# A Notification Service for TINA

*Michel Ruffin, Alban Couturier, Olivier Potonniée, Marcel van der Meulen*

Alcatel Corporate Research Center†

**Sabine Habert**

O2 Technology‡

## Abstract

**This paper presents Alcatel's Notification Service which has been designed in the context of TINA. It has been partly developed in the framework of the TINA auxiliary ACTS projects ReTINA (AC048) and VITAL (AC003). A first prototype has been issued by mid 1996. Based on the users feedback, a second version has been designed and will be delivered by end 1997.**

**This paper presents the rationales for such a service and its main features: generic and typed notification, push and pull communication models, filtering, subscription, federation, fault-tolerance, and quality of service. For each feature, a comparison is done with the four actual OMG proposals for a Notification Service and the rationale for our choices in the light of the TINA use of the service is stressed.**

## 1. Introduction

The Notification Service is a DPE service based on the CORBA technology allowing objects to emit asynchronous information - called notifications - without being aware of consumer objects. Similarly, it enables an object to receive notifications without having to interact with supplier objects. The service acts as a broker between suppliers and consumers.

Managing notifications is a standard issue for telecommunication applications. A CORBA-based generic Notification Service should fulfil multiple needs in the context of TINA. For instance, the service can be used to propagate resource configuration updates, accounting information or alarms for reacting to failures.

OMG has specified a service to transmit events between objects. This service, the « Event Service », provides some of the functionality required to dispatch notifications. However, it misses essential properties that make it unusable as it is in the telecommunication domain: filtering and quality of service. Filtering events should drastically reduce the event traffic which can be very high in large scale systems. The Notification Service must also allow to associate data management to events. This data will be known and used by the service to manipulate events in a specialised way. For instance, a priority associated to each event will allow the Notification Service to order events.

This paper presents a global design of a Notification Service for TINA, based on Alcatel's experience in designing and implementing such a service. It compares this design to the current four proposals answering the [OMG RFP] for a Notification Service: [DTSC/Nortel 97, GMD 97, TID and HP 97, NEC et al 97].

Alcatel's Notification Service has been developed in the framework of the TINA auxiliary ACTS projects [ReTINA] and [VITAL]. The requirements on the service are coming from the TINA architecture designers of these projects but also from Alcatel experience in introducing CORBA into TMN. A first prototype has been issued by mid 1996. Based on the users feedback, a second version has been designed and should be delivered by end '97.

By prototyping a Notification Service, Alcatel's goal is not to develop a product, but to understand the requirements on such a service for telecommunication uses and more specifically for TINA uses in order to encourage the development of the resulting features in future products of DPE service providers.

Section 2 presents the requirements for a TINA Notification Service. Section 3 describes the basic principles of a Notification Service: the architecture of the service, the definition of a notification, the push and pull communication model and the filtering mechanism. Then the different features of a Notification Service are

The work presented in this paper has been partly funded by the European Union ACTS projects ReTINA (AC048) and VITAL (AC003).

† Contact : Michel Ruffin, Alcatel Alsthom Recherche, Route de Nozay, 91460 Marcoussis, France. Tel.: +33 (0)1 69 63 13 57, Fax : +33 (0)1 69 63 17 89, E-mail: Michel.Ruffin@aar.alcatel-alsthom.fr

‡ O2 Technology, 7 rue du parc de Clagny, F-78035 Versailles Cedex, France. Tel.: +33 (0)1 30 84 77 46, Fax: +33 (0)1 30 84 77 90, E-mail: sabine@o2tech.fr

presented: subscription, federation, fault-tolerance and quality of service. Finally, section 5 concludes and presents the future work. A table is given in annex summarising the different features of the Notification Service studied.

## 2. Requirements

### 2.1 Needs for a Notification Service

In the Telecommunications Management Network [TMN] systems notifications (called events in the TMN context) are used to propagate data to unknown sets of components. In this context a notification or an event can be of one of two kinds :

- ***E*rror infor*m*ation** used to inform that an element is close to, or has entered into an error state. The receivers of such information can then act to correct the situation.
- **Infor*m*ative infor*m*ation**. For instance, the availability of a new service in the system, or the expiration of a deadline. This kind of data can be used in any application domain.

The needs for a Notification Service have been identified in several TINA applications:

- In Alcatel's work of introducing CORBA into TMN, which can be considered as a first step towards an industrial implementation of TINA. The Notification Service should provide the functions of traditional TMN's Event Forwarding Discriminators to propagate fault signals. Alcatel's Notification Service prototype will be used through a gateway between a CORBA TMN manager and a CMIP based agent.
- The VITAL project expects the availability of a service providing the same functionality as TMN's Event Forwarding Discriminators in the domain of fault information propagation for use in the context of a full implementation of the TINA architecture.
- In ReTINA , the Notification Service is planned to be used to disseminate accounting information. The TINA resource configuration management could also use this service to propagate the resource state modifications. And the O2 database will use the notification service to propagate events, like the logging of a user or the deletion of an object, to objects managing or controlling the database.

### 2.2 Requirements for a TINA Notification Service

For telecommunications and other application domains, the following requirements on the notification service have been identified:

- **Sca*l*a*b*i*l*i*t*y** is mandatory in telecommunications systems, which are large and highly distributed. Consequently, the notification service should be able to manage important numbers of suppliers, consumers, and events. In order to avoid bottlenecks, the service should not be monolithic.
- ***P*erfor*m*ances** must be kept high, despite of distribution. The communication path between suppliers and consumers must be minimal, and the number of events transmitted on the network must be reduced to the subset of pertinent ones.
- ***F*a*ul*t To*l*erance** is essential for a service to maintain the system in a safe state. This property can be observed at different levels of the service. First, the service must be able to resist crashes, and thus maintain a persistent copy of its state. Second, it must resist to erroneous interactions with suppliers and consumers. For example, it must check validity of filters or notification occurrences in terms of notifications types. Third, it must also ensure the continuity of the service during occurrence of events bursts.
- *Fl*e*x*i*b*i*l*i*t*y* allows users to adapt the service to their needs. This is important for a service that can be used in multiple application domains.
- *Hi*g*h l*eve*l* **interfaces** simplify the developers' task by reducing the learning curve and minimising the code on the client side. Reducing the number of steps to perform an operation reduces the risk of bad manipulations.

Experience in distributed object oriented systems led to identify a set of solutions that fulfil these requirements:

- *F*e*d*eration increases the *p*erfor*m*ance and sca*l*a*b*i*l*i*t*y of the system by balancing the load of one component over multiple components.
- *F*i*l*terin*g* reduces the network load, increasing *p*erfor*m*ance by allowing consumers to precisely specify the events they are interested in. Filtering is applied to the data associated to notifications. Only notifications matching filters are sent to consumers.
- **Q*u*a*l*i*t*y of Service** parameters ensure **f***l*e*x*i*b*i*l*i*t*y** by providing an interface for setting the service to the most suitable behaviour. In particular, event queue management and reliability should be customisable depending on notifications properties. These

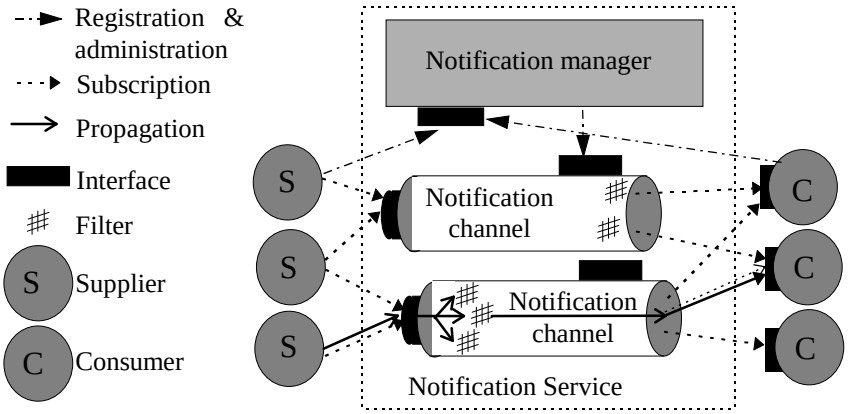


**Figure 1: Notification Service architecture**

properties can be either associated to the kind of notification or to specific instances.

- **T*yp*e*d* an*d* *g*eneric interfaces**, as defined in the OMG Event Service, should both be available to communicate the event data. The choice of the kind of interface is part of the **f*l*e*x*i*b*i*l*it*y*** of the service. Typed interfaces are ***h*igh *l*eve*l* interfaces**, whereas generic interfaces have better ***p*erfor*m*ance**.

## 3. Basic components

This section presents the basic components of the Notification Service: the service architecture, the concept of notification, the pull communication model and the filtering mechanism.

### 3.1 Architecture

Alcatel's Notification Service architecture decomposes the Notification Service in two parts: a notification manager and several notification channels (see Figure 1). Since the notification mechanism has been defined for [TMN] in [X733, X734, X720], normalised concepts have been reused in order to match the telecommunication needs. The TMN Event Forward Discriminator (EFD) which constructs event reports and manages the notification propagation (e.g. filtering and routing) can easily be mapped on the notification channel concept. A notification manager is introduced because general purpose Life Cycle Service functions are not powerful enough to meet the requirements of telecommunications as defined in TMN. This manager is in charge of the global administration of the service (instanciating and removing channels, managing subscription to the service, remembering subscriber's configuration, notification type description, checking types, establishing federation between channels, ...). Subscribers register to the service for supplying or consuming a ***k*ind** of notification (this is defined in Section 4.1). There is one notification channel for each ***k*ind** of notification.

Through the subscription interface, a consumer can dynamically provide a filter to the manager. Filters are used in order to reduce the amount of notifications that a consumer receives. Notification channels provide an interface to the manager for their administration. Among other information, filters are passed to the channels through this interface.

When a supplier or a consumer subscribes for a ***k*ind** of notification and no corresponding notification channel exists, a channel is created by the manager. When subscribing, a supplier gets back in return the reference of a notification channel. Suppliers can then provide notifications to the channel by invoking its notification interface. For each of its consumers, the notification channel uses the consumer's filter to determine whether the notification has to be propagated to the consumer. Propagation is done by invoking the consumer notification interface.

In the example of Figure 1, a supplier invokes the notification channel to propagate a notification. The channel filters the notification according to the filters of each of its consumers. The notification passes through only one of the three filters and is propagated by invoking the corresponding consumer.

The OMG proposals that want to stay compliant with the Event Service specifications ([DSTC/Nortel], [TID/HP] and [NEC]) do not specify the manager part of the architecture and the administration interface of the notification channel is directly accessed by suppliers and consumers. Channel creation and destruction and storage and management of global service information are not directly addressed. [DSTC/Nortel] mentions the use of an event type repository to store the type description of events. [NEC] defines an event network object for managing the federation of channels. Finally [GMD] does

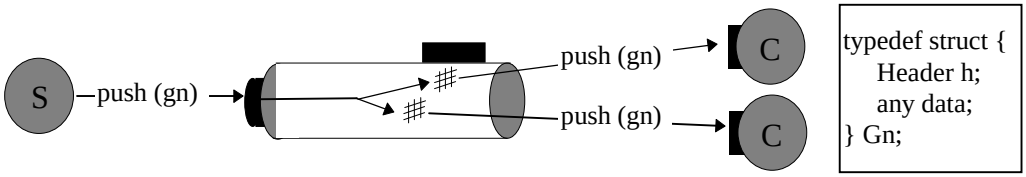


**Figure 2: Generic notification propagation**

not clearly define the channel concept, the propagation of notifications is handled by a notification server, and some internal combination of computational objects assumes the role of the channels.

The manager allows to discharge users from the burden of managing channels (instanciation and destruction) and it offers high level operations on channels such as automatic federation for load-balancing. The split of the service in two kinds of entities (manager and channels) allows to discharge the channels as much as possible from administration work. The main channel task - notification filtering and routing - can thus benefit of more resources. The interface and functions of the channel administration interface are limited to the minimum since a part of the work is done by the manager. This basic choice is directly driven by scalability and performance requirements.

## 3.2 Notification

A notification is passed by invoking an operation on the notification interface. The notification interface is the same for a supplier invoking the service side (i.e. the channel) as for the service invoking a consumer. Interface types can be either generic or typed corresponding to generic or typed notifications.

Both notification propagation models can be seen as multicasting asynchronously on some set of unknown objects after filtering. From the user point of view, typed notifications allow to propagate notifications as a remote invocation on the consumers with the guarantee of type checking. Generic notifications are lower-level and allow to propagate notifications like packets of data. The clients have to pack the data on the supplier side and unpack it on the consumer side. Type checking is not guaranteed and consequently this method is more error prone. The drawback of typed notifications is the complexity of a user-transparent implementation and the fact that the use of typed notifications in this implementation might be time consuming compared to the use of generic notifications.

In Alcatel's Notification Service, generic or typed notifications are **ar*b*itrar*y*** and ***u*ser-*d*efine*d***. This proves to be very convenient for the user because from this point of view the channel interface is defined by the user. In order to get better performances for typed notifications we will consider predefined typed notifications later on (Section 3.2.3).

### *3.2.1 G*eneric notifications

A generic notification is a user-defined set of data and a header. The generic notification interface supports a single operation (let's call it `push ()`) having as a parameter the generic notification. In Alcatel's implementation the data part of the generic notification is encapsulated by a CORBA « any » type.[1] The header may contain information such as quality of service requested, supplier identification, timestamps, notification family information (alarm, error, warning, ...).

Figure 2 presents the propagation of a generic notification. The supplier invokes the « `push ()` » operation of the channel with a generic notification (gn) as parameter. After notification filtering (assuming that the notification passes the filters) the channel invokes the same operation on each consumer.

### *3.2.2* T*yp*e*d* notifications

A typed notification is an operation with an arbitrary number of user-defined typed parameters. The type of the notification is the signature of the operation (operation name and type description of its parameters). A notification interface can contain several typed notifications.

Typed notifications are conceptually more object-oriented than generic notifications, since the propagation


**Figure 3: Typed notification propagation**

[1] The CORBA « any » type allows to invoke dynamically operations taking as parameter some data of unknown type. The « any » is composed of two fields containing the type description and the data

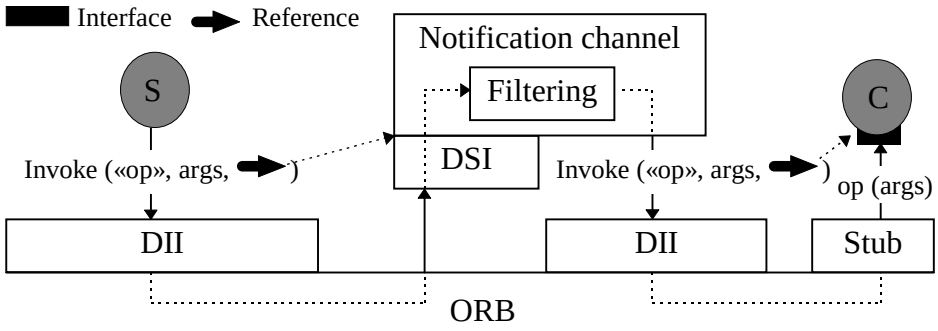


**Figure 4: Implementation of typed notification**

of the notification consists in the propagation of an **ar*b*itrar*y* *u*ser-*d*efine*d*** operation (see Figure 3).

Depending on the use of notifications, both kinds of notification are useful: a low-level mechanism with good performance or a high-level one less error prone.

### *3.***2***.3* **I*mpl*e*m*entin*g* t*yp*e*d* notifications**

Arbitrary client-defined typed notifications constitute a powerful mechanism. In Alcatel's prototype this is implemented by using the Dynamic Invocation Interface (D.I.I.) and the Dynamic Skeleton Interface (D.S.I.).

Figure 4 presents the propagation of a typed notification in our implementation. The supplier wishes to invoke the client-defined operation of name « op » with a list of arguments (noted args) on the notification channel. Notification channels are compiled once and present an interface independent of the client operations. This implies the use of the D.S.I. in order to receive invocations of operations which do not belong to the channel interface. The D.S.I. allows to receive this unknown operation and to transform it to provide to the channel the stringified name of the operations and a sequence of « any » containing the operation parameters.

In the Notification Service V1, the D.S.I. was not used. Instead, the client had to provide a « converter » object for each kind of typed notifications. This « converter » object had to receive notifications from the supplier, transform the invocations into a string and call a notification channel operation to convey the string to the channel. In addition to the burden of writing this object the user also had to provide a factory for the converter object. So the use of the D.S.I. largely simplifies the programming work for the service client.

The use of D.I.I. is needed between the supplier and the channel (see Figure 4) in our current implementation, because the supplier has a reference on a channel object that does not support the method for a specific typed notification in its interface. Between the channel and the consumer D.I.I. is needed because the channel is not linked with the stub for the interface of the consumer.

The main drawback of the use of D.I.I. and D.S.I. is the cost in time of such invocations. For a telecommunication application (and thus for TINA), the few typed notifications which have been defined for TMN can be expected to be suitable for fault-management. Since these kinds of notification can be known in advance, a set of predefined will be added in future versions of the Alcatel's prototype. Since these predefined notifications will be known at compile-time, the use of time consuming mechanisms such as D.I.I. and D.S.I. will not be needed for these notifications.

### *3.***2***.4* **Notification conce*p*t in *OMG p*ro*p*osa*l*s**

[DSTC/Nortel] defines generic notifications as arbitrary sequences of name/value pairs where the names are strings. Some predefined names can be « understood » by the service, allowing to form a notification header. The notification type description is identified by a particular name. Typed notifications are not specified in the proposal.

[GMD] defines four types of generic notifications corresponding to object creation, object deletion, attribute value change and method invocation. The object creation and method invocation types both contain an arbitrary sequence of name/value pairs. Compared to other proposals, there are four kinds of generic notifications with strongly typed headers. Typed notifications are not specified in the proposal but the method invocation type allows to describe an invocation and can be considered as a basic typed notification.

[TID/HP] defines generic notifications as a block of data of any type, arbitrary user-defined and without a header. Typed notifications are not specified.

[NEC] defines arbitrary generic and typed notifications similar to the generic and typed events defined for the Event Service. In addition, structured events are defined for increased filtering performance. These events are composed of a header part, a sequence of name/value pairs and an opaque body. Filtering is limited to the

value. Complex data can be built by for instance passing in an « any » a sequence of « any ».

header and to the sequence section. Anticipating the OMG standard on passing objects by value, an event object encapsulating a structured event is defined.

To our knowledge Alcatel's prototype is the only implementation providing typed notifications that can be arbitrary defined by the user. From the users of our V1 prototype, we learned that whenever possible (when time constraints are not too strong), users prefer to use typed notifications. Generic notifications are an alternative when high performances are needed.

In Alcatel's Notification Service, the type descriptions of notifications must be provided to the service manager before subscribers and consumers can subscribe to them. This allows to avoid the problem foreseen in [NEC] concerning low performance for the filtering of user-defined types, because when providing a description, a name can be associated to notification operations (in the case of typed notification) and data fields. Names can be used for identifying the filtering fields of notifications.

Finally the definition of predefined TMN-like typed notifications allows to overcome the problem of performances for typed notifications, since in this case the use of D.I.I. and D.S.I. is not necessary.

### 3.3 Pull communication model

Until now only the *pu**s**h* communication model has been considered: the notification is moving from object to object at the initiative of the supplier objects. Suppliers invoke the channel when a notification occurs and the channel invokes the consumers on a notification reception.

With the *pull* communication model the notification is moving between objects at the initiative of objects needing notifications: consumers invoke the channel to get notifications or the channel invokes the suppliers to get notifications. The two models can be combined as requested, suppliers and consumers choosing their communication model with the channel at registration time.

The pull communication model implies the need for an additional notification interface. This interface should be supported by the channel when consumers use this model or by a supplier if the supplier wants the channel to pull its notifications. This interface is nearly symmetric to that of the push model. For instance for generic notifications, the consumer needing a notification will invoke the `pull ()` operation of the channel. It will be blocked until a notification will be available. An additional non-blocking operation (`try_pull ()`) is added in order to return even when no notification is available.

In Alcatel's implementation the pull communication model is limited to generic notifications. This is due to a need to assess the usefulness of this mechanism in telecommunication applications before extending it to typed notifications. In the context of VITAL, ReTINA and our experience of introducing CORBA in TMN no use has yet been found of the pull model, but all the OMG proposals provide it.

One of the possible advantage of this model is the fact that a consumer does not have to be an object.

For the sake of clarity, the pull communication model will not be considered in the remainder of this paper except when a feature description requires this.

### 3.4 Filtering

The filtering mechanism used by the Alcatel notification service is directly derived from the EFD filtering mechanism described in [X720]. The EFD filtering description language defined in ASN.1 has been translated in IDL. Writing a filter consists in assembling IDL structures which can be send easily and dynamically through invocations. A filter is a recursive structure organised as a tree of logical expressions, whose leafs are filter items. A filter item is a union storing a constraint (equality, substring, superString, greaterOrEqual, lessOrEqual,...) and a couple of attribute names or values. A filter is a union which switches on a filter type ( TRUE, AND, OR NOT, filterItem) and stores a sequence of filter (in the case of AND and OR), a filter (for NOT) or a filter item (for the filter type FilterItem). An extension to the TMN norms filter has been brought providing the possibility to compare two attributes instead of one attribute and one value.

The Alcatel Notification Service does not need to parse strings describing filters, and can manipulate and forward them without string translation. Moreover, a filter can easily be incremented by applying a logical AND or a logical OR between the existing filter and a given filter. Although this filter notation comes from the TMN norms, it is very close to an IDL notation of the OMG trader constraint language.

The place of the filter is essential in a notification service, because the performance depends on the moment and the place where the notification is processed (and possibly deleted) by the filter.

Placing filters into the channel is essential. This allows to optimise the combination of filtering and routing to decrease the network traffic. Only filtered notifications will be multicasted to consumer objects. Another solution consist in placing the filter between the channel and the consumer in a different CORBA object. In this case filtering is done after routing which means that notifications are broadcasted to every filter. This solution decreases greatly the interest of filtering, since in this case, filtering does not reduce the amount of messages transiting between objects.

In a federated notification service, where multiple notification channel can filter the same notification successively (see chapter 6), the filtering must take place as close as possible to the supplier. It could even be in the supplier's address space itself to avoid any network traffic for unwanted events. However, this approach means to have one notification channel, i.e. the encapsulation of the filtering engine, in the supplier's application, like the EFD in TMN. Through an optional advanced interface, the supplier could offer to the notification manager the mean to create a notification channel in the supplier's address space. This implies that the supplier application has to be developed with its own notification channel and filtering machine. This channel must implement every feature of the classical channel. The manager could manage this channel normally, and update the filter in order to match it to the real demands of the final consumers.

In the same way, to avoid the emission of events at their source when there is no consumer awaiting them, the manager could set the suppliers in a mute mode through the advanced supplier interface, until a new consumer registers. This functionality is not offered in the current implementation of our service, but it is planned to be added to future releases. Nevertheless these mechanisms must only be optional features in order to preserve a light, classical and easy use of the notification service.

These filtering features allow to process the notifications as close as possible to the supplier and to send notifications only when a consumer needs them. Consequently Alcatel's Notification Service's filtering is a powerful tool adapted to the need of telecommunications.

# 4. Advanced features

This section describes the Notification Service advanced features which allow the service to be integrated in telecommunication applications.

## 4.1 Subscription

Several points have to be considered for designing a powerful subscription interface:

- **Descri*p*tion of notifications**. Families of notifications can be named. Their type can be described to the system. Some tools can be provided to organise and manage the description information.
- **Descri*p*tion of s*ub*scri*b*ers**. Suppliers and consumers can be identified logically to recognise them in case of failures or to associate a context to them (For instance a filter can be associated to a consumer).
- ***P*ro*p*a*g*ation of s*ub*scri*p*tion infor*m*ation**. When suppliers register to the service, the description of the notification that they intend to send can be made available to other components of the system. When consumers register, their filter can be forwarded to the different emitters of notifications in the system.
- ***P*rovi*d*in*g* fi/ters**: dynamically or statically, by replacing the whole filter or by incrementing (see Section on Filtering).
- The possibility to s*u*s*p*en*d* and res*um*e the reception/emission of notifications for a while.

### *4.1.1* Notification *d*escri*p*tions an*d* s*ubj*ects

In Alcatel's Notification Service, notifications are organised by s*ubj*ects. A subject is an ar*b*itrar*y*, c*l*ient-*d*efine*d* name for notifications which are sharing the same type or a **set** of types. A subject is the minimum unit for subscription or un-subscription.

Subjects must be declared to the service before being used. When declaring the subject, the list of associated notification types should be described. In the case of a generic notification, the type description corresponds to one or several data types. Whereas in the case of typed notifications, the type description is the interface signature corresponding to the set of operations associated with the subject. The signature is composed of the list of signatures of the operations; this signature of an operation is composed of the operation name and the type description of its arguments. For a typed notification subject, the type description will be used to check at registration time whether suppliers propose to invoke the correct service interface and whether consumers provide the correct interface to the service. For generic and typed notifications, the type description is used in the filtering to identify the data field (generic notifications) or the operation name and the arguments of operations (typed notifications).[2]

A subject is an abstraction allowing to organise notifications logically. Any subject for which a subscription has been done corresponds to an instanciated notification channel.

Subjects can be organised hierarchically. For instance, « failures/hardware/disk » could be a subject in which « disk » is a sub-subject of « hardware ». Subjects are declared by the user and form a hierarchy organised as a forest.

Once subjects are registered by the service, suppliers and consumers can subscribe to several subjects in a single operation and it is possible to register a set of consumers or suppliers for a subject in one operation. When registering, a type description should be provided to the service and type checking is done between the

[2] In the current Alcatel implementation, sets of types are not yet implemented. Only one data type can be associated to a subject for generic notifications, and only one operation description can be associated to a subject for typed notifications.

provided description and the type description which has been registered with the subject.

Initially when this design has been done, it was planned that when registering for a subject an object was registering for all its sub-subjects. This is complex to implement. A possible implementation would consist in having a channel for each sub-subject and using federation for reproducing the hierarchy of subjects. A sub-subject channel being a producer of its higher level subject channel. This is an interesting feature for providing a better API to service subscription and is will be considered for future versions of the service.

All other proposals define a concept similar to a subject. [NEC], [DSTC/Nortel] and [GMD] also specify that a type description of the subject should be stored. Only [DSTC/Nortel] specifies that subjects could be organised hierarchically, but this is not further specified.

The notion of subject provides a powerful API to the service and is possible because the service architecture offers a centralised manager able to cope with general management information.

#### *4.1.2* **Subscriber descriptions**

In the push (respectively pull) communication model consumer (respectively supplier) objects should provide an object reference for allowing the channel to make invocations.

References are not guaranteed to stay unchanged when for example a fault occurs. Consequently a logical identifier is needed to identify the subscriber object independently of the object location and the object instance.

For instance after a client-side failure a new subscriber object can be instanciated which will continue the work of the previous instance.

Identification is also needed for security reasons. It should be possible to restrict notification delivery to a list of consumers. It should also be possible to restrict the acceptance of notifications depending on which supplier emitted the notification.

Identification can also allow the service manager to register configurations. For example a consumer could register a filter description that could be used for each instance of this consumer. This is also used for the suspend/resume mode (see Section 4.1.4).

Alcatel's Notification Service defines subscription identifiers as arbitrary user-defined strings which are used for all of these purposes except security which is not implemented.

In [NEC], [DSTC/Nortel] and in [TID/HP] a subscriber is only identified by its object reference. In [GMD] unique names are used to identify suppliers and consumers.

#### *4.1.3* **Subscription information propagation**

When a subscriber registers for a kind of notification, the information corresponding to this subscription can be propagated through the channel.

When a supplier registers, a description of its subscription can be sent to the consumers. This is called « **offering** » in [DSTC/Nortel]. This « offering » concept implies that a consumer has to provide the necessary interface to support this concept to the service.

When a consumer registers, its filter is passed to the channel. This filter could be moved as near as possible to the supplier location in order to reduce the network traffic by filtering at the source. In fact to be efficient a filter should be built at each level of the transmission chain to gather the requirement of all the consumers concerned by this level. [DSTC/Nortel] calls this « *quenching* ». At the supplier side this requires an interface accepting filters and the possibility to interpret filters. The possibility of updating filters dynamically and incrementally is necessary.

Alcatel's service provides a limited implementation of quenching in the case of channel federation (see Section 4.2 ). Offers propagation could be replaced by the manager interface allowing to get information on subjects. This limited implementation avoids an additional interface for the suppliers or consumers.

[TID/HP] and [GMD] do not include subscription information propagation. [DTSC/Nortel] supports both « quenching » and « offer » concepts. [NEC] only mentions the possibility to propagate filters towards the event source, but does not further elaborate on how this should be done.

#### *4.1.4* **Suspend/ resume mode**

To simplify the service use, a suspend mode is desirable. The suspend mode allows a consumer to stop receiving notifications for a while. When entering the suspend mode, all the notifications queued or transiting by the channel are lost for the consumer. When the consumer resumes the reception of notifications, it does not need to subscribe to the subjects or provide filters again.

Alcatel's prototype, [DSTC/Nortel], [TID/HP] and [NEC] are all providing these modes.

#### *4.1.5* **Conclusion on subscription**

Hierarchical subjects, subscriber identifiers, incremental filtering, and suspend/resume mode were not present in Alcatel's V1 prototype. The features in this list corresponding to a better API have been requested by the users for the V2.

Subscriber identifiers have been added to support fault-tolerance, quality of service and security.

The concept of subjects results in a simple registration interface. This combined with the fact that the notification

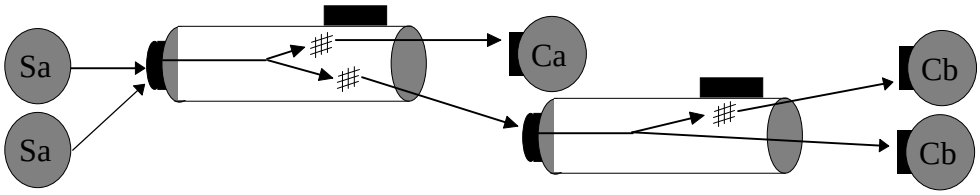


**Figure 5: Channel federation example**

channel interface is defined by the user implies an easy use of the service.

### 4.2 Federation

In case of a lot of recipients and/or emitters or notifications, a notification channel might get congested. A solution for this problem is loadbalancing, the division of the load over multiple channels. Federation is a means to perform this loadbalancing. Federation consists in connecting two channels, with one channel acting as a supplier for the other channel which in its turn is acting as a consumer of the first channel. Federated channels can share the load for better performance and scalability, while maintaining the initial distribution.

Figure 5 presents an example of channel federation. In Alcatel's prototype, two ways to federate channels through the notification server interface have been defined: the « like a supplier » federation, where a channel is asked to register like a supplier to another channel, and the « like a consumer » federation where a channel is asked to register like a consumer channel. The difference between these two is in the filtering. For the first kind of federation, the channel only forwards notifications. For the second kind of federation a filter must be specified, that will be applied to all notifications passing from one channel to the other. This second kind of federation allows specialised channels which offer more fined grained subjects, and reduce the notification load.

If several channels share a common part in their filter, this one can be factorized and passed on to the supplier channel. Such a filter factoring policy allows to reduce the load of the second channel when there is a large amount of notifications or suppliers. This corresponds to the [DSTC/Nortel] quenching mechanism.

Federation also allows to have one Notification Services per network administrative domain in order to structure the notification space. Federation can be intra- or inter- domain. An example of inter-domain federation is given in Figure 5. In Figure 5 we can define two network administrative domains : A (Sa, Ca) and B (Sa, Cb),. The first channel belongs to the notification service A while the second belongs to B. The federation allows a notification emitted in the A domain to be propagated to the B domain.

Intra-domain federations generally aim at filter factorizing or decreasing the address space and site load. For example all high priority notifications of a subject could be propagated to a separate channel, whose subject could be "emergency".

The Alcatel prototype does not offer channel federation, but instead it offers subject federation through the notification manager interface. In contrast with [DSTC/Nortel], [TID/HP] and [NEC], the federation is not performed directly on the channel, but on the manager. This allows to federate subject easily without having to know notification channel instances or their reference.

In the current implementation, subject federation is done at the initiative of a service administrator. Further study needs to be done to federate channels automatically since the policy for federation depends on the service use.

Federation is essential for scalability and performance in telecommunication applications to reduce the network traffic. The fine-tuning of federation at subject-level instead of service-level allows to customise the service to a clients needs.

### 4.3 Fault-tolerance

In TMN, notifications are used to propagate information about failures, so the Notification Service is a building block for reliability in the system. A similar use of the Notification Service by TINA can be expected. A mechanism used for guaranteeing reliability should provide at least the same level of reliability as the expected system reliability. Consequently, the Notification Service should provide different levels of reliability according to the needs of its users.

The different levels of reliability are defined by the kind of fault or malfunctioning behaviours which are considered: it can be the overload of a consumer or a channel[3] which is not able to accept the notification in time, a communication failure between the channel and its subscribers, a site fault (the memory is lost) that can either results in a channel or a subscriber crash or a disk failure resulting in the loss of data stored on a disk.

[3] The case of channel overload is not treated here since in the case of telecommunication some specialised solution can be studied and will be developed in section 4.4.4.

To solve communication failures, overload of a consumer and consumer crash, notification logging can be used to discharge the channel and to re-inject the notifications later on. An additional mechanism on the supplier side is necessary to resist channel crash to change the old channel reference by the reference of the new channel. With the pull communication model, this mechanism will also be necessary on the consumer side. To solve a channel crash, additional information should be logged to remember the channel state: the list of consumers in the push model, the list of suppliers in the pull model, and the list of filters. Some other information can be kept by the service, e.g. the description of the notification types in the case of Alcatel's Notification Service and the federation configuration. This information should be logged to resist site failure (manager crash in Alcatel's case). Finally logged data should be replicated to resist disk failures. The number and place of replica defines the degree of reliability and availability.

Logging allows to save the necessary data for recovery, but a mechanism should be added to do the recovery. Besides ad-hoc solutions for simple recovery situations, transactions can be considered to automate complex recovery. These solutions can be a part of the quality of service modes.

The reliability of the system depends on that of the logging mechanism. The use of a customisable logging service [Ruffin 92] can provide an adapted quality of service: Log replication provides high reliability, log distribution determines availability, and high level logging techniques provide low overhead.

The Alcatel Notification Service provides three levels of fault tolerance: the logging of notifications, the logging of channel states (list of subscribers, filters and notification logging) and manager state information logging. Control operations are provided to select these features and fine-tune them. Notification logging or channel state logging can be switched on at channel level or at service level (for each channel managed by the service). This fine-tuning allows to take into account the quality of service customised to the user needs. The quality of service can be further fine-tuned by using an appropriate transactional model. The use of transaction and logging services is considered for future versions of the service.

[TID/HP] and [DSTC/Nortel] do not define features for fault-tolerance. [GMD] only specifies the logging of notifications. [NEC] defines logging of notifications, logging of the channel state, logging of the global service configuration state and the use of the CORBA Transaction Service.

Fault-tolerance of the service is important for fault-tolerant applications and thus for telecommunications. Being able to choose the level of fault-tolerance (e.g. at channel-level, at service-level) allows to fulfil different applications needs.

## 4.4 Quality of Service

In telecommunications, robustness and performance are of high importance. The requirements on applications, especially management impose the need for a flexible Quality of Service. The Notification Service must for example be able to face a sudden rise of the notification traffic or a more structural overload. To cope with those critical situations, the Notification Service provides the following features: ageing, priority, delivery guarantee and a policy to resist event overload.

### *4.4.1* **A*g*ein*g***

Like for messages in traditional communication protocols, a limit of duration can be assigned to notifications while transiting between suppliers and consumers. A notification which has not been transmitted to consumers after this duration is discarded.

In case of federation, this mechanism (called ageing) can be used in order to avoid cycles. If some channels have been federated in a cycle, a notification can loop indefinitely. With this mechanism the notification will be eventually discarded.

Ageing prevents the buffers to get overloaded, but it can discard relevant notifications. This problem can be remedied by combining this feature with other QoS features.

### *4.4.***2** ***P*riorit*y***

In the generic notification header, a priority field is defined in order to cope with local overload.

An option to be set at the notification manager allows highest priority notifications to be filtered and disseminated before lower priority notifications in the same channel. This feature enables consumers to receive the most relevant notifications as soon as possible.

This priority policy is finer than the basic mechanism where a priority level is assigned to each channel.

### *4.4.3* **De*l*iver*y gu*arantee**

Automatic deletion of notifications can occur when ageing and priority are combined during local overload ; lower priority notifications are waiting for higher priority notifications to be disseminated and risk to be discarded if the channel does not filter them quick enough. The result is an uncontrolled loss of notifications that can create an inconsistency in the system.

To prevent this undesired deletion of notifications an option that guarantees the delivery of a notification can be defined. This option can be set at the notification manager interface, in order to keep lowest priority notifications that

can not be disseminated. These low priority notifications can then be send when the channel is not overloaded any more.

Ageing, priority and guarantee delivery are basic QoS features. They are proposed in [GMD], [TID/HP] and [NEC] too.

#### *4.4.4 P*o*l*ic*y* to resist event over*l*oa*d*

Notifications can be used to propagate error states. Those notifications must be treated in order to avoid a failure of the system. If the error is not processed quickly enough, it might lead to a cascade of errors, leading to a burst of events and an overload of the system. The Notification Service must be able to resist to such situations. Two directions can be investigated to prevent an overload of the system: preventive and curative.

The preventive direction consists in defining a mechanism that minimise the situation where a burst occurs. For example, the use of priorities allows to specify that error events must be processed before informative events. It will result in a smaller response time to errors. The filtering is also one mechanism that reduces the load of the system: events transmitted are only those accepted by managers.

But there is some cases where the flow of events increases and the Notification Service has no control over it. In order to resist to such situations, the service must have a mechanism to reduce the event propagation. A simple policy is to discard events with the lowest priorities. This can be done automatically when the notification lifetime is set: the higher priority notification are forwarded, meanwhile the lower priority ones get old enough in the queue to die. Another possibility is to block suppliers until the load reduces.

But more complex policies can be chosen, that both keep all events, and allow suppliers to continue their work. For example, the overloaded Notification Service could send an exception to suppliers trying to send new notifications, after having registered them in a list of unsatisfied clients. Then, when the load is getting down, the service would advertise its availability to its unsatisfied clients. These suppliers would consequently resend the notification, if it is still meaningful.

Another strategy can be the **s***h***a***d***o***w* **c***h***anne***l***s**. Classically, a subject in one notification domain is associated to one channel. But when the notification manager detects an overwhelming traffic, the critical subject could have multiple channels in the network thanks to a notification factory dedicated to create shadow channels. In the push model, a shadow channel will receive the same configuration than the classical channel concerning the consumers. Its reference will be returned in an exception « new channel created, retry on this one » by the normal channel to the suppliers it can not cope with. When the load slowdown , the notification manager warns suppliers to return to the old reference. This can be done with the same exception mechanism. For the pull model, supplier must be replaced by consumer and vice versa. The load of the traffic will not decrease on the network, but the load will be shared by the normal channel, and its shadow counterpart(s).

In the same way, the COS trader could be coupled with the notification service to register subjects, and associated channels, in order to offer an alternative to the management access of the notification service.

It appears during the Alcatel's Notification Service V2 achievement that service advanced functions could be eased if the notification suppliers and consumers could offer an interface allowing some management, for example the shadow mechanism could be easily implemented with it. The supplier/consumer could be warned by the notification service state and even could receive instructions. Nevertheless, these additional interface must stay optional, in order to allows flexibility for notification service users.

## 5. Conclusion and future work

A Notification Service provides asynchronous multicast between CORBA objects. Compared to an Event Service, the notification service major additions are the support for filtering and some quality of service modes. Filtering allows to decrease the traffic which is an essential point for telecommunication applications and consequently for TINA services. Performance, quality of service and fault-tolerance are major considerations for a real-time system and thus for telecommunication software.

### 5.1 Features for a TINA Notification Service

We believe that the following service characteristics have to be considered for the design of a TINA notification Service:

*G***eneric an***d* **t***yp***e***d* **notification**. Generic and typed notifications provide the service user with a trade-off between a low-level mechanism with good performances and a higher level mechanism less error prone. In the second case, the cost in performance can be reduced by the use of predefined typed notifications.

*F***i***l***terin***g***.** Filtering should be at the channel level for performance and scalability purposes. It should be dynamic and incremental in order to provide the quenching mechanism.

**S***ub***scri***p***tion**. A high level API for subscription is not essential but it is important for an easy use of the service. The concept of subjects organised hierarchically is promising and needs to be developed and assessed.

***F*ed*eration***. Federation at subject level is essential for performance and scalability, and supports the network administrative domain concept.

***F*a*ul*t-to*l*erance**. Fault-tolerance is ubiquitous in industrial telecommunication applications. A customisable fault-tolerant Notification Service should be a building block for higher level fault-tolerance.

**Q*u*a*l*it*y* of service**. Quality of service is a basic feature in real-time systems and thus in telecommunication software. We still need to investigate which quality of service features are useful for a notification service dedicated to telecommunication applications.

**Notification *m*ana*g*er**. A notification manager which handles channel instanciation and deletion, notification description registration, subscription, federation, high level fault-tolerance and recovery provides to the user a set of automatic high level functions, simplifying the use of the service and centralising the service state. The second advantage of the notification manager stems from its ability to perform a part of the work, discharging channel objects of a part of the administration task. Channel tasks can be more concentrated on their main task of communication. This is a very valuable feature in the context of communication.

### 5.2 Future work

New work begins around Alcatel's Notification Service in the context of Vital and ReTINA ACTS projects. In addition to the completion of the actual features by for instance integrating the Notification Service with a logging service more advanced studies should be done.

As a service for reliability, the reaction of the service in case of critical situations such as the increasing of notification flow beyond the possibilities of the service need to be evaluated in the context of real telecom applications in order to refine the adequate solutions according to application's needs.

A second research direction for the Notification Service is its interfacing with transaction services and models. According to the integration with particular transaction models such as the open nested model or the queuing model different profiles of the Notification Services can be provided to the users according to different quality of service requirement for fault-tolerance.

The next step for telecommunication application is a real-time notification service integrating time constraints for real-time needs in communication.

## 6. References


[NEC et al. 97] NEC, BEA, Expersoft, Fujitsu. IBM. ICL, IONA, Oracle, TIBCO, Visigenic: « Notification Service Submission », OMG TC Document telecom/97-06-01, June 3rd, 1997.

[DSTC and Nortel 97] DSTC, Nortel Technology: « Notification Service ». OMG document telecom/97-05-02, June 2nd, 1997.

[GMD 97] GMD Fokus : « OMG Notification Service » OMG TC document 97.x.x June 3rd, 1997.

[OMG 94] The Object Management Group: « Common Object Services Specification ». V I 1.0. Framingham, MA, 1994.

[OMG RFP] OMG document: telecom/97-01-03.

[ReTINA] http://www.chorus.com/Research/retina.html

[Ruffin 92] Michel Ruffin: « KitLog: a Generic Logging Service ». Proceedings of the 11th Symposium on Reliable Distributed Systems, Houston, TX (USA), pages 139-146. October 5-7, 1992.

[TID and HP 97] TID, HP: « Joint submission to Notification Service RFP » Edition 2.0, June 1997.

[TMN] ITU-T, M3010, Principles for a Telecommunications management network

[VITAL] http://www.mari.co.uk/vital.

[X720] CCITT X720: « Structure of management information-Management information model ».

[X721] CCITT X72: « Structure of management information-Definition of management information ».

[X733] CCITT X73: « System management -Alarm reporting function ».

[X734] CCITT X734: « System management - Event report management function ».