesis (14 Nov 1997) <URL:http://www.aciri.org/mjh/thesis.ps.gz>

[17] W. Yeong, T. Howes, S. Kill, "Lightweight Directory Access Protocol" (v2), IETF RFC 1777, Mar 1998, <URL:rfc1777.txt>

[18] The Directory: Overview of Concepts, Models and Service. CCITT Recommendation X.500, 1988.

[19] R. Braden, D.Clark, S.Shenker, "Integrated Services in the Internet architecture: an overview", IETF RFC 1633, Jun 1994. <URL:http://www.isi.edu/div7/rsvp/pub.html>

[20] S. Blake (Torrent), D. Black (EMC), M. Carlson (Sun), E. Davies (Nortel), Z. Wang (Bell Labs Lucent), W. Weiss (Lucent), "An Architecture for Differentiated Services", IETF RFC 2475, Dec 1998 <URL:rfc2475.txt>

[21] Bouch, A., Sasse, M., & DeMeer, H. G (UCL), " Of packets and people: A user-centred approach to Quality of Service", In Proc. IWQoS'00, May 2000, <URL: http://www.cs.ucl.ac.uk/staff/A.Bouch/42-171796908.ps>

[22] Richard J. Gibbens and Frank P. Kelly, "Distributed connection acceptance control for a connectionless network", In Proc. ITC16, Edinburgh, 1999, <URL: http://www.statslab.cam.ac.uk/~frank/dcac.html>

[23] Martin Karsten, Jens Schmitt, Lars Wolf and Ralf Steinmetz (TUD), "An Embedded Charging Approach for RSVP", In Proc IEEE IFIP IWQoS'98, Napa, CA, USA, pp91-100, May 1998 <URL: http://www.kom.e-technik.tu-darmstadt.de/publications/abstracts/KSWS98-1.html>

[24] Bob Briscoe, Mike Rizzo, Jérôme Tassel & Kostas Damianakis, (BT), "Lightweight Policing and Charging for Packet Networks", in Proc. IEEE OpenArch 2000, pp77-87, Tel Aviv, Israel (Mar 2000), <URL: http://www.labs.bt.com/people/briscorj/papers.html#e2char_oa00>

©Copyright 2000, the Members of the M3I Consortium

[25] G Huston (Telstra), "Next steps for the IP QoS Architecture", IAB Internet Draft - work in progress, Jun 2000, <URL:draft-iab-qos-01.txt>

[26] Open Distributed Processing Reference Model (RM-ODP), ISO/IEC 10746-1 to 10746-4 or ITU-T (formerly CCITT) X.901 to X.904. Jan 1995.

[27] Scott Shenker (Xerox PARC), David Clark (MIT), Deborah Estrin (USC/ISI) and Shai Herzog (USC/ISI), 'Pricing in Computer Networks: Reshaping the research agenda', SIGCOMM Computer Communication Review Volume 26, Number 2, Apr 1996, <URL:http://www.statslab.cam.ac.uk/~frank/PRICE/scott.ps>

[28] Bob Briscoe, 'The Direction of Value Flow in Multi-service Connectionless Networks', In Proc, Second International Conference on Telecommunications and Electronic Commerce (ICTEC'99), Nashville, TN, US, (6-8 Oct 1999) <URL:http://www.labs.bt.com/people/briscorj/papers.html#valflow>

[29] Jérôme Tassel, Bob Briscoe, Alan Smith, "An End to End Price-Based QoS Control Component Using Reflective Java", in Proc 4th COST237 workshop "From Multimedia Services to Network Services" (LNCS pub. Springer-Verlag), Lisboa, Portugal (15-19 Dec 1997), <URL:http://www.labs.bt.com/people/briscorj/papers.html#QoteS>

[30] Bob Briscoe, Mike Rizzo, Jérôme Tassel, Kostas Damianakis, "Lightweight Policing and Charging for Packet Networks", in Proc Third IEEE Conference on Open Architectures and Network Programming (OpenArch 2000), pp77-87, Tel Aviv, Israel (26-27 Mar 2000) <URL:http://www.labs.bt.com/people/briscorj/papers.html#e2char_oa00>

[31] Bob Briscoe (ed), "M3I Pricing Mechanism Design; Price Reaction", M3I Eu Vth Framework Project IST-1999-11429, Deliverable 3 Pt II, Jul 2000, <URL:http://www.m3i.org/private/>

[32] K. K. Ramakrishnan (AT&T Labs Research) and Sally Floyd (LBNL), "A Proposal to add Explicit Congestion Notification (ECN) to IP", IETF RFC 2481, Jan 1999 <URL:rfc2481.txt>

[33] Bob Briscoe & Sandy Johnstone (ed) {?}, " M3I Platform technologies", M3I Eu Vth Framework Project IST-1999-11429, Deliverable 0, Jul 2000, <URL:http://www.m3i.org/private/>

[34] Sean McCreary and kc claffy, "How far does the average packet travel on the Internet?", CAIDA, 25 May 1998, <URL:http://www.caida.org/Learn/ASPL/>

[35] Martin Karsten (TUD) (ed), "M3I Pricing Mechanisms (PM); Design" M3I Eu Vth Framework Project IST-1999-11429, Deliverable 3, Jun 2000, <URL:http://www.m3i.org/private/>

[36] Mike Rizzo, Bob Briscoe, Jérôme Tassel and Konstantinos Damianakis, "A Dynamic Pricing Framework to Support a Scalable, Usage-based Charging Model for Packet-switched Networks", in Proc First International Working Conference on Active Networks (IWAN'99) (LNCS 1653 pub. Springer-Verlag), Berlin, Germany (30 Jun -2 Jul 1999) <URL:http://www.labs.bt.com/people/briscorj/papers.html#dynprice>

[37] Dave Songhurst (Lyndewode) (ed), "Charging Communications Networks; From Theory to Practice", Pub: Elsevier, ISBN: 0-444-50275-0

©Copyright 2000, the Members of the M3I Consortium

[38] George Fankhauser, Burkhard Stiller, Christoph Vögtli and Bernhard Plattner (ETH Zürich), "Reservation-based Charging in an Integrated Services Network", INFORMS Telecommunications Conference, FL, 8-11 Mar 1998, <URL:ftp://ftp.tik.ee.ethz.ch/pub/people/stiller/paper/informs98.ps.gz>

[39] Open Settlement Protocol (OSP) ETSI TIPHON Technical Specification 101321 version 1.4.2, Dec 1998, <URL:http://www.etsi.org/>

[40] L. Berger, T. O'Malley, "RSVP Extensions for IPSEC Data Flows", IETF RFC 2207, Sep 1997, <URL:rfc2207.txt>

[41] Burkhard Stiller (ETH-Z) (ed), "M3I Charging and Accounting System (CAS) Design", M3I Eu VthFramework Project IST-1999-11429, Deliverable 4, Jul 2000, <URL:http://www.m3i.org/private/>

[42] Benjamin N Grosof (IBM), Yannis Labrou (Uni Maryland) and Hoi Y Chan (IBM), " A declarative approach to business rules in contracts: Courteous logic programs in XML ", In Proc ACM E-Commerce, EC'99, 3-5 Nov 1999, <URL: http://www.ibm.com/iac/ec99/>

[43] N. Brownlee (Uni Aukland), " SRL: A Language for Describing Traffic Flows and Specifying Actions for Flow Groups", IETF RFC 2723, Sep 1997, <URL:rfc2723.txt>

©Copyright 2000, the Members of the M3I Consortium