# Market Managed Multi-service Internet

# M3I

*European Fifth Framework Project IST-1999-11429*

## *Deliverable 2*

## Architecture Part II

## Construction

**The M3I Consortium**

Hewlett Packard Ltd, Bristol UK (Coordinator)
Athens University of Economics and Business, GR
BTexact Research, Ipswich GB
Eidgenössische Technische Hochschule, Zürich CH
Technical University of Darmstadt, DE
Telenor, Oslo NO
Forschungszentrum Telekommunikation Wien Betriebs-GmbH, AT



*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

## Document Control

| | |
|---|---|
| **Title** | Construction: Construction |
| **Type** | Private deliverable (eventually public) |
| **Authors** | Bob Briscoe |
| **email** | <mailto:bob.briscoe@bt.com> |
| **Origin** | BT Research |
| **Wk package** | 3 |
| **Doc ID** | arch_pt2_m3i.pdf |

## AMENDMENT HISTORY

| Version | Date | Author | Description/Comments |
|---|---|---|---|
| V1.0 | 07 Jul 2000 | Bob Briscoe | First Issue |
| V1.1 | 25 Oct 2001 | Bob Briscoe | Formatting changes & minor edits |
| V1.2 | 28 Oct 2001 | Bob Briscoe | Structural draft: Split into 2 parts |
| V1.3 | 26 Nov 2001 | Bob Briscoe | Draft for review comments on new structure |
| V2.0 | 22 Feb 2002 | Bob Briscoe | Mended logical flow of text, cross-references etc. for 2 part structure; Added citations of recent M3I documents; Fixed errors in description of GSP; Clarified difference between Session Characterisation instance and type; Added lessons on interface definitions, esp. QoS. Added Glossary |
| V2.1 | 27 Aug 2003 | Bob Briscoe | Removed confidentiality markings |



  

# Contents



  

  

  

# 1 Introduction

This document presents our architecture for a market managed multi-service Internet (M3I). As explained in Part I, the task is to encompass all multi-service Internet architectures simultaneously in one architecture. To avoid confusion, we therefore term these sub-architectures 'service plans'. A service plan is a combination of a network control architecture and the business model used to offer it as a service. The architecture is open, not only encompassing current and proposed service plans, but also facilitating the creation of new ones.

This document is an 'architecture' not a 'design'. We define architecture as a specification of:

- why certain functions are best provided separately;
- what service they each offer and what their interfaces are;
- information structures that will have common use across the system such as identifiers.

A design, on the other hand, would specify how the parts should be composed together. Even a high level design would say $A$ uses $B$, not just that $A$ and $B$ should be logically separate. However, although an architecture shouldn't completely specify the 'wiring' between all the modules, it should give guidance on legal and illegal wiring, on where functions are best deployed, on the layering of generic functions beneath less generic ones etc. The M3I architecture therefore specifies building blocks and interfaces, but it also sets out principles to guide the design process of any new service plan.

The M3I Architecture is delivered in two parts: Principles (Part I [8]) and Construction (Part II — this part). Part I is designed to be readable alone. It goes to the core of what a multi-service Internet is and extracts fundamental principles from this exercise. It then gives a high level overview of the building blocks described in this second part in order to describe sensible ways to put them together. In contrast, this second part cannot be read alone — part I is an essential pre-requisite.

This second part describes the construction kit of the architecture — the building blocks and their interfaces. Indeed, it will be seen that the building blocks are very rudimentary — much as the principles were very fundamental. Specification is high level, but relatively precise. Because the building blocks are so rudimentary, we also introduce various compositions of the building blocks into useful sub-systems — themselves building blocks, but molecular rather than atomic. 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, hence the need for the principles in Part I.

Taken as a whole, the architecture achieves the original ambitious goal of openness, encompassing current and proposed service plans, as well as enabling innovative new ones.

## 1.1 Approach

The technique adopted was to identify a small but sufficient set of **service building blocks** from which we could compose all four major sub-systems of the top level architecture, already introduced in part I:

- **network service**
- **market control** of its load
- **application and policy control** of the market (by **human customers or providers**)
- and **charging** within all this.

The diagram of the top level architecture is repeated here as a reminder (Fig 1), but its description is not — the reader should refer to part I.

We found that the **service components** shown in the high level architecture were *not* the most rudimentary service building blocks. Within them, more fundamental building blocks could be put together in various ways to create different types of each component for different scenarios.

How we chose these rudimentary building blocks was a mix of rationality and intuition from experience that would be both difficult and unnecessary to describe. However, it would also be wrong to just present our choices without justification. Some justification is therefore given by running through a selection of use-cases designed to stretch the architecture. These show how each component can be very different in different scenarios,

  

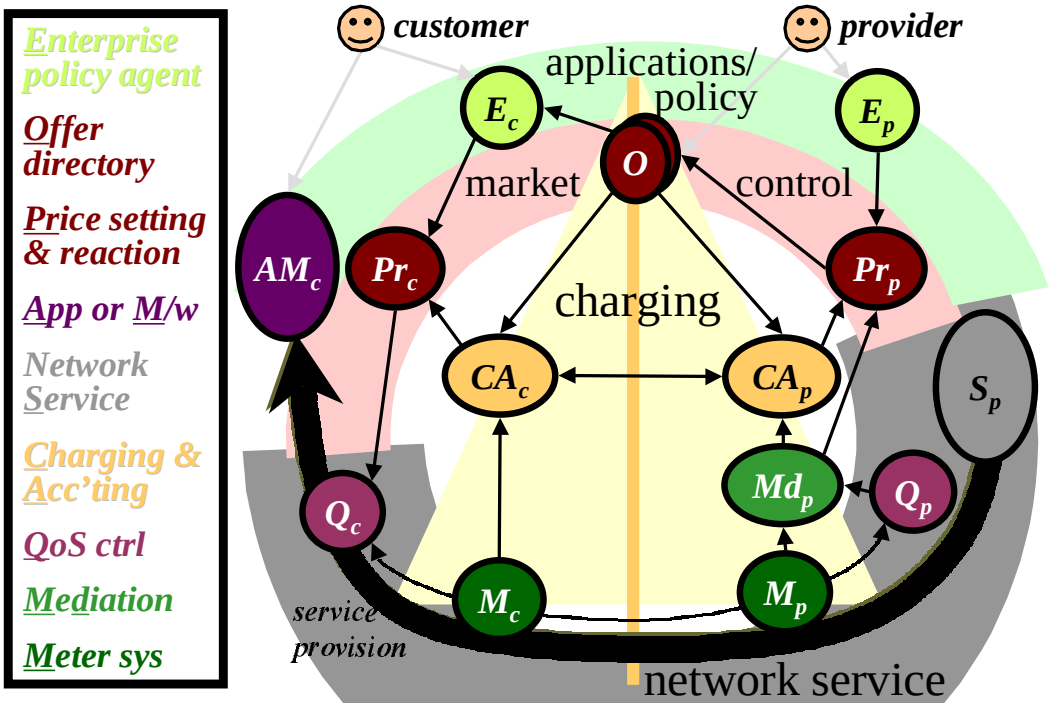


Figure 1: High level architecture

bringing out the underlying functions within each component. We draw scenarios from the M3I requirements specification [2], but we follow an order that builds up one use case from the last, rather than rigidly following the M3I requirements scenarios. Further detail on these scenarios is given in [1]. This also includes use-cases for further scenarios created after this architecture, which gives at least a degree of confidence that the architecture is applicable to new as well as existing scenarios.

Nonetheless, this document primarily reports the result of our work rather than processes like use-case analysis that we used — not the means but the ends. Therefore, having identified the set of rudimentary building blocks, each is defined giving 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. As explained in Part I, 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 (which may be a soft state directory) in which to put each item of configuration information. 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 architecture is primarily a logical one, focusing on enterprise, functional computing and information viewpoints [29]. The approach that will be adopted towards defining the engineering and technology viewpoints is introduced, but this document avoids a systematic definition from these viewpoints, because they are very scenario-dependent. We do give occasional comments where important engineering or technology issues are at stake. Examples of the engineering and technology details of each service component can be found in the design and implementation documents for scenarios investigated in the M3I project: network service and network QoS control [35, 36]; offer dissemination and price setting [35, 36]; customer enterprise policy, price reaction and customer QoS control [11, 30]; provider enterprise policy and charging and accounting [44, 45].

## 1.2 Document structure

As we have said, this second part follows on from the high level overview of the architecture that concluded part I. In that overview, we explained why we always focus on customer-provider interfaces and clarified where M3I technology might be deployed in concrete cases to provide some context. We introduced the four loosely layered sub-systems and ended by focusing on the layer that is strictly outside our infrastructure: application and policy, in order to emphasise how we expect innovation to be enabled in that space. Therefore, the reader should now have a broad idea of the service components of the high level architecture, and the classes of applications that will use them.

In the rest of this document, we first work step-by-step through our selected use cases (Section 2) to help give context and justify our later choices.

The more formal body of the document begins with Section 3 where we formalise our definitions and notation with the help of the glossary in Appendix B.

Section 4 introduces the **service building blocks**. Firstly, we give a brief overview, describing and classifying them. Then we give the definition of each one. Although configuration is necessary before operation in real life, we deliberately organise out explanation the other way round. So we start by identifying those building blocks in use during **operation** of services and the main information structures that continuously pass between them. 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 **application building blocks**, despite being external to the infrastructure.

Section 5 is a systematic classification of all the **interfaces** between pairs of service building blocks, including the external interfaces of the infrastructure to applications. Our approach to defining interfaces is fairly intuitive, but as our work stretches from low level network interfaces to high level application interfaces, compromises in our approach have been necessary, which are explained in Appendix A.1 for completeness. 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 the full set of interface and protocol definitions of the M3I Architecture.

Section 6 starts by defining a few useful compositions of service building blocks. These **service components** can then used to build a complete design to implement a specific service plan, for instance one of the M3I scenarios. There are many techniques available for describing the composition of a distributed system design, but none are particularly good at crossing boundaries between low level embedded systems and high level object-oriented design. Therefore, we provide an overview of the necessary dimensions of a good description of such a composition in Appendix A.2.

We end by highlighting limitations and further work and drawing conclusions in Section 7.

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 at run-time to build their own service plans (using the principles as well, of course). 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 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 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 possible scenarios, some more of which are given in the M3I specification of requirements [2] and the description of M3I scenarios [1].

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** 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 $X_c$.

## 2.1 Edge-centric use case

Fig 2 shows the use-case of configuring and operating a scenario of particular interest to the M3I project, where the network service is monitored and controlled as far as possible from the end-customer's systems.

The scenario is termed the **dynamic price handler** and is described in detail elsewhere [2, 1]. In brief, the provider offers service by simply applying a tiny fixed price to each congestion mark received on a packet (assuming explicit congestion notification — ECN [40]). This gives the receiver the incentive to request the sender to change the sending rate appropriately on per packet time-scales. A variant has the sender paying congestion charges on behalf of the receiver. The payer uses a dynamic price handler agent to determine the sending rate that returns optimum net value, where net value is the difference between the utility and cost of a certain rate of delivery.

The end-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 end-customer 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 2.2).

This use case can be used to describe either half of the **dynamic price handler** scenario — sender or receiver. 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 2.4.2) can then be used for the interface between the two customers ($I5$), while the inter-network use case (Section 2.3) 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 [11].

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 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 [42, 7].

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. 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.

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,** $\mathbf{E_p}$, and it places the offers $(2)$ in a more public **offer directory,** $\mathbf{O}$, 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,** $\mathbf{E_c}$. 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.

The **price setting** function, $\mathbf{Pr_p}$, 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 $\mathbf{AM_c}$ 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 [46] or by being explicitly notified by the application

  

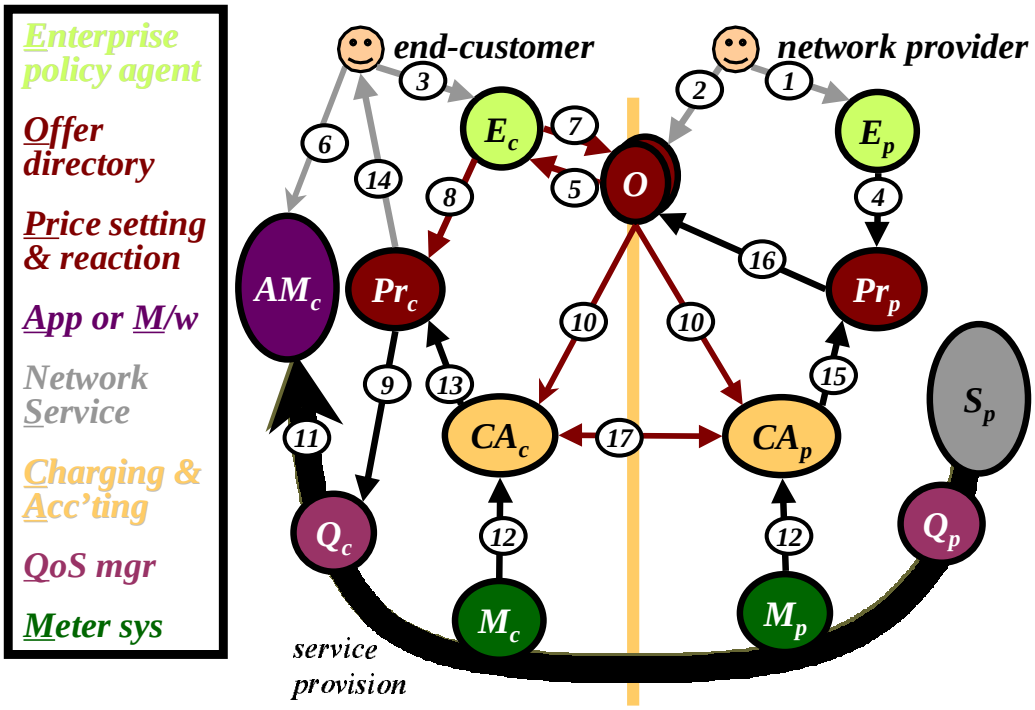


Figure 2: Edge-centric use case

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. At the same time, the agent will configure the **price reaction** function, $\mathbf{Pr_c}$, 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**, $\mathbf{Q_c}$, for this session (9). In parallel to all this, acceptance of the offer triggers configuration of both **charging and accounting** systems, $\mathbf{CA}$, 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**, $\mathbf{Q_c}$, 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**, $\mathbf{Q_p}$. The data load and any QoS requests are **metered**, $\mathbf{M}$, as they pass. Before going into detail, we will briefly explain why, in this edge-centric use-case, both the customer ($\mathbf{M_c}$) as well as the network provider ($\mathbf{M_p}$) meter the load. The customer's readings are fed back through the customers charging system to the price reaction sub-system, 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 [11] 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 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 [10]. 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.

## 2.2 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 end-customer's system, but not monitoring of charges. In order to be able to dynamically react to the level of her charges, the end-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.

  

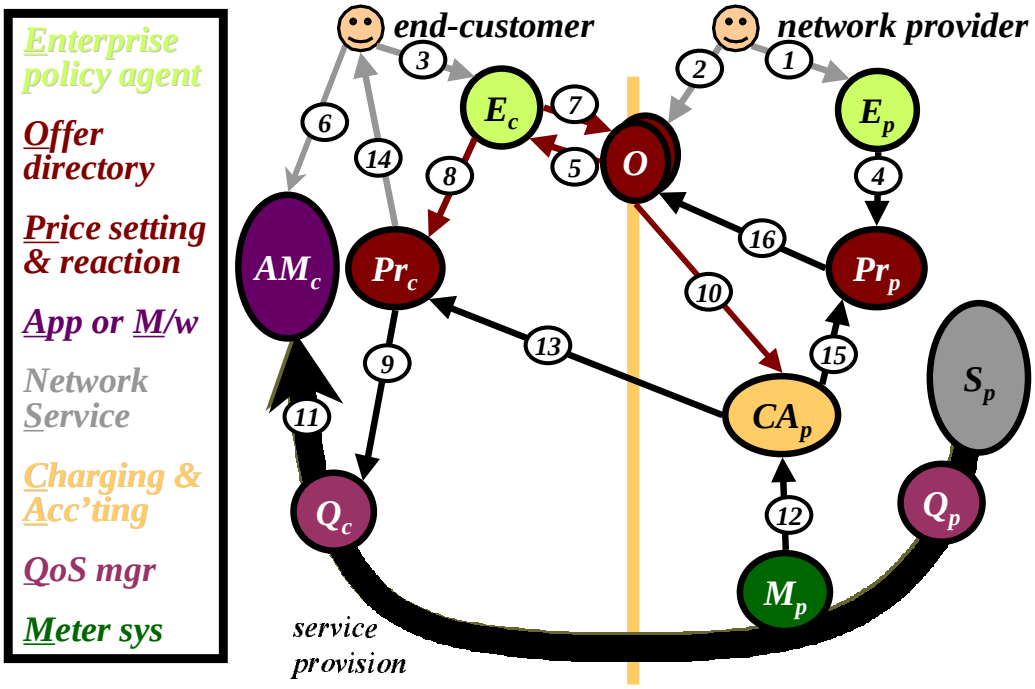


Figure 3: Edge-control use case

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 [2]. 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 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 end-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 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 end-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 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).

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

  

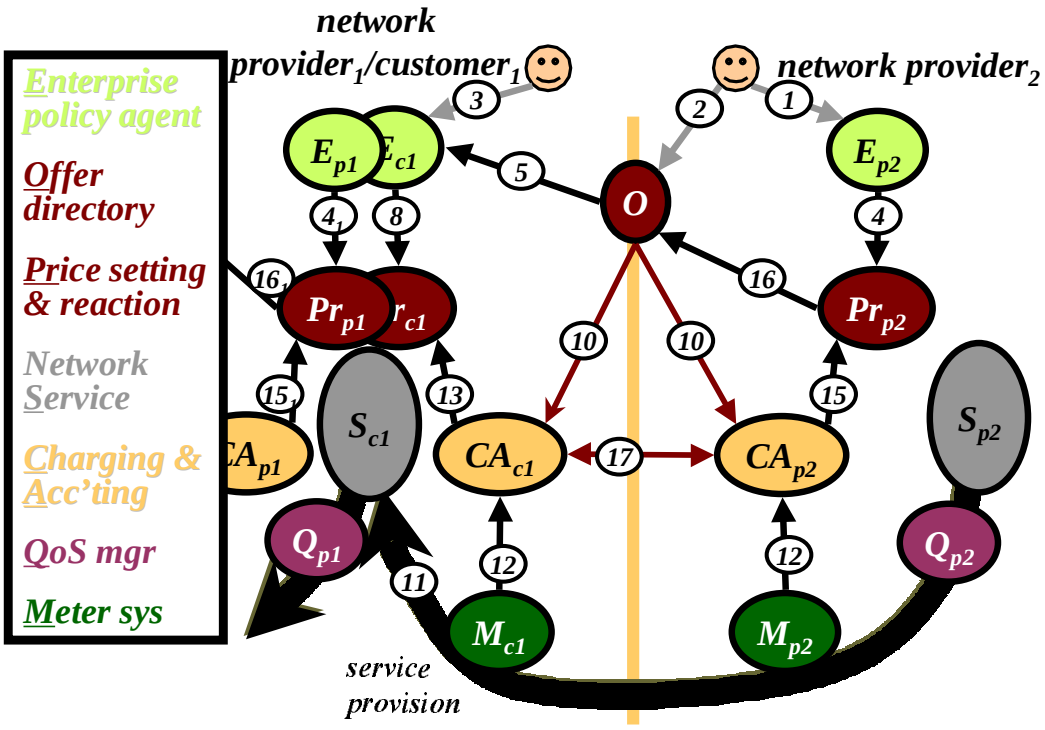


Figure 4: Inter-network use-case

for standardisation of the basis of 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** (Part I [8]).

## 2.4 Guaranteed stream provider use-case

The M3I requirements document [2] 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. This very useful scenario is described in more depth in [1], with a full design given in [36] and results of experiments in [37].

In this document, we attempt to show that it is possible to separate these two roles into stand-alone businesses (this issues is discussed further in [31, Section 6.2]). We also attempt to allow heterogeneity of service propositions by *not* requiring the risk broker to span both ends of a communication.

### 2.4.1 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 of the network provider then of the risk broker and end-customer.

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 [19]. 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, 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, $Pr_{bc}$, $(8)$ 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 [2], 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.

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 $C = aT + bV + c$ [43]. $C$ 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 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.

  

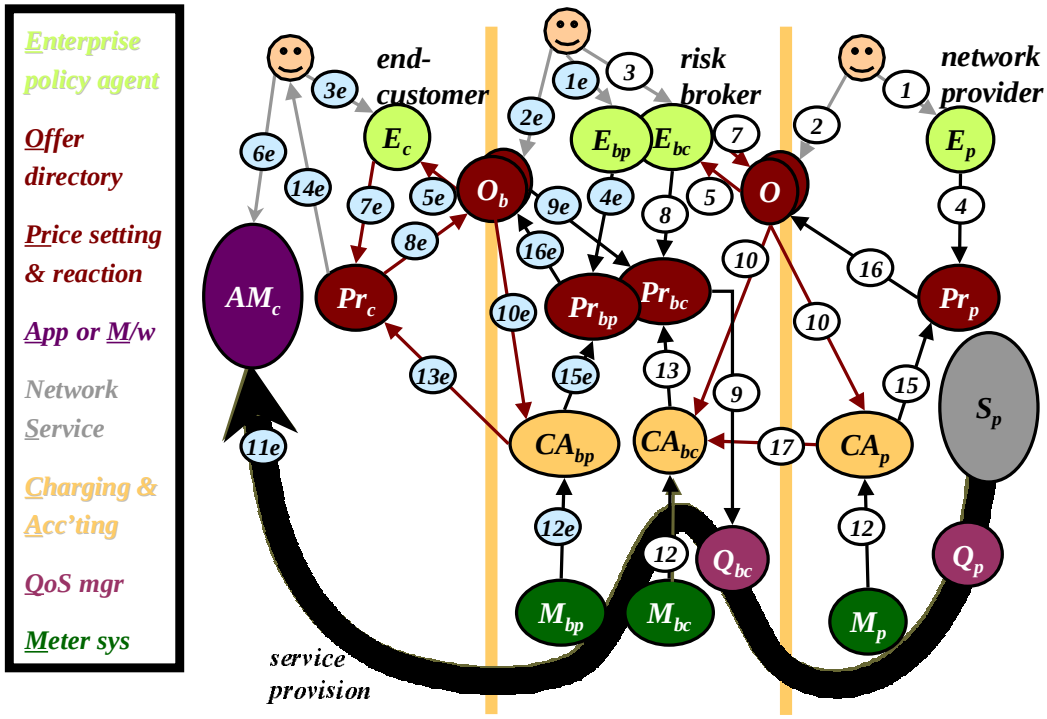


Figure 5: Risk broker use case

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. The direction of the traffic for which a guarantee is required will also have to be made clear, which is provided by the PATH/RESV distinction of RSVP as well.

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 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 copied 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.

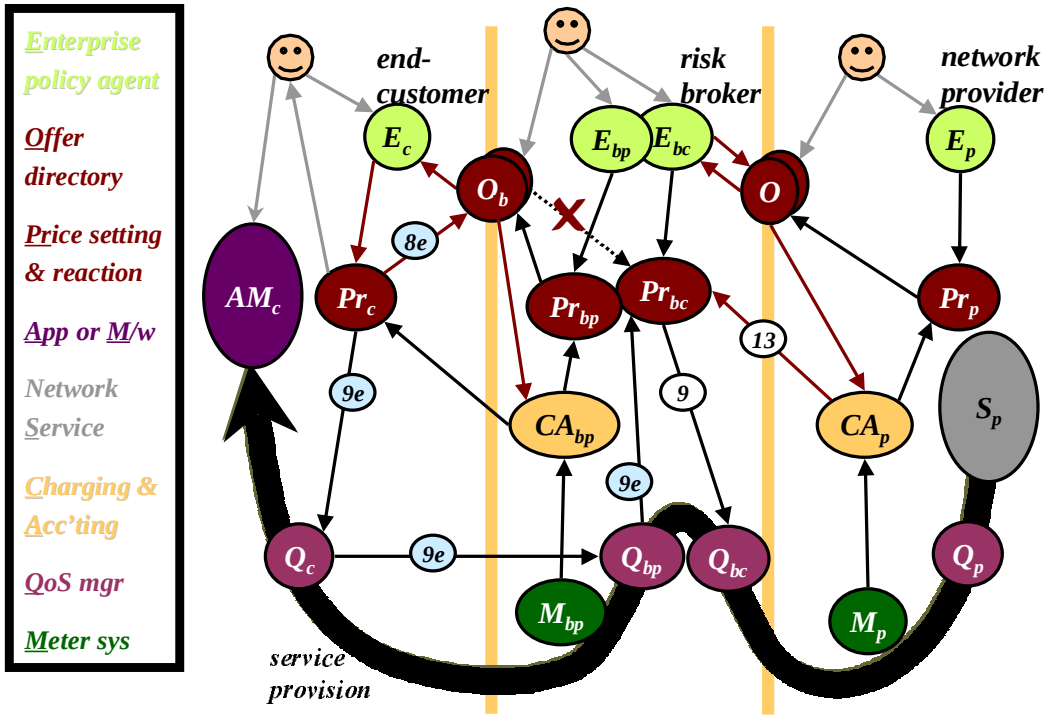


Figure 6: Alternative risk broker use case

No routers in the middle are configured with RSVP processing enabled, so they fall back to forwarding RSVP requests as data. This ensures both (all) ends participate in end to end RSVP signalling as with Intserv, and QoS specifications can be intercepted and passed to the customer's price reactors, $Pr_c$, at all ends. However, routers in the middle avoid the alleged scalability problems of per flow processing against reservations.

If all ends do not support RSVP signalling the QoS specification could be signalled end-to-end 'over the top of' the middle, using session control protocols. However, the QoS specification in the session description still needs to be understood by all ends (just not the 'middles'). 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 (in contrast to the more mainstream M3I scenario where the risk broker is integrated with a QoS gateway, which can implement call admission control to protect itself from periods of high pricing). 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 [11].

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 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.

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.

### 2.4.2 Clearinghouse

The use case described here uses the end-to-end clearing architecture of [7], rather than the embedded clearing of [16] or the centralised clearing of the open settlement protocol [15]. 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 acc-epted 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 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.

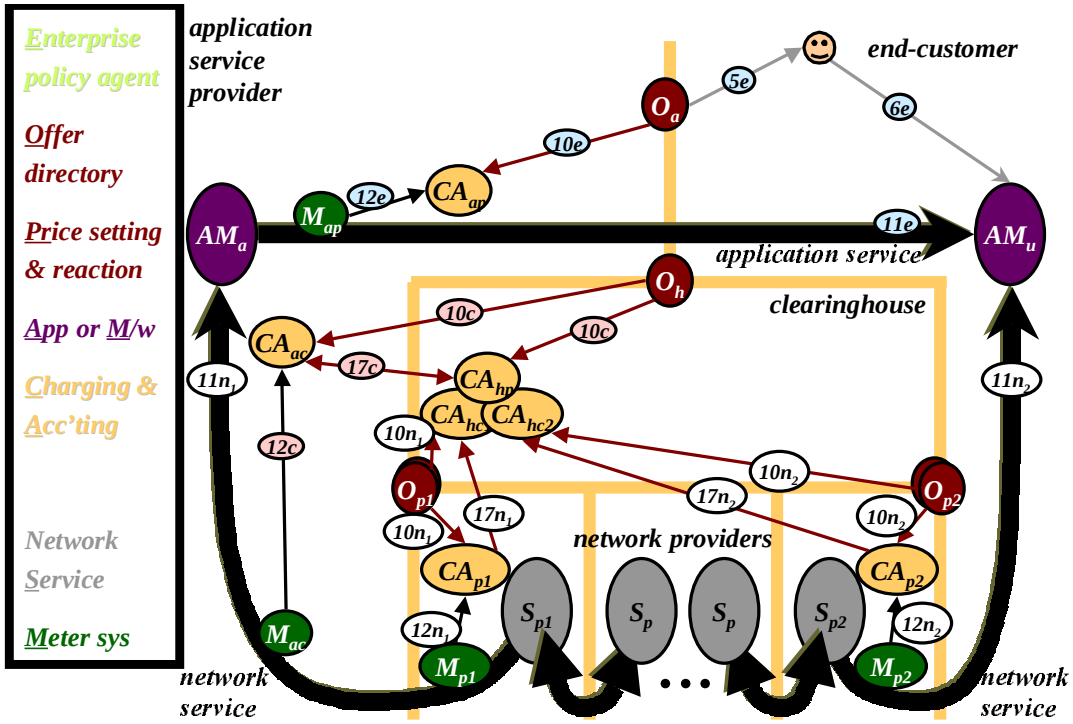


Figure 7: Clearinghouse use case; operations phase

Note that the clearinghouse does no metering itself. It relies on receiving accounting records for each session from the two network providers ($17n_1$ & $17n_2$). 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.

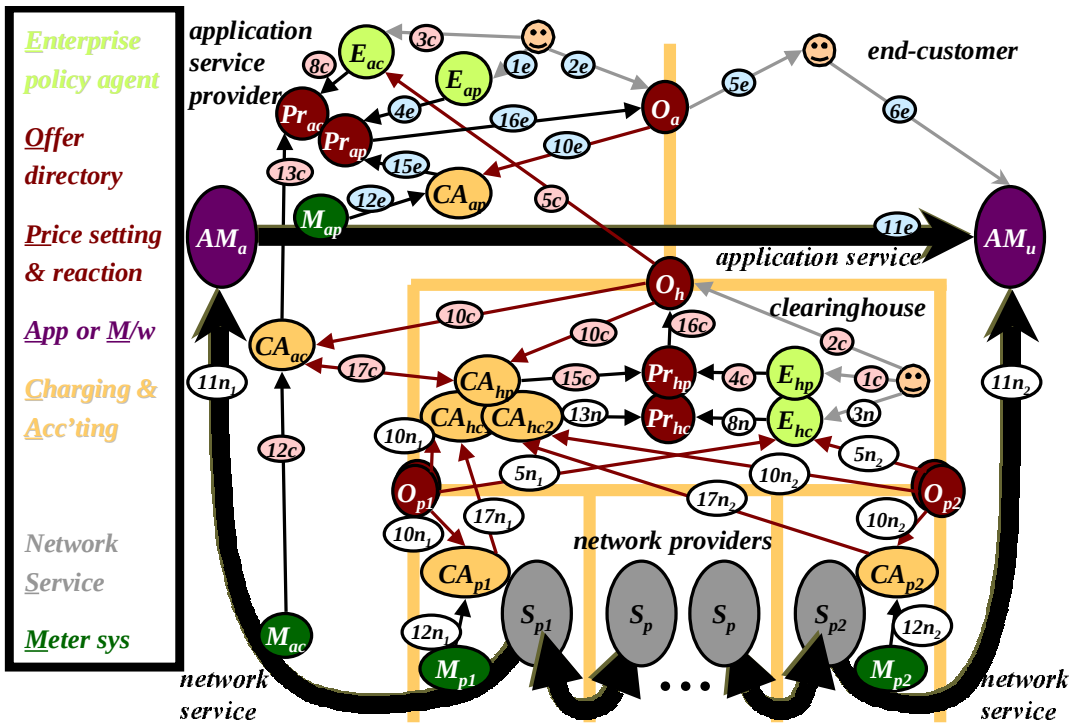


Figure 8: Clearinghouse use case

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 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 [20]. 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).

  

# 3 Definitions

Having introduced the whole approach, we now start the more comprehensive body of this document by defining our terms more precisely. A full glossary of terms is provided at Appendix B.

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 9 and Fig 10 collect together the legends for the rest of the figures in this architecture. The three symbols along the top of Fig 9 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.

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 are not shown for clarity.

The meaning of each symbol for information (or state) in the lower half of Fig 9 will be introduced as we proceed.

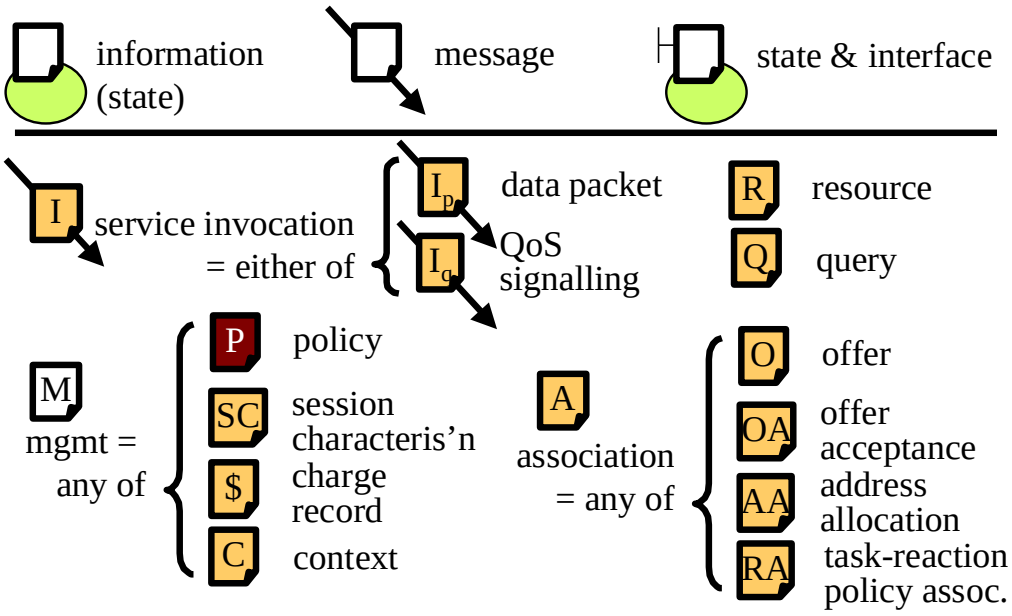


Figure 9: Legend: state

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.

  

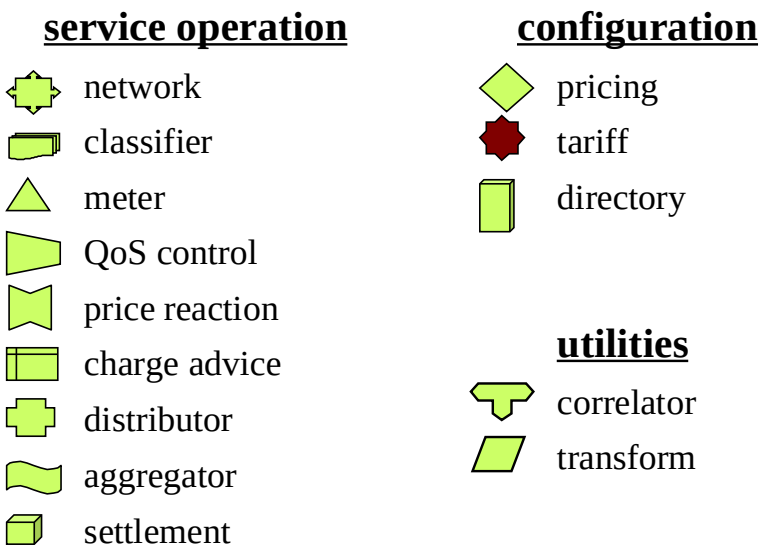


Figure 10: Legend: service building blocks

Fig 10 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:

- 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.

Sometimes, during the configuration phase in particular, it is not state itself that is used, but the *type* of the class of object that would hold the state. Where this is the case, it will be highlighted in the text.

  

# 4 Building blocks

## 4.1 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 [5], diffserv [3], sample path shadow pricing (SPSP) [19], TUD/ETH embedded RSVP charging [34] including by auction, BT diffchar [10], 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. Our aim is to enable this, but also allow more generic approaches.

### 4.1.1 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) [19], and some configurations of BT's diffchar [10]. 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.

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.

### 4.1.2 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 [23] 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 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 4.3.3 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.

## 4.2 Operation building blocks

### 4.2.1 Networking

Perhaps surprisingly, the **networking** building block (Fig 11) 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, $\mathbf{R}$,** namely bandwidth, buffer space, and scheduling priority. The network service is invoked by sending **invocation messages, $\mathbf{I}$,** 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) [6], so in general we will stick to the term invocation.

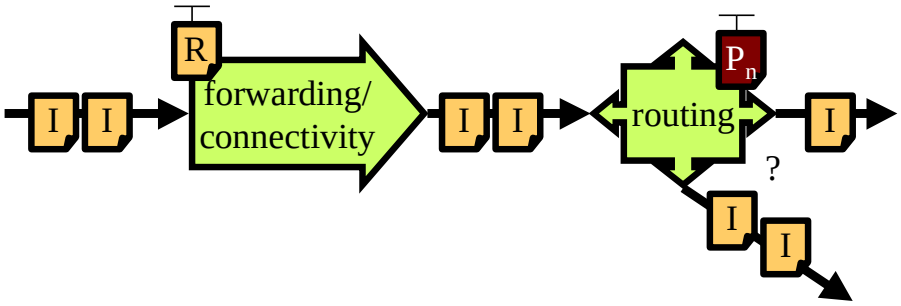


Figure 11: Networking

The routing function simply decides into which output to forward each invocation based on routing **state, $\mathbf{P_n}$,** 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 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 [47] that also takes account of the diffserv code point field [39] 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.

  

### 4.2.2 Classifying

A **classifier** building block (Fig 12) separates a series of invocations into sets based on rules in the **policy,** $\mathbf{P_c}$.

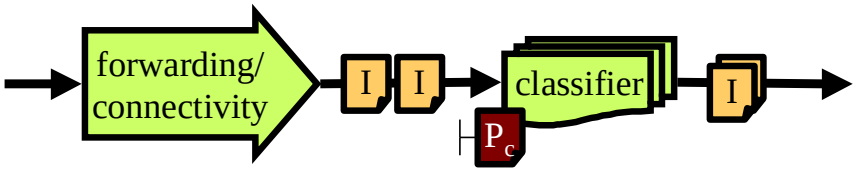


Figure 12: Classifying

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 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.

### 4.2.3 Metering

A **meter** building block (Fig 13) 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,** $\mathbf{P_m}$. 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.

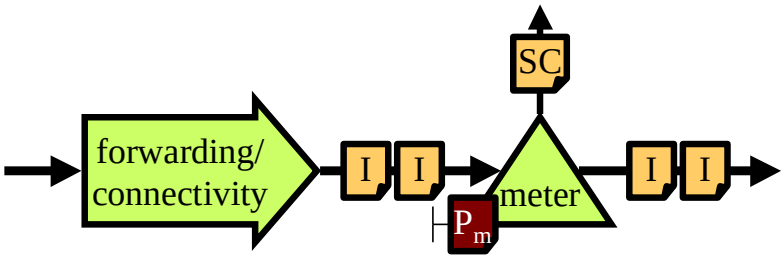


Figure 13: Metering

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 [13]) it might make sense to ensure message reliability, depending on the impact of loss, and the vulnerability to denial of service attacks.

### 4.2.4 QoS, rate, admission or access control

The **QoS control** building block (Fig 14a) 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.

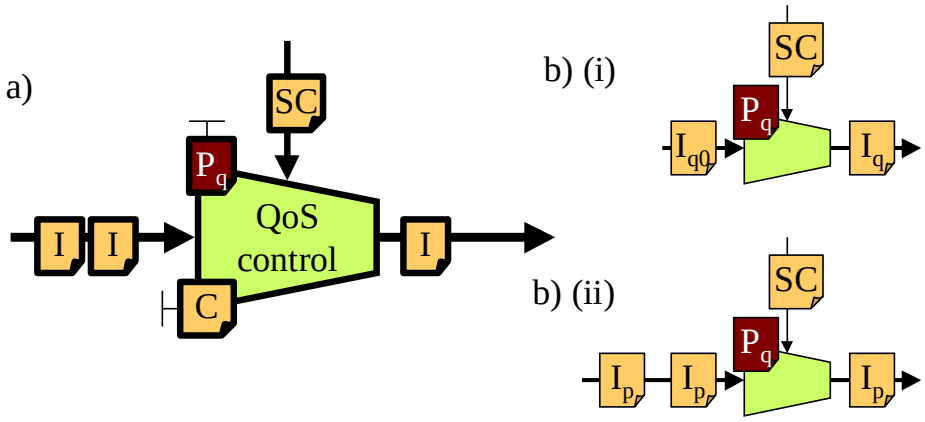


Figure 14: QoS, rate, admission or access control

The nature of the control is open to wide interpretation. It may drop random invocations in order to maintain the desired rate [49] (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, $\mathbf{P_q}$**.

Fig 14b 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 14b(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 [11], 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 14b(ii). The QoS control policy, $P_q$, that controls this building block, may be a static QoS specification (e.g. an 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 [11].

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 [49].

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.

### 4.2.5 Charge advice

The **charge advice** building block (Fig 15) turns the characterisation of a session into advice of how much money would be charged, given a certain **tariff** with certain pricing **policies,** $\mathbf{P_t}$. The result also depends on the **context,** $\mathbf{C}$.

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 4.3.3).

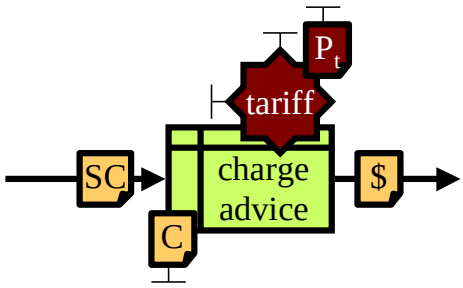


Figure 15: Charge advice

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.

  

### 4.2.6 Price/charge reaction

The **price reaction** service building block outputs the long-term control for the QoS control building blocks — the **QoS control policy, $\mathbf{P_q}$**. It acts on behalf of the customer through the customer's **price reaction policy, $\mathbf{P_r}$**. 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 4.3.11 and in more depth in [11].

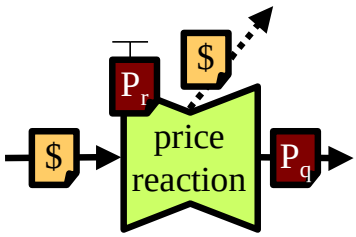


Figure 16: Price/charge reaction

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.

### 4.2.7 Distribution

The **distribution** building block (Fig 17) is broadly semantically similar to a routing element. It operates on **control messages, $\mathbf{M}$**, which are simply defined as any of $P$, $SC$, $C$ or $. **Policy, $\mathbf{P_d}$**, defines to where different types of input messages are forwarded (including to multiple destinations). The policy effectively defines the control management workflow.

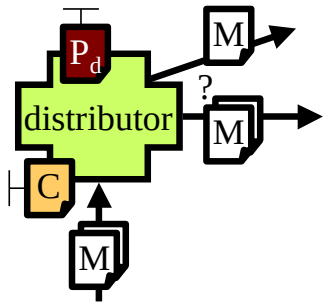


Figure 17: Distribution

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'.

### 4.2.8 Aggregation

The **aggregation** building block (Fig 18), 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, $\mathbf{P_a}$**. In its arbitrary mode, the building block responds to queries about details of past input in order to formulate the requested aggregated reports.

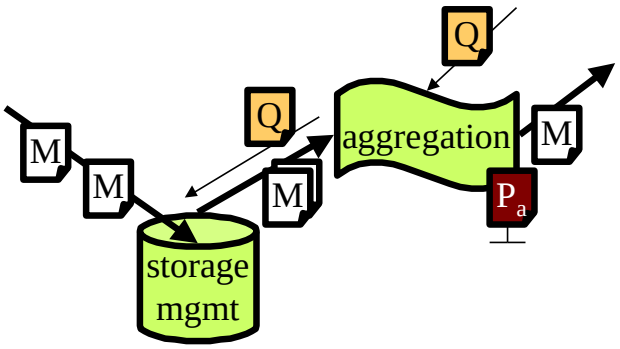


Figure 18: Aggregation

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.

### 4.2.9 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 19) 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.

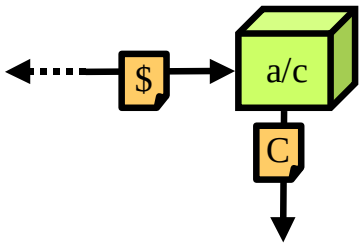


Figure 19: Settlement

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

## 4.3 Configuration building blocks

### 4.3.1 Directory

A heavily used building block for many aspects of configuration is the **directory** (Fig 20). 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.

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 [26].

If a directory is implemented using hard state, it need not be as a centralised database accessed using LDAP [50] or X.500 [14] 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 [23], 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.

  

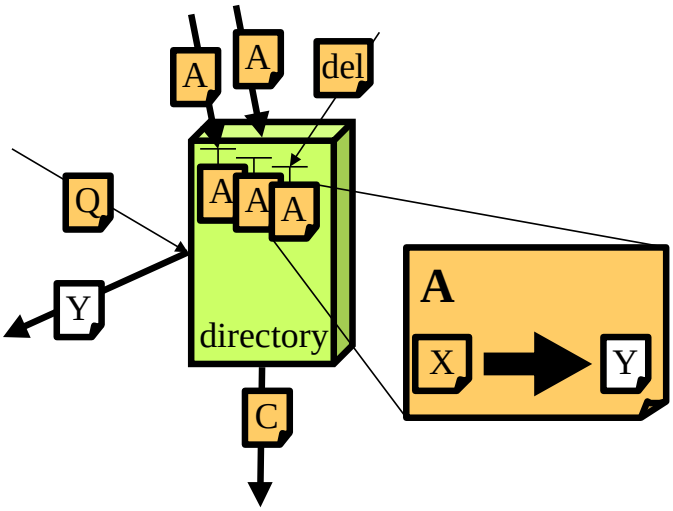


Figure 20: Directory

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.

### 4.3.2 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 21) 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.

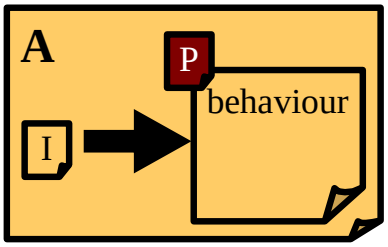


Figure 21: Service definition

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).

### 4.3.3 Tariff

A **tariff** is an algorithm that contains, at its simplest, a vector of **prices,** $\mathbf{P_t}$, to be applied to a vector of session characterisation. Therefore the tariff is specified by the *type* of the session characterisation and the corresponding prices. More complex algorithms with considerable conditional semantics are also common. See Section 4.2.5, Charge advice.

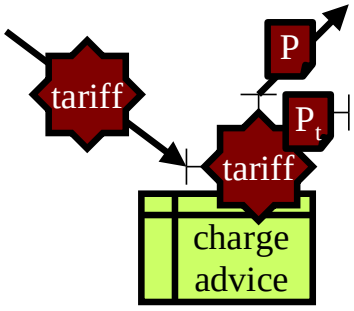


Figure 22: Tariff configuration

As well as setting the prices in the tariff (Section 4.3.7), 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 22. 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) in order to extract the type signature[41]. 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.

### 4.3.4 Offer

An **offer, O** (Fig 23), 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 [21]. 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 Part I [8].

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

- Description

  

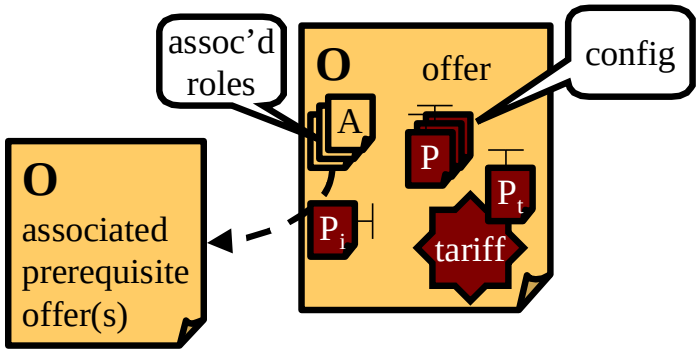


Figure 23: Offer

- 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) [24] 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, $\mathbf{P}$**, 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, $\mathbf{A}$**, 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 session. Roles nominated might include services such as the clearinghouse, the charging function provider etc. [2], 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).

The pricing of the tariff would be set by **pricing policies** $\mathbf{P_t}$. 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.

### 4.3.5 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 20 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 [17] & SMTP [48] use MIME payloads [18], SAP [25] & SIP [22] use SDP [24], RSVP [6] defines its own payload extension capabilities).

### 4.3.6 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).

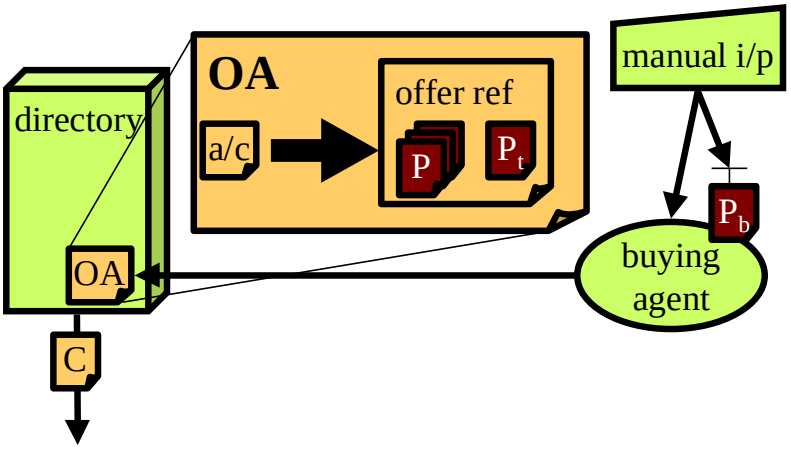


Figure 24: Offer acceptance

Fig 24 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,** $\mathbf{P_b}$, 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,** $\mathbf{A}$, 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 $\mathbf{P}$, which set the customer **context,** $\mathbf{C}$ output from the directory.

### 4.3.7 Price setting

The role of the **price setting** building block (Fig 25a) 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.

There are two different classes of price setting function:

  

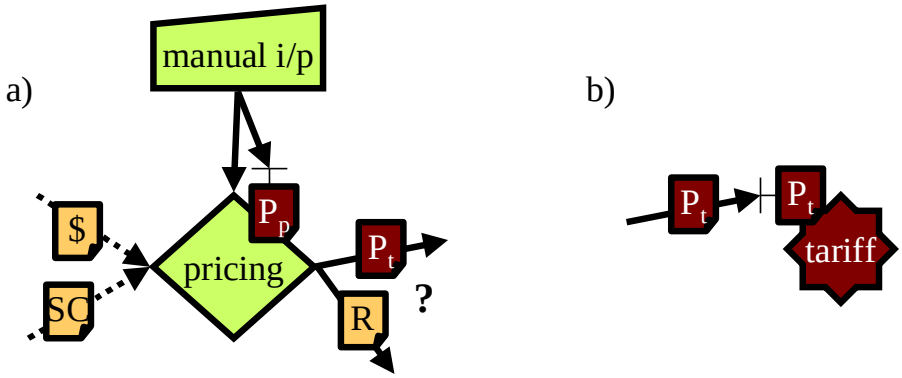


Figure 25: Price setting

- 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, $\mathbf{SC}$. 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,** $\mathbf{P_t}$, 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,** $\mathbf{R}$). 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,** $\mathbf{P_p}$, 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 [20]. 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.

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 25b 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.

### 4.3.8 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 26). Note that we can use this association for allocating any name or address, not just network addresses.

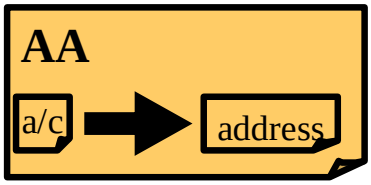


Figure 26: Address allocation

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.

### 4.3.9 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.

In Fig 27 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.

### 4.3.10 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 [5], this is directly under the control of the application author, which makes considerable sense. However, with the initial differentiated services proposal [3], 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 [4]. 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 28.

  

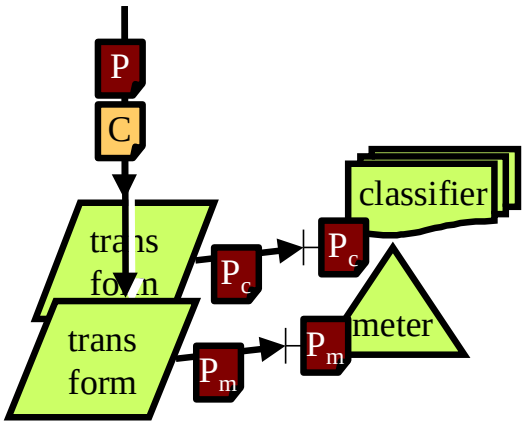


Figure 27: Classifier and meter configuration

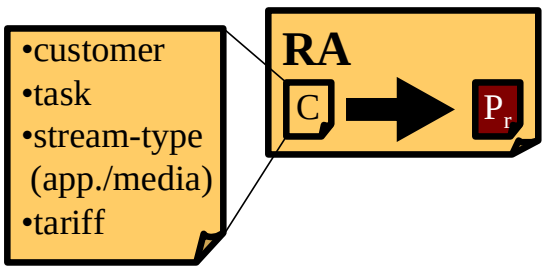


Figure 28: Task-reaction policy association

  

The **context,** $\mathbf{C}$, 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,** $\mathbf{P_r}$.

This **association,** $\mathbf{A}$, 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.

### 4.3.11 Price reaction policy configuration

Assuming the repository of task-reaction policy associations introduced above, Fig 29 shows the **price reaction** building block being configured with the **policy,** $\mathbf{P_r}$, relevant to a particular **context,** $\mathbf{C}$, by looking up the **task-reaction policy association,** $\mathbf{RA}$, between the two in a **directory**.

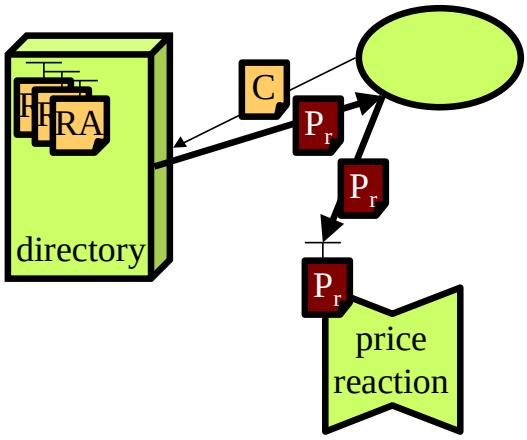


Figure 29: Price reaction policy configuration

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.

### 4.3.12 Distribution and aggregation configuration

For completeness, Fig 30 shows the **distribution and aggregation functions being configured** by application of the relevant **policies,** $\mathbf{P_d}$ & $\mathbf{P_a}$. The configuration of the distribution function also depends on the current **context**, $\mathbf{C}$.

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.

## 4.4 Utility building blocks

### 4.4.1 Correlation

The **correlation** building block (Fig 31) is likely to be used primarily as a security defence. It operates on **management control** message payloads, $\mathbf{M}$ (defined earlier). It takes at least two inputs, either in time or space and gives an output **context message,** $\mathbf{C}$, 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

  

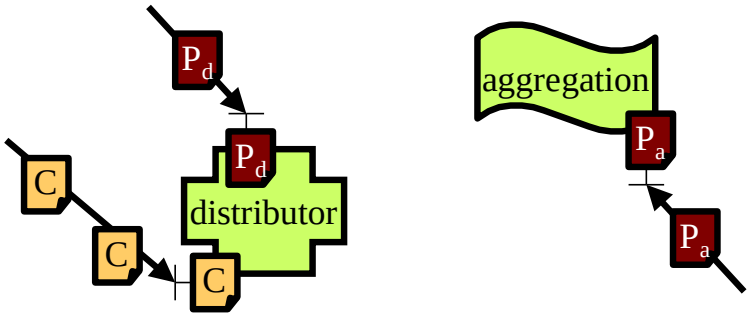


Figure 30: Distribution and aggregation configuration

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).

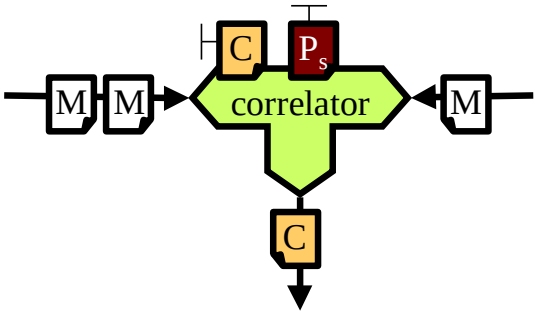


Figure 31: Correlation

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

### 4.4.2 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.

## 4.5 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 (Part I). On the theme of building 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 services that use the network service

  

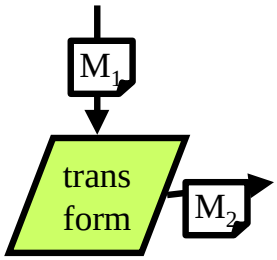


Figure 32: Transformation

- selling and buying 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 openness 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 33) 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):

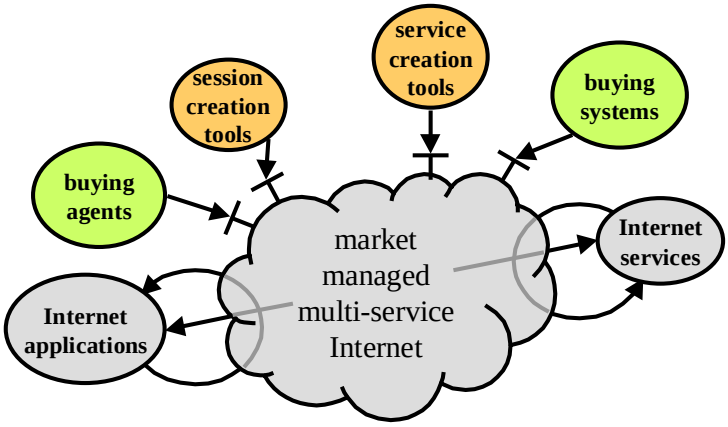


Figure 33: Applications of an M3I

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

- 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 commu-

  

nicate 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 4.3.7) 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!

## 4.6 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 seem unclear why we called packets invocations. This was the only term that covered both data and signalling packets — a consequence of the Internet handling both types of packets together. 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 [28].

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 (and eventually 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.

## 4.7 Interim summary of parts

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

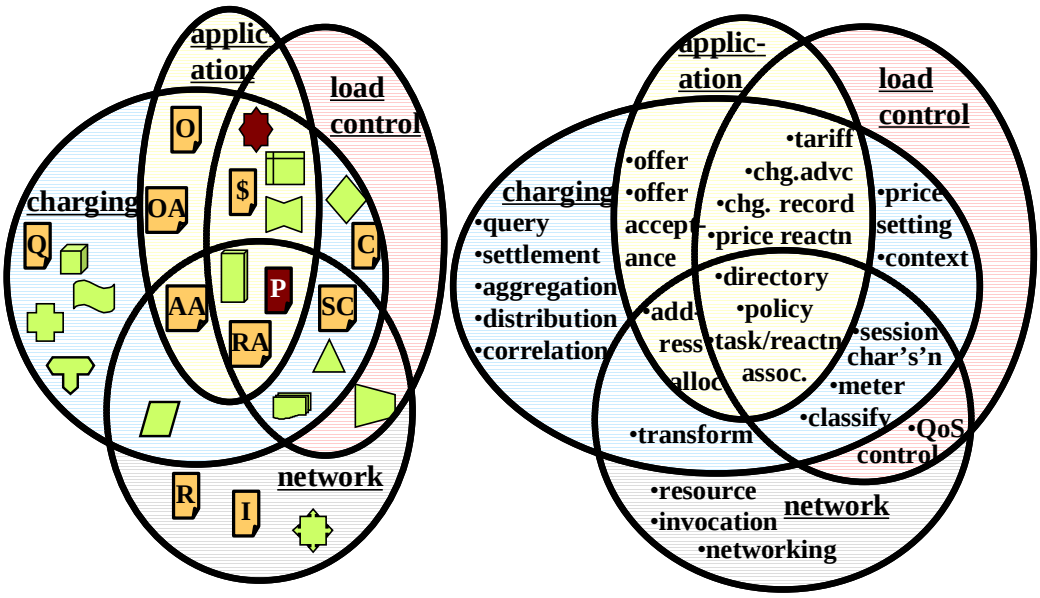


Figure 34: Classification of main building blocks and message payload types

Fig 34 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.

  

# 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 described in Appendix A.1. We identify internal APIs in italics followed by (A). All other interfaces may be remote, so 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, $\mathbf{I_p}$**, or **QoS signalling, $\mathbf{I_q}$** (e.g. RSVP [6]). Setting **resource, $\mathbf{R}$**, if it is possible to do remotely, is done using protocols specific to the link layer in use. Setting **routing policy, $\mathbf{P_n}$**, and propagating it throughout the Internet, is done with standard routing protocols.

The M3I project will deliberately avoid altering any of these protocols, as the aim is to apply pricing to whatever can be passively metered. Therefore, we have no need to discuss network interfaces further.

## 5.2 Service operation interfaces

### 5.2.1 Session characterisation interface family

<table>
<tr><th>input to<br>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>-<br>$SC5$</td><td>-</td><td>$SC3$</td><td>?</td><td>$SC3$?</td></tr>
<tr><td>Distributor</td><td>oneway</td><td rowspan="2">-</td><td colspan="3" rowspan="2">$SC4$</td><td rowspan="2"></td><td rowspan="2"></td><td></td><td></td></tr>
<tr><td rowspan="2">Aggregator</td><td>oneway</td><td>$SC4$</td><td>?</td><td>$SC4$?</td></tr>
<tr><td>query-response</td><td colspan="8">-</td><td>$SC6$</td></tr>
<tr><td>Price reaction</td><td>oneway</td><td colspan="8">-</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="8">-</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 2: Session characterisation interface family

We assume at most six session characterisation interface types, $SC1$–$SC6$, 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 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 [13].

**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 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 $SC3$ & $SC4$, 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 $SC3$ & $SC4$.

**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), 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.

More detailed and precise specifications of many of these session characterisation and charge advice interfaces are given in the relevant design [44, 35, 11] and implementation [45, 36, 30] reports.

### 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 $I_q$ | $P_q$ |

Table 3: QoS control policy interface family

The contents of a QoS control policy may either re-use existing static QoS signalling protocols and interfaces (e.g. RSVP and RAPI) [11] or define the parameters of a dynamically adapting QoS controller. Implementation experience late in the project reported in [30] has shown that an interface that only passes static values is insufficient, at least for the dynamic price handler scenario. However, a compromise is necessary between, at one extreme, giving a low level QoS controller a completely non-linear policy just once, or at the other, very frequently updating a static policy. Various ideas across this spectrum will be evaluated analytically and experimentally, but unfortunately beyond the end of the M3I project.

  

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. 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 [33], but there is obviously no concept of describing congestion feedback policy in RSVP.

## 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 4: Tariff interface family
$T$ = tariff protocol

The issues concerning a tariff protocol and API and a pricing protocol and API are discussed in [35, 36].

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 4.3.3) will need to be standardised and we have proposed a standard tariff description framework [27, 36] allowing a small set of standard protocols, rather than assuming the world will standardise around a single one.

TUD's previous work referred to in the above reference embeds pricing information in an extension to RSVP. BT's previous work [41] 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 work above can use 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 |
|---|---|---|---|
| Buying agent | request-confirm | $P_r1$ | - |
| | event listener | $P_r2$ | - |
| task-reaction association directory | query-response | - | $P_r$ |

Table 5: Price reaction policy interface family

The contents of a price reaction policy are discussed in [11] and fully defined in [30]. 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 6: 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 [6]. The real-time flow measurement (RTFM) working group of the IETF has defined the simple rules language (SRL) [12] 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.

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 7: 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, $\mathbf{P_d}$, and aggregation, $\mathbf{P_a}$** ) 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, $\mathbf{P_p}$, the buying policy, $\mathbf{P_b}$, and security policy, $\mathbf{P_s}$**. 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 8: 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 [50].

# 6 Compositions

Below we take the approach already described in Appendix A.2, 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.

## 6.1 Components

Table 9 is a list of the sub-systems that appear in at least one of the use cases in Section 2 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 9: Service building blocks for use case sub-systems

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 35) 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.

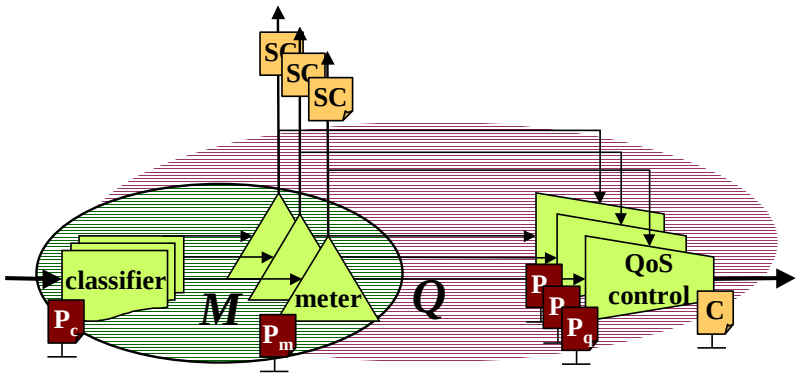


Figure 35: Meter system and QoS manager service components

### 6.1.2 QoS manager service component

The QoS manager consists of a classifier, with one meter and one QoS control service building block per classification. In other words, it consists of a meter system service component as just defined, combined with as many QoS control building blocks as there are classifications. It is shown as $Q$ in Fig 35. This component is very similar to the diffserv policer used as an example composition later (Fig 41a).

The QoS manager component exposes all the interfaces of the building blocks from which it is made, except the internally connected ones. Thus a QoS controller component has the following interfaces:

- $P_c$ in
- $P_m$ 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 36, as it is irrelevant for an end-customer. A provider QoS controller would include this additional interface (for access control, as discussed in Section 4.2.4).

  

### 6.1.3 Mini charging & accounting service component

Fig 36 shows four of the use case sub-systems listed in Table 9 in a typical setting on a single customer host such as in the dynamic price handler or guaranteed stream provider scenarios (M3I requirements [2]). For comparison, the inset bottom left of Fig 36 shows the sub-systems as drawn earlier when describing the use cases for these scenarios (Fig 2 on p11 or Fig 5 on p17). 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 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.

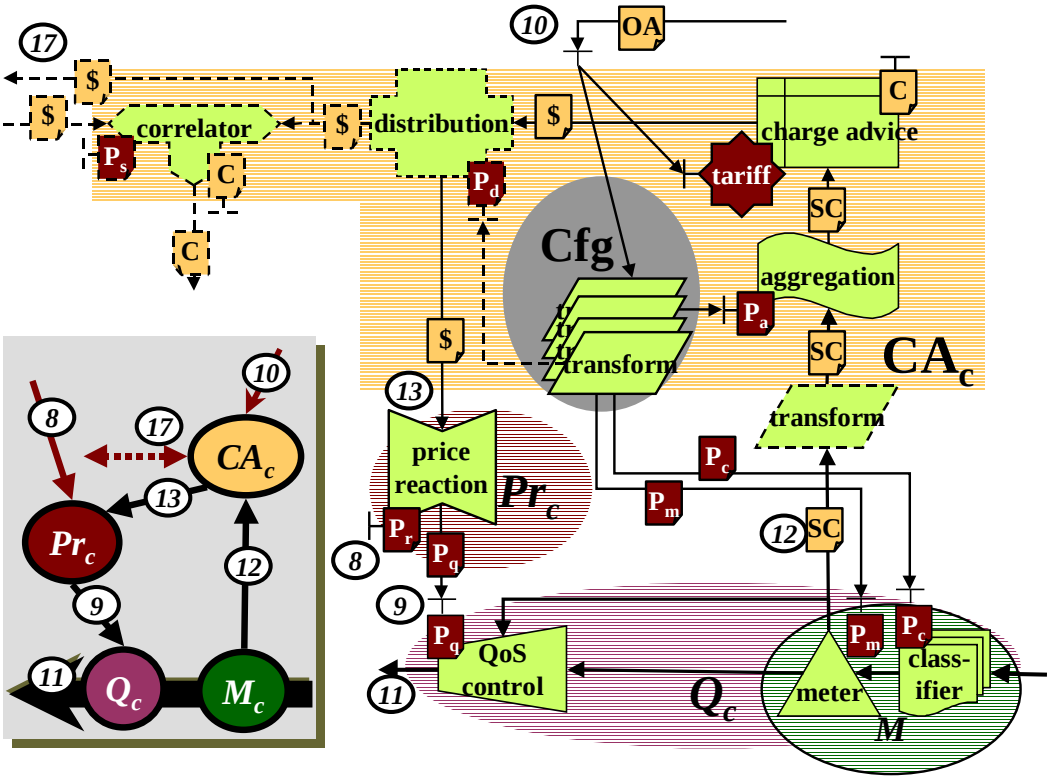


Figure 36: End-customer service components

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 stream 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 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 4.3.3).

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 [11], with a complete implementation in [30]. 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 [44]. 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 37 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 4.2.8). This reduces the 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 4.2.8). 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.

  

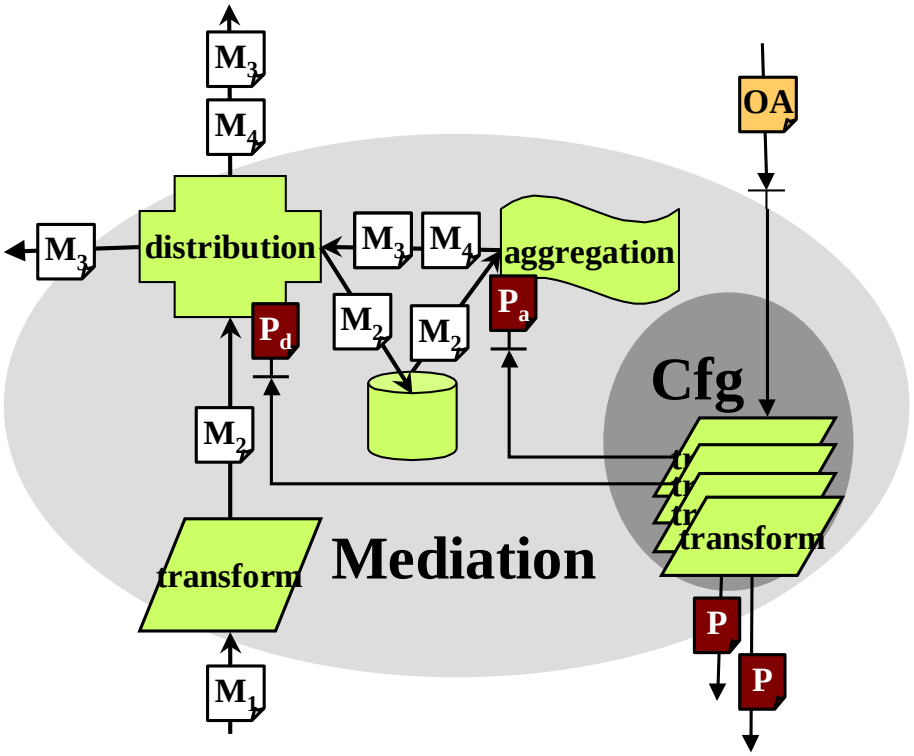


Figure 37: Mediation service component

  

### 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 based on the scenarios in the M3I requirements specification [2]. Comprehensive details on the design of charging and accounting systems can be found in the M3I charging and accounting system design [44], with full implementation details in [45]. 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 [44]), in which case service component would be a valid description.

Fig 38 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, 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.

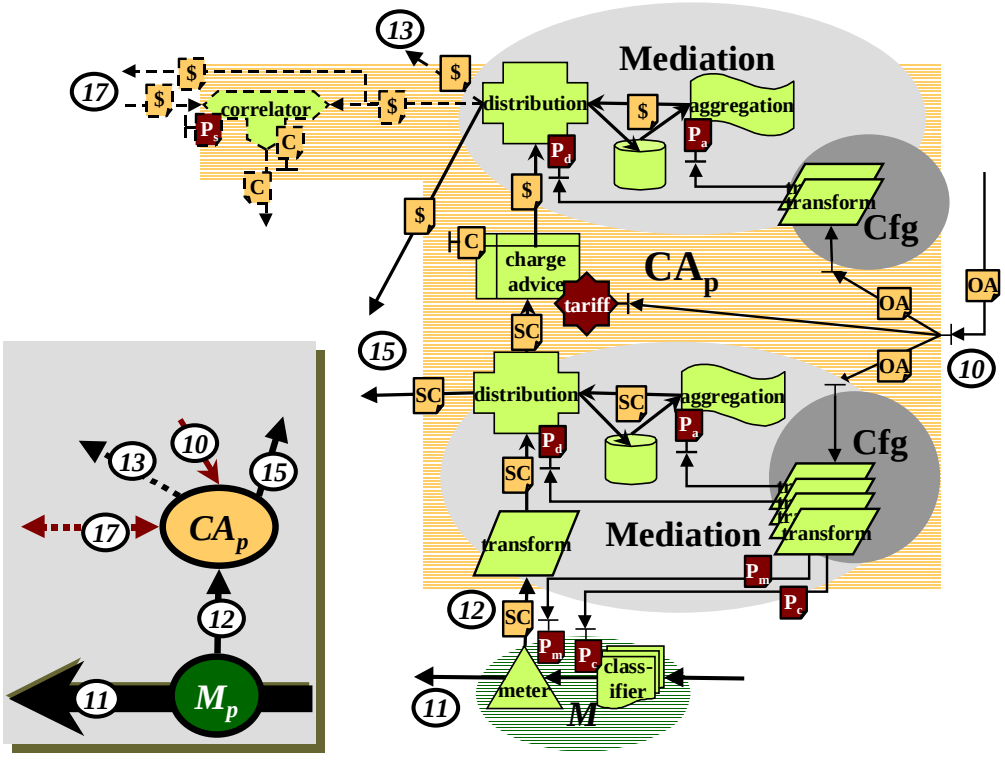


Figure 38: Charging and accounting service component

As with the mini charging and accounting system, for comparison the inset bottom left of Fig 38 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.

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 4.3.7).

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 composition examples

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. Below we describe an exemplary composition of a whole service plan, wherever possible using the service components we have just defined to hide underlying complexity.

### 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 39). 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 39 are the two charging and accounting systems. The internals of each of these are shown in Fig 36 & Fig 38. This proves that a large part of the architectural complexity is in these two systems. 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 2.1, 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 4.3.11. The rest of the customer side is exactly as in Fig 36, 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 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].

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

  

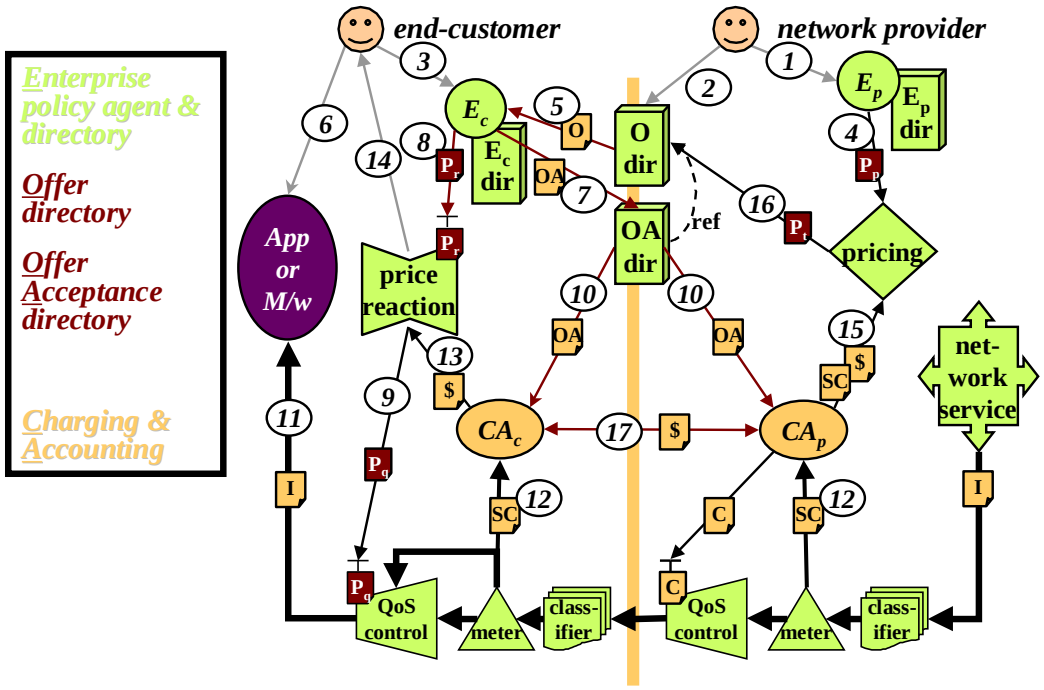


Figure 39: Dynamic price handler composition

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 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

The DPH/ECN scenario example above is a strong guide to the instantiation of many of the other use cases in Section 2. The relevant reports on each M3I scenario include their own building block/component diagrams. In general, the use case sub-systems map directly to the components and building blocks listed against them in Table 9. However, such an approach cannot be applied blindly.

  

# 7 Conclusions

## 7.1 Limitations & further work

As we said in Part I, 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.

## 7.2 Summary

The overall summary for both Part I & Part II of this document is given in Part I, as it is designed to stand-alone. Below is the summary for just Part II.

An architecture for a market managed multi-service Internet has been defined. It is a decomposition of the low level functions of networks, their load control, the charging system that polices each customer-provider relationship and the applications & policies that drive the system. Load control is always ultimately within a market context at some time-scale, which is therefore the abstraction used. The resulting architecture is uniformly presented in a single stretch from the low level embedded control of the networking data path up to high level corporate policy. The building blocks and interfaces between them form a self-consistent set with none dangling, other than those deliberately intended to be for human input only. Thus, we are a significant step closer to the goal of *self*-managing the Internet with a market. Previously this has been possible on longer time scales, but we have stretched the time scale spectrum down to per packet, but only where there is demand for such fine-grained control.

As well as decomposition, re-synthesis into a variety of different service plans has been demonstrated. Well-known traditional service plans are supported as well as a wide selection of new proposals, some proposed before the architecture was defined, and others after. As few assumptions as possible have been made on the business models the Architecture must support. Commercial openness has been achieved by abstracting the market control and charging aspects away from the network infrastructure, describing the commercial offers and relationships as information structures, rather than embedding assumptions into the infrastructure. Not only network providers are empowered to create new service plans, but they can also allow their customers to create innovative new business models too – true commercial openness.

## Acknowledgements

Contributors are all acknowledged in Part I.

  

# References


[1] Jörn Altmann. M3I; ISP business model report; prototype descriptions. Deliverable 7.2, M3I Eu Vth Framework Project IST-1999-11429, URL: http://www.m3i.org/private/, January 2001.

[2] Ragnar Andreassen (Ed.). M3I; Requirements specifications; reference model. Deliverable 1, M3I Eu Vth Framework Project IST-1999-11429, URL: http://www.m3i.org/, July 2000.

[3] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services. Request for comments 2475, Internet Engineering Task Force, URL: rfc2475.txt, December 1998.

[4] A. Bouch, M. Sasse, and H. G. DeMeer. Of packets and people: A user-centred approach to quality of service. In *Proc. International Workshop on QoS (IWQoS'00)*, URL: http://www.cs.ucl.ac.uk/staff/A.Bouch/42-171796908.ps, May 2000. IEEE/IFIP.

[5] R. Braden, D. Clark, and S. Shenker. Integrated services in the Internet architecture: an overview. Request for comments 1633, Internet Engineering Task Force, URL: rfc1633.txt, June 1994.

[6] R. Braden (Ed.), L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) — version 1 functional specification. Request for comments 2205, Internet Engineering Task Force, URL: rfc2205.txt, September 1997.

[7] Bob Briscoe. The direction of value flow in multi-service connectionless networks. In *Proc. International Conference on Telecommunicatons and E-Commerce (ICTEC'99)*, URL: http://www.btexact.com/projects/mware.htm, October 1999.

[8] Bob Briscoe. M3I Architecture PtI: Principles. Deliverable 2 PtI, M3I Eu Vth Framework Project IST-1999-11429, URL: http://www.m3i.org/, February 2002. (To appear).

[9] Bob Briscoe and Sandy Johnstone (Eds.). M3I platform technologies. Deliverable 0, M3I Eu Vth Framework Project IST-1999-11429, URL: http://www.m3i.org/private/, July 2000.

[10] Bob Briscoe, Mike Rizzo, Jérôme Tassel, and Konstantinos Damianakis. Lightweight, end to end, usage-based charging for packet networks. In *Proc. IEEE Openarch 2000*, pages 77–87, URL: http://more.btexact.com/projects/mware.htm, March 2000.

[11] Bob Briscoe (Ed.). M3I pricing mechanism design; Price reaction. Deliverable 3 Pt II, M3I Eu Vth Framework Project IST-1999-11429, URL: http://www.m3i.org/, July 2000.

[12] N. Brownlee. SRL: A language for describing traffic flows and specifying actions for flow groups. Request for comments 2723, Internet Engineering Task Force, URL: rfc2723.txt, September 1997.

[13] N. Brownlee, C. Mills, and G. Ruth. Traffic flow measurement: Architecture. Request for comments 2063, Internet Engineering Task Force, URL: rfc2063.txt, January 1997.

[14] The directory: Overview of concepts, models and service. Recommendation X.500, CCITT, 1988.

[15] Open settlement protocol (OSP) version v2.1.1. Technical specification 101321, ETSI TIPHON, URL: http://www.etsi.org/, August 2000.

[16] George Fankhauser, Burkhard Stiller, Christoph Vögtli, and Bernhard Plattner. Reservation-based charging in an integrated services network. In *Proc. 4th INFORMS Telecommunications Conference, Boca Raton, FL*, URL: ftp://ftp.tik.ee.ethz.ch/pub/people/stiller/paper/informs98.ps.gz, March 1998.

[17] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol — HTTP/1.1. Request for comments 2616, Internet Engineering Task Force, URL: rfc2616.txt, June 1999.

[18] N. Freed and N. Borenstein. Multipurpose Internet mail extensions (MIME) part two: Media types. Request for comments 2046, Internet Engineering Task Force, URL: rfc2046.txt, March 1998.

[19] Richard J. Gibbens and Frank P. Kelly. Distributed connection acceptance control for a connectionless network. In *Proc. International Teletraffic Congress (ITC16), Edinburgh*, pages 941–952, URL: http://www.statslab.cam.ac.uk/~frank/dcac.html, 1999.

[20] Amy R. Greenwald, Jeffrey O. Kephart, and Gerald J. Tesauro. Strategic pricebot dynamics. In *Proc. ACM E-Commerce (EC'99)*, URL: http://www.ibm.com/iac/ec99/, November 1999.

[21] Benjamin N. Grosof, Yannis Labrou, and Hoi Y. Chan. A declarative approach to business rules in contracts: Courteous logic programs in XML. In *Proc. ACM E-Commerce (EC'99)*, URL: http://www.ibm.com/iac/ec99/, November 1999.


  

[22] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg. SIP: Session initiation protocol. Request for comments 2543, Internet Engineering Task Force, URL: rfc2543.txt, March 1999.

[23] Mark Handley. *On Scalable Internet Multimedia Conferencing Systems*. PhD thesis, Dept. of Computer Science, UC London, URL: http://www.aciri.org/mjh/thesis.ps.gz, November 1997.

[24] Mark Handley and Van Jacobsen. SDP: Session description protocol. Request for comments 2327, Internet Engineering Task Force, URL: rfc2327.txt, March 1998.

[25] Mark Handley, Colin Perkins, and Edmund Whelan. Session announcement protocol. Request for comments 2974, Internet Engineering Task Force, URL: rfc2974.txt, October 2000.

[26] Vesna Hassler. X.500 and LDAP security: A comparative overview. *IEEE Network*, 13(6):54–64, 1999.

[27] Oliver Heckman, Vasilios Darlagiannis, Martin Karsten, and Bob Briscoe. Tariff dissemination protocol. Internet draft, Internet Engineering Task Force, URL: http://www.m3i.org/papers/draft-heckmann-tdp-00.txt, March 2002. (Expired).

[28] G. Huston. Next steps for the IP QoS architecture. Request for comments 2990, Internet Architecture Board, URL: rfc2990.txt, November 2000. Status: informational.

[29] Open distributed processing reference model (RM-ODP). Standard 10746-1 – 10746-4, ISO/IEC, January 1995. Or ITU-T (formerly CCITT) X.901 to X.904.

[30] Arnaud Jacquet (Ed.). M3I; price reaction: Detailed design. Deliverable 12.2, M3I Eu Vth Framework Project IST-1999-11429, URL: http://www.m3i.org/private/, February 2002.

[31] Sandy Johnstone (Ed.). M3I; final summary report. Deliverable 17, M3I Eu Vth Framework Project IST-1999-11429, URL: http://www.m3i.org/private/, February 2002.

[32] Martin Karsten and Jens Schmitt. Admission control based on packet marking and feedback signalling – mechanisms, implementation and experiments. Technical Report TR-KOM-2002-03, TU-Darmstadt, URL: http://www.kom.e-technik.tu-darmstadt.de/publications/abstracts/KS02-5.\%html, May 2002.

[33] Martin Karsten, Jens Schmitt, Nicole Beriér, and Ralf Steinmetz. On the feasibility of RSVP as general signalling interface. In *Proc. 1st International workshop on Quality of future Internet Services (QofIS'00)*, URL: http://www.fokus.gmd.de/events/qofis2000/html/abstracts.html\#KAR0900:Fe\%asibility,, September 2000. COST263.

[34] Martin Karsten, Jens Schmitt, Lars Wolf, and Ralf Steinmetz. An embedded charging approach for RSVP. In *Proc. International Workshop on QoS (IWQoS'98)*, pages 91–100, URL: http://www.kom.e-technik.tu-darmstadt.de/publications/abstracts/KSWS98-\%1.html, May 1998. IEEE/IFIP.

[35] Martin Karsten (Ed.). M3I pricing mechanism (PM) design. Deliverable 3 Pt I, M3I Eu Vth Framework Project IST-1999-11429, URL: http://www.m3i.org/, June 2000.

[36] Martin Karsten (Ed.). M3I; integrate pricing system and network technology. Deliverable 12.1, M3I Eu Vth Framework Project IST-1999-11429, URL: http://www.m3i.org/private/, June 2001.

[37] Martin Karsten (Ed.). GSP/ECN technology & experiments. Deliverable 15.3 PtIII, M3I Eu Vth Framework Project IST-1999-11429, URL: http://www.m3i.org/, February 2002. (updated by [32]).

[38] Frank P. Kelly, Aman K. Maulloo, and David K. H. Tan. Rate control for communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research Society*, 49(3):237–252, 1998.

[39] K. Nichols, S. Blake, F.Baker, and D. Black. Definition of the differentiated services field (DS field) in the IPv4 and IPv6 headers. Request for comments 2474, Internet Engineering Task Force, URL: rfc2474.txt, December 1998.

[40] K. K. Ramakrishnan, Sally Floyd, and David Black. The addition of explicit congestion notification (ECN) to IP. Request for comments 3168, Internet Engineering Task Force, URL: rfc3168.txt, September 2001.

[41] Mike Rizzo, Bob Briscoe, Jérôme Tassel, and Kostas Damianakis. A dynamic pricing framework to support a scalable, usage-based charging model for packet-switched networks. In *Proc. Int'l W'kshp on Active Networks (IWAN'99)*, volume 1653, URL: http://www.btexact.com/projects/mware.htm, February 1999. Springer LNCS.

[42] Scott Shenker, David Clark, Deborah Estrin, and Shai Herzog. Pricing in computer networks: Reshaping the research agenda. *ACM SIGCOMM Computer Communication Review*, 26(2), April 1996.

  

[43] David Songhurst, editor. *Charging Communications Networks; From Theory to Practice*. Elsevier, 1999.

[44] Burkhard Stiller (Ed.). M3I charging and accounting system (CAS) design. Deliverable 4, M3I Eu Vth Framework Project IST-1999-11429, URL: http://www.m3i.org/, June 2000.

[45] Burkhard Stiller (Ed.). M3I charging and accounting system (CAS) implementation. Deliverable 13, M3I Eu Vth Framework Project IST-1999-11429, URL: http://www.m3i.org/private/, July 2001.

[46] Jérôme Tassel, Bob Briscoe, and Alan Smith. An end to end price-based QoS control component using reflective Java. In *Proc. 4th COST 237 workshop*, URL: http://www.btexact.com/people/briscorj/papers.html\#QoteS, December 1997. Springer LNCS.

[47] (Unidentified). Quality of service routing (qosr). Working group charter, Internet Engineering Task Force, URL: http://www.ietf.org/html.charters/qosr-charter.html, Continuously updated.

[48] Various. Simple mail transport protocol (SMTP). Request for comments (Various), Internet Engineering Task Force.

[49] R. Yavatkar, D. Pendarakis, and R. Guerin. A framework for policy-based admission control. Request for comments 2753, Internet Engineering Task Force, URL: rfc2753.txt, January 2000.

[50] W. Yeong, T. Howes, and S. Kill. Lightweight directory access protocol. Request for comments 1777, Internet Engineering Task Force, URL: rfc1777.txt, March 1995. (v2).

  

# A Justification of approach

## A.1 Interface definition approach

The full definition of each service building block (Section 4) includes their *function* and an enumeration of all required inputs and expected outputs — that is the **interfaces**. Then, a detailed list of all the interfaces required between them can be produced (Section 5), but only 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:

### A.1.1 Interface distribution

In any particular scenario, certain interfaces will be between two 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. 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 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 [9].

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).

### A.1.2 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** (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 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** 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.

### A.1.3 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, for our specific purposes, the **service primitives (methods)** employed or the **other protocol parameters** are not the main concern, both usually being fairly trivial.

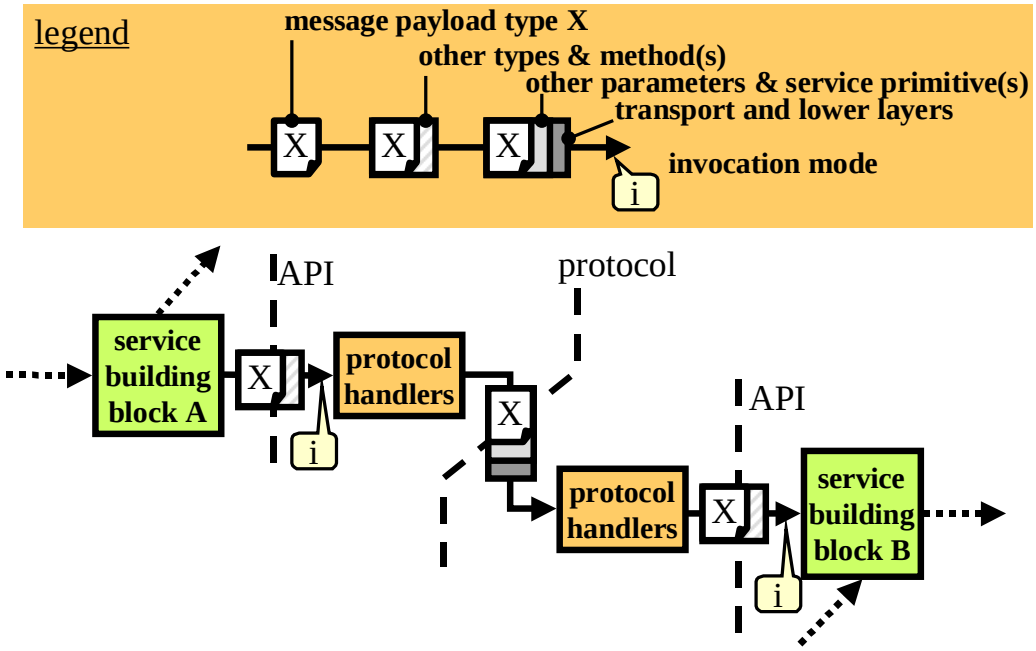


Figure 40: Interface approach

To summarise, Fig 40 shows diagrammatically the approach to identifying interfaces. 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 to rename or even cast to a common type, 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 4.2.5 is request-reply ($i = r/r$). From Section 4.2.6 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 [15]) or we may find this is inadequate and have to improve on it. The interaction mode of both ends is defined as request-reply. 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.

## A.2 Composition definition approach

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

### A.2.1 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 41a — the symbols used are introduced in Section 4 and briefly in Section 4.1 using Fig 34). 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 41b). 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 41c — 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.

  

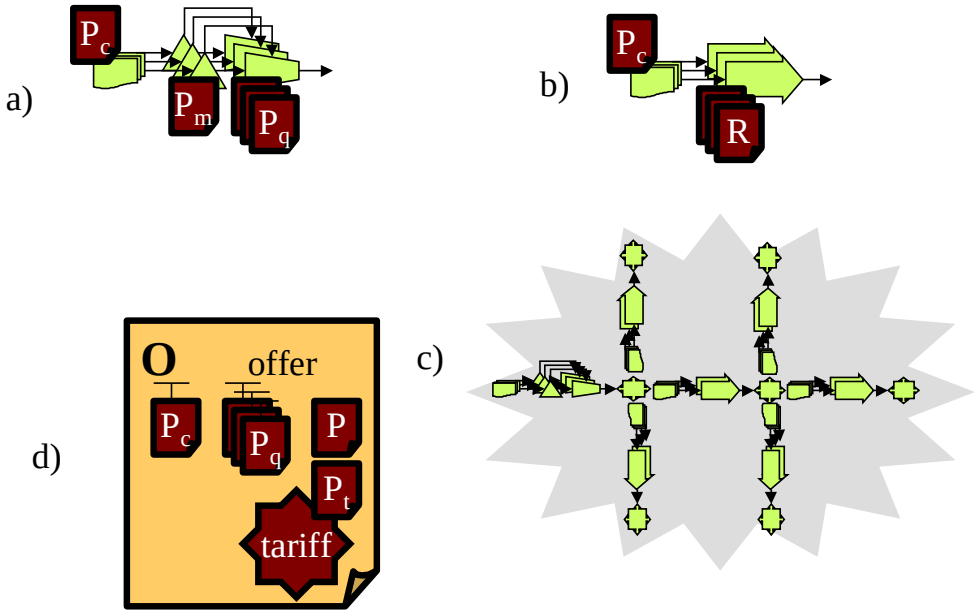


Figure 41: a) diffserv policer; b) diffserv classifier; c) diffserv network and c) diffserv offer

In Section 4.3.4 we define a **service offer** more precisely, but the brief description of the diffserv offer shown in (Fig 41d) should suffice for this introductory note. From top left to right, our example **diffserv offer** includes:

- The **classifier policy,** $\mathbf{P_c}$ to be completed by the customer. The mappings that the customer wants between application protocols and diffserv code-points.
- The **QoS control policy,** $\mathbf{P_q}$, 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,** $\mathbf{P}$. 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,** $\mathbf{P_t}$, 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 a **'wiring' 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 composition of a design. It is also necessary to specify the **granularity** of composition: by **deployment granularity**, by **role granularity** and by **event granularity**.

### A.2.2 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 10 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 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 | aggregation | user | application protocol | host | management system | router: edge interface | router: non-edge interface | domain |
|---|---|---|---|---|---|---|---|---|---|---|
| diffserv policer | | | | | | | | • | | |
| diffserv classifier | | | | | | | | | • | |
| diffserv network | | | | | | | | | | • |
| $P_c$ | | | | | | | | • | • | |
| $P_q$ | | | | | • | | | • | | |
| R | | | | | • | | | | • | |
| ... | | | | | | | | | | |

Table 10: 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.

### A.2.3 By role granularity

Table 11 is similar to the left hand half of Table 10, 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, e.g. 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;
- 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.

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.

  

| per: service component | End-customer | End-user network provider | | Access provider | | Backbone provider | | Value-added service provider | | App/content provider | | Comms service provider | | Charging function provider | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 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 11: Composition granularity by role
W = writable; (W) = writable but unlikely; R = readable; (R) = readable but unlikely

### A.2.4 By event granularity

Table 12 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 | reservation | session | query | addr. alloc'n | daily batch | price-chnge | billing period | offer acceptance | svc lifetime |
|---|---|---|---|---|---|---|---|---|---|---|
| set $P_c$ | | | | | | | | | • | |
| access $P_c$ | • | | | | | | | | | |
| set $P_q$ | | | | | | | | | • | |
| access $P_q$ | • | | | | | | | | | |
| set $R$ | | | | | | • | | | | |
| access $R$ | • | | | | | | | | | |
| ... | | | | | | | | | | |

Table 12: Composition granularity by event

### A.2.5 Notes to clarify the Diffserv composition example

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

  

# B Glossary

As well as those terms defined in Section 3 for distinguishing different elements of this architecture, below we provide a useful list of the M3I project's definitions of a number of terms used in relation to market control of Internet quality of service.

**Accounting** The aggregating of collected resource data into accounting records, whose format is defined by the purpose (billing, capacity and trend analysis, cost allocation, auditing).

**Accounting Record (AR)** The information on resource consumption/usage per user and service, to be used as a basis for charging. NOTE: the content of the AR may include: a duration of a session, start and end time, the geographical position/distance of a high-speed link utilised, number of transactions done, QoS parameters obtained, etc.

**Billing** Aggregating charging information into a bill invoiced to a customer.

**Charge** Charge determines the amount of monetary value that needs to be paid for a particular resource utilisation. It is contained in a charging record.

**Charge advice** The process that produces a charge record

**Charging Record (CR)** The information on the charge determined per user for the service, to be used as a basis for billing.

**Charging** Applying the prices to an accounting record, according to some function on the resource units (tariff).

**Cost** (With respect to the network service) determines the monetary equivalent on equipment, installation, maintenance, management, and operation of networks, network entities, and service provisioning.

**Customer** A stakeholder paying for a service provided by a provider. provider. A customer may also be a provider, either to another party in the value chain.

**End-customer** A customer of a provider that does not sell-on the service, but consumes it directly.

**Mediation** The aggregation and possible re-formatting of metering data. Similar to accounting.

**Metering** The measurement of the resource units consumed by e.g. a flow, a user, a session.

**Network provider** A stakeholder that makes a network service available.

**Offer** An advertisement of a service that includes the contractual terms and conditions, one of which is the tariff

**Price** Determines the monetary value set on the service provided to the user by the provider. The price with respect to a service parameter it the partial derivative of the service charge with respect to that parameter. It may be based on charges and costs or it may be determined by other marketing means.

**Price reaction** A misnomer for charge reaction that became stuck as project terminology due to initial project work package naming.

**Pricing** Setting prices on the service/product/content.

**Session** An instance of use of a service for a bounded time, even if the duration is unknown in advance.

**Session characterisation** An accounting record.

**Settlement** Payment of an outstanding bill or liability.

**Tariff** The formula combining prices and measurable service parameters used to calculate the charge.

**Tariffing** Defining the tariff on the service/product/content.

**User** The party using the (network) service, who may or not be the customer.

The following are abbreviations used in this document, with their scope of usage wherever possible:

**API** Application programming interface

**AR** Accounting record

**ASP** Application Service Provider

**AUEB** Athens University of Economics and Business

  

**BPF** Berkeley packet filter
**BT** British Telecommunications plc
**CA** M3I Charge advice
**CAS** M3I Charging and Accounting System
**CR** Charge record
**CP** Content provider
**Diffserv** IETF differentiated services
**DNS** Domain Name Service
**DPH** M3I Dynamic Price Handler
**DS** IETF Diffserv
**ECN** IETF Explicit congestion notification
**ETSI** European Technical Standards Institute
**GSP** M3I Guaranteed Stream Provider
**HTTP** Hypertext Transfer Protocol
**IDS** Intrusion detection system
**IETF** Internet Engineering Task Force
**Intserv** IETF integrated services
**IP** Internet Protocol
**ISP** Internet Service Provider
**LDAP** IETF Lightweight Directory Access Protocol
**M3I** Market Managed Multi-service Internet
**MIME** IETF Multi-part Internet Mail Extensions
**OSP** ETSI Open Settlement Protocol
**PEP** IETF Policy enforcement point
**PDP** IETF Policy decision point
**QoS** Quality of Service
**Pr** M3I Price reaction policy
**Pq** M3I QoS control policy
**PATH** An IETF RSVP message from the data sender to routers on the path, and onward to the receiver
**RAPI** RSVP API
**RESV** An IETF RSVP message from the data receiver to routers on the path reserving resources
**RFC** IETF Request for comments
**RSVP** IETF Resource Reservation Protocol
**RTFM** Real-time flow measurement
**SC** M3I Session Characterisation
**SAP** IETF Session Announcement Protocol
**SDP** IETF Session Description Protocol
**SDR** Session Directory tool
**SIP** IETF Session Initiation Protocol
**SPSP** Sample path shadow pricing [38]
**SRL** Simple rule language
**TCP** Transmission Control Protocol
**XML** Extensible Mark-up Language

  