Chapter 14

Supporting Service Quality Assurance via Trouble Management

Dr R. Sinnott, T. Gringel, Dr M. Tschichholz, W. Vortisch
GMD Fokus, Kaiserin-Augusta-Allee 31, 10589 Berlin, Germany

Key words: Trouble Management, Trouble Ticketing, Service Level Agreements, Service Quality Assurance, CORBA, TINA.

Abstract: The open service market encourages competition between service providers. To attract and keep customers, service providers require – amongst other things - better tools and techniques to increase their competitiveness. In this paper we address one area for tool support: namely, tools for the support of service quality assurance, i.e. so that checks can be made to ensure that services (and the networks they operate over) fulfil the expectations of customers who have subscribed to them. To demonstrate this, we show how trouble management techniques can be applied to develop generic and reusable components. The test-bed for this work is based on a TINA platform Y.TSP that has been extended with a trouble management component. We show how this trouble management component can be used to support service quality assurance via two application cases studies.

1. INTRODUCTION

The deregulation and subsequent opening-up of the telecommunications market offers many potential advantages to service providers and customers. Service providers now have the opportunity to compete in new markets, whilst customers are offered a wider choice of service providers and hence the services they offer. With this in mind, the need for service providers to be more competitive is paramount. This competitiveness can take many forms. It might be through the portfolio of services offered – including add-on services; through the costs associated with the access and usage of those services; through the quality of the services on offer etc. In this paper we address one area of competition: *service quality assurance*. Put simply, this

J. N. de Souza et al. (eds.), *Managing QoS in Multimedia Networks and Services*

is the ability to check that services that customers subscribe to fulfil their expectations. When this is not the case, e.g. due to problems (troubles) with the services, then appropriate actions are taken to resolve the problems and potentially, to offer some form of compensation to the affected customers. It is precisely this level of flexibility, i.e. offering compensation for problems that have occurred, that help to distinguish – and hence make more competitive! – different service providers.

As well as direct benefits to the customers through making the service providers more competitive, components that support the ability to monitor and potentially resolve troubles can also be applied to the direct financial advantage of the service provider. Thus for example, these same components can be applied to try to minimise any discounts that might have to be given to customers affected by troubles.

Troubles or failures more generally themselves can take many forms. It might be the case that complete hardware and software failures occur, in which case automated trouble management requires some form of backup, e.g. through contact phone numbers or addresses of people capable of resolving the problem. A less drastic form of failure might be through a certain feature of a service not working properly, e.g. the throughput of a service is not high enough. We shall see how trouble management techniques can be applied to cover a myriad of different failures affecting different systems using different technologies. We note that this incorporates failures that occur at different system levels, e.g. at the service level and network levels. What is crucial is that the information related to the troubles can be passed to the appropriate management systems and possibly the people or administrators responsible for resolving the problem.

It should be pointed out that in the open service market, integrating different systems offering potentially disparate trouble handling or fault management systems developed with different business processes in mind is a non-trivial matter. It is often the case that the work on integrating trouble management systems initially requires, work on integration of business processes. In this paper we focus predominantly on CORBA [2] based technologies applied to trouble management and do not address integration of business processes. Work in this area is documented in [6].

The rest of the paper is structured as follows. Section 2 gives an overview of trouble management and presents the main ideas behind trouble tickets and their application. Section 3 provides an overview of the architecture of the systems used to demonstrate trouble management. Section 4 focuses upon the service based trouble management components and discusses their design. Section 5 illustrates the application of the trouble management components via two cases studies: a music on demand service (MusicShop) and a image distribution service developed as part of a Dutch Auction flower

service. Finally we draw some conclusions on the work and identify future extensions.

2. OVERVIEW OF TROUBLE MANAGEMENT

Dealing with troubles that can potentially occur in different administrative and technological domains is a complex activity, especially for open systems, i.e. not simple interworking between a closed set of software elements. Ensuring that the necessary information is transferred to the appropriate components, or if necessary, to the people in charge of finding and identifying the trouble, is a complex activity. This is especially so where numerous different types of failure can occur with numerous causes and manifestations. To help provide a level of structuring to this open-ended problem, various standards [18], initiatives [15-17] and research projects [3,4,9] have taken place. A common approach taken by these has been through the notion of *trouble tickets*. Loosely, a trouble ticket is something that allows for the lifecycle of a trouble or failure to be monitored and acts as a basis for transferring the appropriate trouble information between different management systems. Figure 1 highlights the state diagram of a trouble ticket.

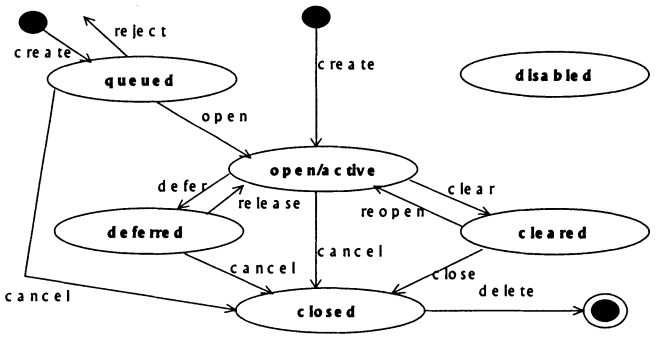

Figure 1. State Diagram for a Trouble Ticket

Trouble tickets can be created in state queued (where the trouble monitoring and resolution has not yet begun) or state open/active in which case the trouble is being worked on. A trouble ticket might be rejected if no such trouble exists. Once a trouble has been resolved, i.e. the failure has been resolved by a suitable person or fault management system, the trouble ticket moves to state cleared. It might be the case at this stage that the affected parties, e.g. the customers are requested to verify that the service or network is now functioning correctly – or at least that the trouble previously reported has been resolved satisfactorily. If this is the case then the trouble

ticket moves to state closed where the information associated with the lifecycle of that trouble can be written to a log. This might then be used for administrative purposes, e.g. to perform statistics on the troubles that have occurred and their causes etc.

Trouble tickets can also be disabled or deferred depending upon whether the information related to the trouble ticket cannot be updated for a particular reason, or the trouble itself cannot be dealt with at a given time.

It should be emphasised that disparate parties can be the instigators for the creation of a trouble ticket, e.g. a customer might complain about a service exhibiting a particular failure. Alternatively, it might be the service itself that informs customers about problems it is having – or will potentially have. Another alternative is that it is the network that the service makes use that might be the source of the initial trouble identification. The point is that troubles can occur anywhere and be identified by numerous sources.

The simplistic idea of trouble tickets lend themselves to supporting a generic and open approach for trouble management, i.e. they provide a common core functionality for a plethora of troubles to be realised, be they at the service or network levels. To achieve this, a standardised way to transfer trouble information is necessary. The ITU-T standard X.790 defines various managed object classes which support a broad spectrum of potential failures that might occur with ITU-T applications [18]. This document was used as a basis for the CORBA based SMART Customer to Service Provider Trouble Administration documents [15-17]. Essentially, these documents provide descriptions of the interfaces that should be supported by trouble management systems so that trouble information can be passed around successfully. The following diagram provides a snapshot of the functionality presented in the SMART document.

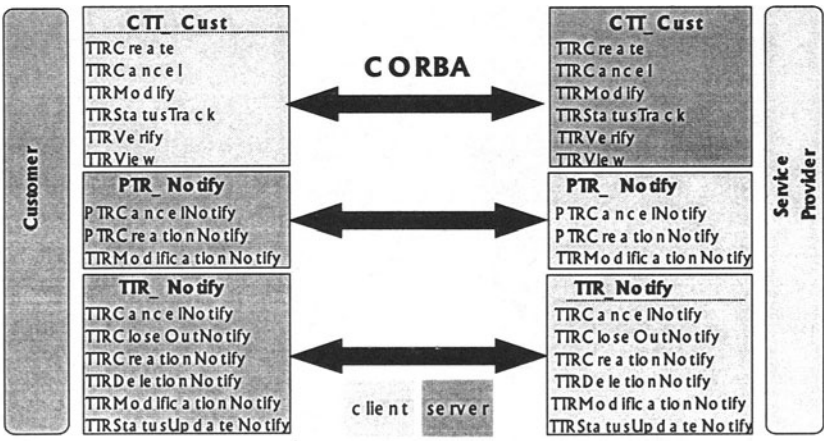

Figure 2. SMART CORBA based Customer to Service Provider Interfaces

These interfaces allow for the sending of the informations related to trouble tickets, e.g. to create trouble tickets (TTRCreate), to update the status of existing trouble tickets (TTRStatusUpdateNotify), to close existing trouble tickets (TTRCloseOutNotify) etc. Since troubles themselves can take a myriad of forms and involve numerous software and hardware components these interfaces typically possess many detailed lists of parameters. Through setting appropriate values for these complex sequences of parameters and providing appropriate information related to the troubles themselves, e.g. their nature and where they are occurring, appropriate trouble management systems can deal with the troubles.

Recent works have shown [1,20] how it is possible to build integrated inter-domain trouble management systems using these technologies. Specifically, the ACTS FlowThru project [5] has shown how gateways, e.g. for TMN-CORBA interworking, and generic trouble ticketing components could be built to support integrated trouble management. We note also that interworking of trouble ticketing systems using TMN technologies was also undertaken in the Eurescom Project P612 [3,4]. Our emphasis here is on one of the CORBA based trouble ticketing components that was developed in the FlowThru project: the TINA Trouble Ticket Reporting System (TTRS). In particular we show how this component has been integrated within the GMD Fokus Y.TSP platform [19] so that it can deal with troubles that might occur with services and networks, and with the consequences of those troubles.

3. TINA ARCHITECTURE AND Y.TSP PLATFORM

The TINA architecture [9] defines a set of concepts, principles, rules and guidelines for constructing, deploying, and operating CORBA based services. The major principles are based on the Reference Model for Open Distributed Processing [8]. The purpose of these principles are to insure interoperability, portability and reusability of software components; independence from specific technologies and to share the burden of creating and managing systems among different business stakeholders, such as consumers, service and connectivity providers [18]. Reference Points are defined to specify conformance requirements for TINA products [13].

TINA provides a set of specifications, e.g. Computing Architecture, Distributed Processing Environment Architecture, Service Architecture and Network Resource Architecture [10-14]. The Service Architecture introduces the underlying concepts and provides information on how telecommunication applications and the components they are built from, have to behave. Central to the Service Architecture is the concept of a *session*. Three sessions are identified: the **access session** which represents mechanisms to support access to services (service sessions) that have been

subscribed to; the **service session** which includes functionality to execute and control and manage sessions, i.e. it allows control of the communication session; the **communication session** which controls communication and network resources required to establish end to end connections.

Various components have been identified as being useful in the different sessions. Of relevance here are the access session related subscription and accounting components. The accounting component is used to store the charges/discounts a user incurs when using a given service. This includes the service and network charges. The subscription component is used to store customers subscription information, e.g. the services they have subscribed to and the agreements they have on the usage of those services.

The access session user application (asUap), provider agent (PA), initial agent (IA) and user agent (UA) enable service selection and secure service access in a TINA environment. The relation between these and the service session components is given in Figure 3.

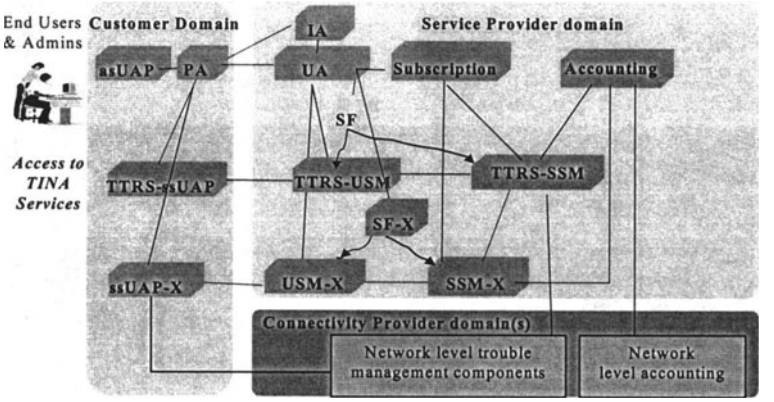

Figure 3. Description of the Y.TSP Components

The service session components typically consist of a service factory (SF) which is used for starting instances of services ofen represented as a service session manager (SSM) used to control the global logic of the service and one or more user service session managers (USM) used to control each users participation (and state) in that service. Users interact with the service itself through some service session specific user application (ssUAP). We note that the Y.TSP platform effectively provides a complete infrastructure in which services - themselves represented as service session components - can be deployed and used with minimal integration effort.

Included in the above diagram are the service session related components for the trouble ticket reporting system (TTRS). This is the component used to facilitate trouble management in the Y.TSP platform. The TTRS service

can be subscribed to as with any other service deployed in the platform. Once subscribed to however, this service is started automatically once a subscriber starts their access session. This enables users to receive notifications of problems with services or networks affecting them immediately, i.e. before they try to use the service or network respectively.

4. OVERVIEW OF THE TTRS COMPONENT

To provide any form of automated or semi-automated trouble management requires consideration of numerous issues. One of the key issues considered in developing the TTRS component was that it should, as far as possible, be independent of the services themselves. Of course, if the TTRS component is to help in performing automatic trouble resolution, i.e. without the need for human intervention, then this implies that the TTRS component and the service - or more typically the fault management part of the service - itself interwork to some extent.

Whilst it is true that the resolution of the particular troubles themselves is service specific, the nature of the interactions between the TTRS component and the different services is more generic. Specifically, the requirement is that the service supports some form of interface that allows for the exchange of trouble information. Put simply, the role of the TTRS component is to act as a central location where informations on troubles can be recorded, discovered and directed to the appropriate entities, e.g. fault management systems or other potentially remote trouble ticketing systems. The resolution of the problem itself - should it exist! - is up to the appropriate fault management system of the service. The architecture of the TTRS component is given in figure 4.

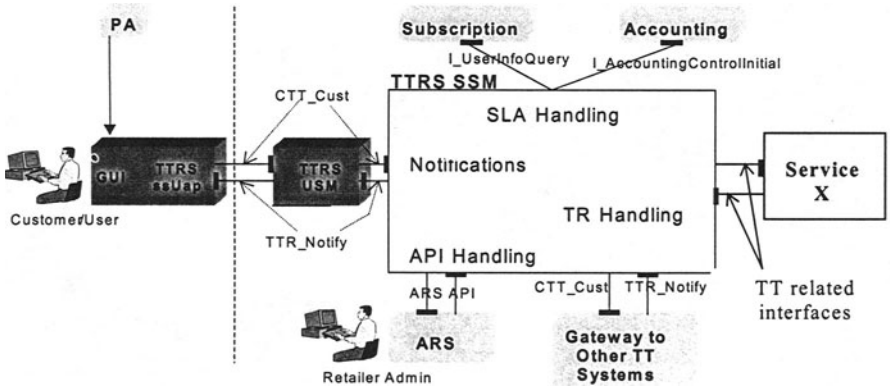

Figure 4. Architecture of the TTRS Component

The CCT_Cust and TTR_Notify interfaces depicted in Figure 4 are identical with those given in section 2. That is, the TTRS component implements the SMART IDL. More precisely, the SMART IDL has been implemented so as to provide a CORBA wrapper around a commercial trouble ticketing system: the Action Request System from Remedy [7]. This has been used predominantly as a database for trouble tickets, however it is quite possible to use it to perform other activities. For example, to send emails to users affected by troubles who do not currently have an active session. In addition the ARS was used to design the templates for trouble tickets held in the database. We consider the interface between the TTRS component and two example services in section 5.

The user interface to the TTRS service (the TTRS ssUAP) is itself represented as a traffic light with an additional window for writing trouble tickets or showing the information associated with existing troubles. The TTRS GUI traffic light is green when no troubles exist that affect this user; red when a trouble does exist that affects this user and yellow when troubles exist that need to be verified, i.e. the trouble has been resolved but that the user needs to verify that this is the case.

The TTRS component also interacts with subscription and accounting components. As stated previously, offering trouble management facilities in combination with service discounts for troubles incurred makes a service provider more competitive. To realise this the subscription component of the Y.TSP platform, has been extended to support service level agreements (SLAs). SLAs correspond to agreements made between customers and service providers upon the expected quality of the services subscribed to. When problems occur and are subsequently resolved (or possibly during the resolution process!), checks on the SLAs for the affected users are made, and if violated, can result in discounts being issued for the usage of that service. The most general SLAs common to all services are found in table 1.

Table 1. Generic SLA Properties

Property Name	Property Type	Range	Description
Service Availability	short	0...100	% Svc availability per report period
Activity time from	string	"0:00-24.00"	Daily service availability begin
Activity time to	string	"0:00-24.00"	Daily service availability end
Reporting period	short	28..31	Reporting period in days of month
Max Time to repair	float	0...∞	Max time (mins) for resolution
Time between failures	float	0...∞	Min time (mins) between failures
Minimum Errors	short	0.. ∞	Minimum number of errors allowed
Reliable Mode	Boolean	True/False	Optional transport mode supported
Maintenance time	float	0...∞	Maintenance time (mins) per report period

We note here though that violations of SLAs take two major forms: event based or non-event based. An example of an event based SLA is

TimetoRepair. Thus when a trouble occurs and subsequently resolved a check can be made immediately to see whether a violation has taken place. Service availability on the other hand checks for violations over some time period, e.g. a month. As such once a particular trouble has been resolved it will not necessarily lead to an immediate check for a violation.

We point out that certain of the SLA properties support various levels of violation. For example, when set by a subscriber the *TimeToRepair* property is given as a range of integer values, e.g. < 0, 1, 5, 10, 20, 60 >. Thus when a trouble occurs, the longer the trouble takes to be resolved the larger the violation that takes place. This in turn is used to influence the magnitude of the discount that is given. Thus here for example, if the service is fixed within 59 seconds then no violation takes place. If it is fixed within 61 seconds however, then a level-1 violation takes place etc. If it takes over 1 hour to repair then a level-5 violation takes place etc.

5. TROUBLE MANAGEMENT CASE STUDIES

5.1 The MusicShop Service

The MusicShop service has been developed and implemented as an e-commerce service integrated in the Y.TSP platform. It provides customers (musicians and consumers) with a multimedia store and retrieval system using Web-based technologies. It is implemented using the Oracle Application Server and the Oracle 8 RDBMS together with appropriate CORBA/Java technologies. The culmination of these technologies help provide a scaleable, reliable and manageable platform for hosting shared network applications such as data-driven multimedia content.

Musicians can upload music (or other media types, e.g. text, graphics etc) which can subsequently be downloaded by consumers. Musicians are charged for storing their music and general administration of their accounts, e.g. adding, deleting, renaming music folders etc, whilst consumers are charged for downloading music. The MusicShop service also supports a fault management interface which is used for transferring information related to trouble tickets associated with the MusicShop service. Specifically, the MusicShop supports a subset of the TTR_Notify interface described previously so that it can receive information on trouble tickets that have been created by TTRS due to potential faults with the MusicShop service. In addition, the fault management interface of the MusicShop service can be used to issue notifications about troubles it is currently having to the TTRS service via an interface offering a subset of the CTT_Cust interface.

Several scenarios have been developed to highlight the usage of trouble tickets in supporting service quality assurance using the MusicShop and

TTRS services. These scenarios illustrate both how trouble management systems can be applied for normal trouble resolution as well as highlighting how discounting for troubles incurred can be achieved. Examples of failures include the stopping of the http-listener so that the service effectively will not respond when the user attempts to interact with it or through the database being unavailable. We consider the former scenario.

To begin with we assume that the http-listener has been killed. Once the user has identified that the service is down they request that a trouble ticket is created using their TTRS-GUI. In this request they include the trouble object (MusicShop service), the trouble type (not responding) and other information that they might think is useful to the resolution process.

Upon reception of the request TTRS checks with the subscription component to check if this user subscribes to that service, and if so finds other affected users. The subscription information including any SLAs that are set for the affected users of MusicShop are returned. TTRS then creates trouble tickets for the affected users. Particularly important at this point is that a record of the time at which the trouble was initially identified is taken.

TTRS then issues a notification to the fault management part of MusicShop stating that it has created trouble tickets to a *not-responding* trouble with MusicShop. The fault management system then checks if such a problem exists, and issues an appropriate notification if it finds the trouble and begins working on resolving it. At this point TTRS sends notifications to all customers of the MusicShop service indicating the problem with the MusicShop service, thereby making their TTRS GUI go from green to red. The identity of the trouble tickets is also passed over at this time and can then be used for querying the state of the trouble by the user.

Once the fault management system of MusicShop has resolved the problem, i.e. the http-listener has been restarted, it sends an update to the TTRS component. Based upon the parameters of the notification, the TTRS informs customers that the problems is resolved and that they do/do not have to verify that it now works correctly, i.e. it now responds. At this point their TTRS GUI goes from red to yellow/green respectively. The time at which the problem was resolved by the fault management system is recorded. Once the users verify that the trouble no longer exists, the trouble tickets kept by TTRS for this particular problem are closed and the TTRS GUI shows green.

Once the trouble tickets are closed, TTRS checks whether any of the SLAs for the affected users have been violated. In this scenario, the *TimeToRepair* parameter is of particular relevance. The time difference between the trouble resolution and trouble identification times is taken. If this exceeds the value for an SLA limit as determined by the subscription information, then appropriate discounts are given, i.e. discounts are sent by TTRS to the accounting component indicating the reason for the discount.

5.2 The Dutch Auction Service

The Dutch Auction service implements a distributed online version of the real Dutch Auction which takes place daily in Aalsmeer in the Netherlands. Each day, the world biggest flower auction is held with flowers brought by farmers where they can be auctioned immediately.

The service itself supports two user roles, namely one *mediator* and an arbitrary number of (flower) *dealers*. The mediator (or auctioneer) controls the selling of the flowers. In contrast to an ordinary auction, where dealers bid with progressively increasing prices, a mediator's clock decrements the price until one of the dealers accepts the offer. At this point, all dealers clocks are stopped and reset in preparation for the next flower lot.

As a companion service to Dutch Auction, the Image Distribution Service (IDS) has been developed. This service allows dealers to get an image of the flowers currently being auctioned. Specifically, the mediator sends flower images to dealers currently using the Dutch Auction service.

The Dutch Auction and Image Distribution Service have stringent requirements on the transmission and delivery of the information they deliver, i.e. the bids and images respectively. These services need to ensure that the bids or images being transferred, are done so in a concurrent and timely manner, e.g. to ensure that all dealers have equal opportunity to bid for flower lots or to see images of the flowers. In this regard, the Dutch Auction service makes use of IP multicast and RSVP, the Internet Resource reSerVation Protocol. The result is the provision of a quality of service enhanced binding which allows a bandwidth guaranteed message transfer for, in this case, price bid information. The Image Distribution Service itself makes use of a Lightweight Reliable Multicast Protocol. It is important to note that this protocol can operate in both reliable or unreliable modes. Figure 5 illustrates the configuration of the image distribution service and the TTRS component.

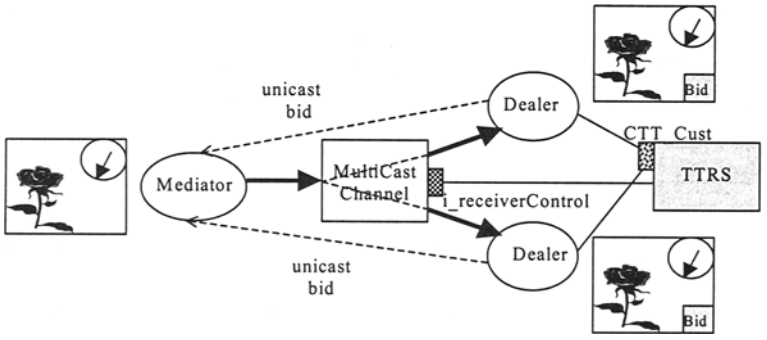

Figure 5. Configuration of Image Distribution Service and TTRS

Here the dealers receive images of flowers via the IP Multicast and make bids for them using a unicast connection. Normally the image distribution service runs in unreliable mode, however, should errors occur, e.g. due to too much network traffic with the connection between the mediator and the dealers, then it is possible to switch to (the more expensive) reliable mode. We note that errors are manifest as a lack of flower image.

In this scenario, requests to create trouble tickets for dealers suffering from connectivity problems, i.e. lack of flower image, are automatically sent to the CTT_Cust interface offered by TTRS. The parameters included in this are the trouble object (IDS), the trouble type (lack of image), and an interoperable reference (IOR) to the i_receiverControl interface. The TTRS component then interacts with the subscription component to see if a violation has occurred for this particular user. In this scenario the violations that are of interest are based upon the number of errors that have occurred and the time period over which they have occurred. Thus a dealer suffering from bursty types of failures, i.e. perpetual loss of images over a particular time period, can, provided they have selected the *true* option for the SLA switching to reliable mode parameter, switch transport modes. This is achieved through interacting at the i_receiverControl interface and changing the transport mode. This interface offers a single oneway operation that allows to set the different transport modes. Users who do not subscribe to this transport mode switching SLA parameter, are offered the option of upgrading their SLA values.

In comparison to the MusicShop-TTRS scenario various points are worth noting about the IDS-TTRS scenario. In this scenario the trouble tickets recorded in TTRS effectively have no useful history. That is, they do not require opening and clearing etc. Rather, here the trouble tickets act predominantly as timed counters which can be used to check the burstiness of recorded troubles. When a particular user has experienced more errors in a particular time period than that given in their SLA parameters, then the TTRS component automatically switches their mode for them.

Also in this scenario, the users troubles are isolated and largely independent of one another. Thus it is not the case that the request to create a single trouble ticket can result in TTRS creating many trouble tickets, e.g. for all users who subscribe to this service, and have to maintain the state information of all of them.

6. CONCLUSIONS

This paper has argued that to attract and keep customers, service providers require – amongst other things - better tools and techniques to increase their competitiveness. We have presented one example of such a

trouble management tool together with case studies to illustrate its functionality. We have shown how this tool can be applied both to help in the trouble resolution process and as a basis for offering discounts to customers. The availability of such tools is increasingly necessary given the overlap between services and the networks they make use of, and the associated increase in complexity in the trouble management process.

The applicability of such tools to help minimise costs incurred is especially relevant to service providers. These cost minimisations can take several forms and offer the potential for numerous optimisations. For example, it is possible through such tools to attempt to avoid SLA violations from occurring, e.g. by prioritising the trouble resolution process for troubles that will result in a violation. Of course, a reduction in the trouble resolution process time is both beneficial to the service provider – so that more revenue can be generated, as well as to the customers themselves, since their services are available more often.

REFERENCES

[1] N. Agoulmine, D. Dragan, T. Gringel, J. Hall, E. Rosa, M. Tschichholz, Trouble Management for Multimedia Services in Multi-Provider Environments, Journal of Network and Systems Management, March 2000.
[2] Object Request Broker 2.0. Object Management Group, John Wiley & Sons, 1995.
[3] Covaci, S.; Dragan, D. Customer Care Solutions: Interoperable Trouble Ticketing Management Service, Proceedings of IS&N'97, Cernobbio (I), May 1997.
[4] Covaci, S.; Marchisio, L.; Milham, D. J., Trouble Ticketing X Interfaces for International Private Leased Data Circuits and International Freephone Service, Proceedings of 6th NOMS Conference, New Orleans, USA, February 1998.
[5] www.cs.ucl.ac.uk/research/flowthru/
[6] D.Lewis, A Development Framework for Open Management Systems; Interoperable Communication Network Journal, Baltzar Science Publishers, Q1, 1999.
[7] Remedy Corp - Part Number GSG-320-001, "Getting Started Guide"
[8] Reference Model of Open Distributed Processing, ITU-T Recommendations X.901..4 : ISO International Standard (DIS) ISO/IEC 10746-1..4, 1995.
[9] www.tinac.com
[10]H. Mulder (Ed.), TINA Business Model and Reference Points, Version 4.0, May, 1997
[11]A. Gavras, W. Takita, TINA DPE Architecture, Version 2.0b0, November, 1997
[12]F. Steegmans, TINA Network Resource Architecture, Version 3.0, February, 1997.
[13]P. Farley, R. Minetti, TINA-C Ret Reference Point Specifications, v1.0, Jan, 1998
[14]L. Kristiansen, TINA-C Service Architecture, Version 5.0, June, 1997
[15] TMF, Service Provider to Customer Performance Reporting Business Agreement, Issue 1.0, March 1997
[16]TMF, Customer-Service Provider Trouble Administration Information Agreement, Issue 1, March 1997
[17] Corba Interface Specification for Customer to Service Provider Trouble Administration, Revision 0.2, TM Forum, SMART Group, November, 1997
[18]Trouble Management function for ITU-T Applications – ITU-T Rec. X.790

[19]www.fokus.gmd.de/research/cc/platin/
[20]D. Dragan, T. Gringel, J. Hall, R. Sinnott, M. Tschichholz, W. Vortisch, Integrated Trouble Management to Support Service Quality Assurance in a Multi-Provider Context, Proceedings of 7th IS&N Conference, February 2000, Athens, Greece.