Application Server Architecture for Open Networks

Christos Solomonides and Mark Searle
University College London
{c.Solomonides, m.Searle}@ee.ucl.ac.uk

Abstract: *A key element to API technologies is the gateway. This is a Server architecture that provides the interface between client requests and the network services. Whilst it is in the interests of simplicity and ease of implementation for the client server interface to be as straightforward as possible, special consideration must be given to the interface to maintain security, integrity, scalability and general manageability of what is essentially a fragile access to precious network resources. This paper presents an initial architecture for an Application Server (API Server).*

1. Introduction

The approach taken by advocates of Application Programming Interfaces (API) such as Parlay is to open up access to the various protected functional entities (FE) of operators to third party service providers, through the use of carefully defined class libraries. The opening up of FEs such as switch functions and service control points is of enormous importance to third party software developers.

Application Programming Interface (API) technologies are being developed to support the emerging heterogeneous network-of-networks environment. Such networks are being developed in an environment that is highly chaotic in the sense that the demands placed on networks are changing rapidly. This also has the effect of denying network and service designers a clear means of extrapolation to future requirements. It also makes it undesirable or even impossible to plan for supporting architectures whose standardisation efforts and complexity force lead times that run into years. It is instead desirable to support key technologies that allow the fast and flexible deployment of new services. API technologies such as Parlay [1] were initially developed with telecommunication applications in mind, partly as an attempt to allow the telecommunication domain to respond to this environment. It is clear though that the API is not limited to the telecommunications domain. In fact APIs are all the more relevant to the IP domain where the need for rapid service creation tools is all the more pressing.

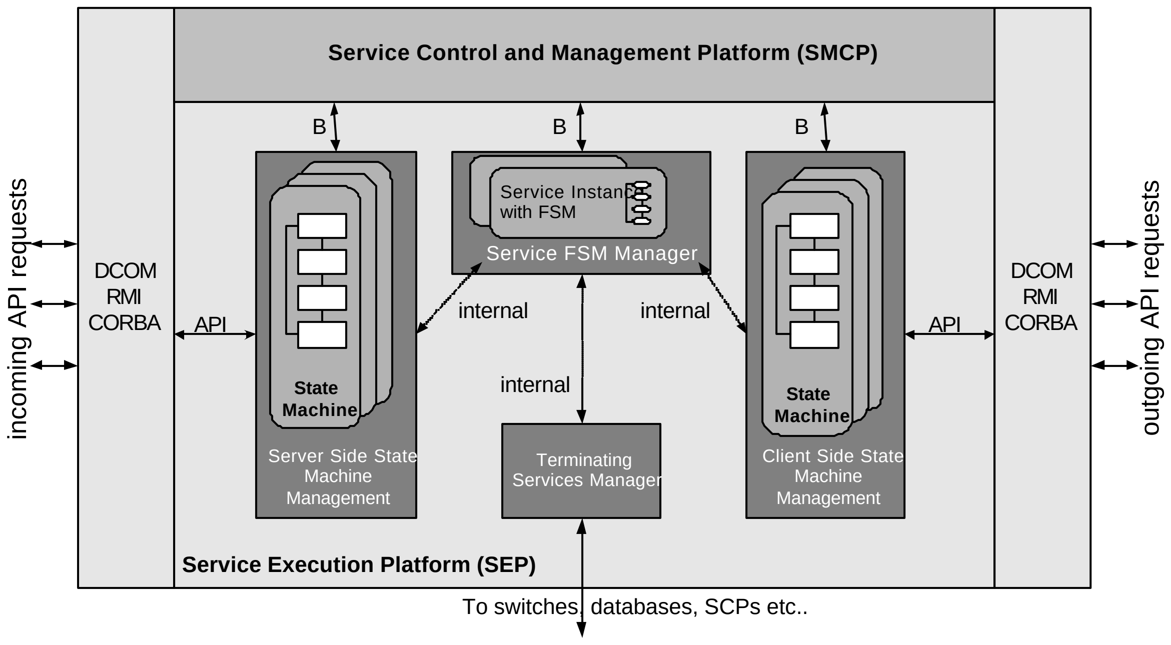
The move towards providing open access to the core functionality of the network is likely to lead to a rapid growth in the number of new services because it provides an open set of programming libraries that can be viewed as a kind of 'HTML' for control. Through these libraries it is possible for third parties to develop applications by building software from the blocks provided by multiple operators. This should allow third-party service providers the means to access core network functionality in a secure and resilient manner. Another example of an API is that developed through JAIN (APIs for Integrated Networks) [2]. JAIN aims to provide a lower level of control aimed at open access for signalling services such as the Intelligent Network Application Part (INAP) and SS7 user parts. Parlay and JAIN are complementary.

2. The API Server Architecture

The concept of state models and state model behaviour in telecommunications has already proven essential to management and network intelligence applications of the core network. A specific area in telecommunications, which makes extensive use of state models, is that of the Intelligent Network (IN) [3]. In particular the basic call process and its state model, the Basic Call State Model (BCSM) is possibly the most important state model in telephony, upon which the majority of IN services are based. The IN Capability Set 2 (CS-2) [4] through Call Party Handling (CPH) allows the fine-grained manipulation of a call (specifically a call leg) as well as more complex operations such as the merging and splitting of call segments. Of course this is achieved at the "expense" of very complex (and well defined) interactions between the switches involved.

The authors believe that the CPH capabilities of IN CS-2 provide a powerful way to describe the behaviour of complex multi-party and multi-service interactions. They have incorporated these ideas into defining a similar mechanism in the API Server. One of the main advantages of such a state model is the resilience and the proven track record of the IN model. Of course a state-model approach imposes overheads in terms of managing the model, however, it is critical to maintain a guaranteed level of service for such an API Server. It also makes clear how distributed applications interact with each other.

Figure 1 depicts the Architecture of the API Server. The server consists of two parts; The Service Execution Platform (SEP) and the Service Management and Control Platform (SMCP). Access to the server is made possible through distributed object technologies such as DCOM/RMI or CORBA. This paper focuses on the architecture of the SEP platform.



Service Control and Management Platform (SMCP)
B
B
B
Service Instance with FSM
Service FSM Manager
incoming API requests
DCOM
RMI
CORBA
API
State Machine
Server Side State Machine Management
internal
internal
internal
Terminating Services Manager
State Machine
Client Side State Machine Management
API
DCOM
RMI
CORBA
outgoing API requests
Service Execution Platform (SEP)
To switches, databases, SCPs etc..

Figure 1: API Server Architecture

The Service Execution Platform (SEP) contains all the functional elements necessary for the execution of service logic. There are two state models, one for server requests and one for client requests. The Server-Side state machine manager (S_SSM) is responsible for handling incoming requests. These cause instances of services to be created and are managed using the CPH approach adopted by IN CS-2. Where requests need to be forwarded, a Client-Side state machine (C_SSM) is created. Both the Server and Client SSMs are based on the IN BCSM. The Terminating Services Manager handles incoming requests that can be served without having to forward supplementary service requests.

In addition to the Server and Client SSMs the API Server requires IN-type FSMs in order to accomplish a CPH-like behaviour. The IN_FSMs contain the Service Segments and Service Segment Associations. An S_FEAM creates new Service Segments (SS) and Service Segment Associations (SSA) in a similar manner to the CPH model. An Inter-SSM Interface (ISI) interface allows the communication between originating and terminating BCSM-type models.

2.1 The Server and Client State Models

When a new service request is received a service instance is created, together with the service policies. The server-side state machine SSM monitors the activities of the server in the execution of a service. The state machine goes through the lifecycle of instantiation, message passing and termination/billing. The state model also implements points in call (PIC) to allow the triggering of other services. The state machine is thus given some flexibility for manipulation similar to the case of IN event and trigger detection points, although the supplementary services that this might facilitate are as yet undefined. The client-side state machine represents the client in the instantiation and message-passing phase of a service. The client-side is not essential in all cases (for example, when a service request is a terminating service).

Information held as part of a service policy includes supporting and conflicting services (for providing information on feature interaction), service order execution (necessary in situations where services need to be executed in a specific order), and service timeout policies (required in situations where the reply from a service may be needed for further processing). The service policies also contain information, which indicates whether the particular service needs to initiate supplementary services (i.e. dependant services).

3. Interoperability and Standardisation Issues

The authors believe that there is a delicate balance between the needs of standardisation and the requirement to simplify the technology. The more complex the technology the more complex will be the standardisation process. The level of Interoperability between API Servers clearly has a direct impact on the level of inter-working that can be achieved among service providers. It is clear that unless there is a strong move towards standardising this interface at the Services Layer, complete interoperability will not be achieved. It does not mean however that there are no in-between scenarios.

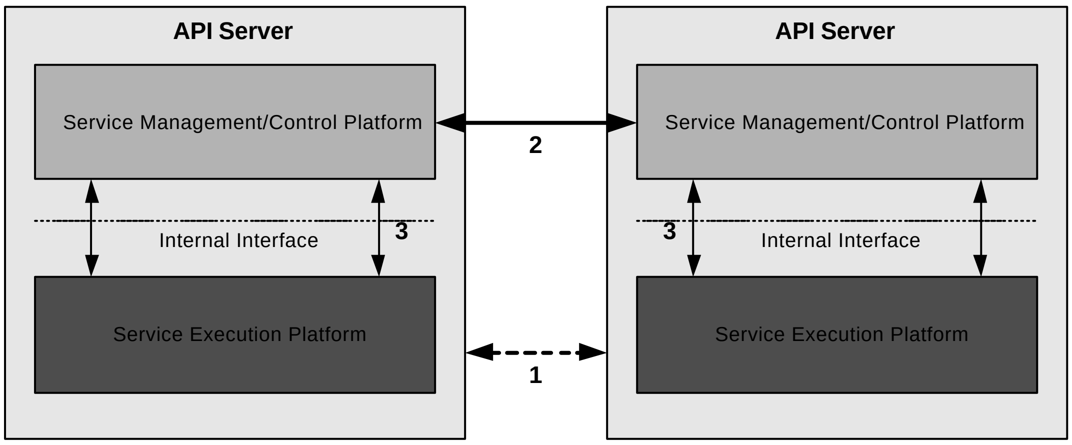

API Server
Service Management/Control Platform
Internal Interface
3
Service Execution Platform
API Server
Service Management/Control Platform
3
Internal Interface
Service Execution Platform
2
1

Figure 2: Potential Levels of Common Interfaces

Figure 2 depicts the different levels of interoperability. Interface 1 identifies the general scenario where a common interface is achieved between the API Servers. Under this scenario, the internal interface (3) is clearly proprietary. The second scenario tries to define a common interface among the Service Management/Control Platforms, through which peer API Servers have limited access to the Service Execution Platform. However, important aspects such as remote service management, on-the-fly subscription to new services as well as allowing remote control of the service execution environment can be provided. For example a SP could request the termination of an executing service in a remote API Server. This would only be allowed if the party requesting the termination of the service is the same party who initiated its execution. Of course such a termination request needs to propagate to lower API Servers. Another service that can be provided across the same interface is a simulation service. For example a remote API Server could request that requests, following a simulation request, not be executed on the core network. A benchmarking service could also be provided which would allow the testing of delay and performance of links across interconnected API Servers. Of course it is clear that a common interface across the SMCP, implies that some requirements are imposed on the internal interface in order to achieve that functionality.

4. Conclusions

Opening up the core network would lead to a rapid growth of new services as well as business opportunities for Third-party service providers. While a lot of work is currently focused at protecting the core network (and for very good reasons) it is also important to provide a framework that would protect the service providers. This paper has identified one possible approach, which could be adopted by introducing an element at the Services Layer, called the API Server. Its aim is to provide a structured way for SPs to inter-work. The architecture presented borrows heavily from IN principles. The reasons for this have already been discussed, however it is important to re-emphasise the proven track record of state models as means of controlling concurrency in real time systems. While this work is still in its infancy the authors believe the current approach could provide part of the solution for controlling interactions among Service Providers.

5. References

[1] The Parlay Group, www.parlay.org
[2] JAIN – "APIs for Integrated Networks",

http://java.sun.com/products/jain/index.html

[3] ITU-T Recommendation Q.121x Series on Intelligent Networks for CS-1, Geneva 1995.

[4] ITU-T Recommendation Q.122x Series on Intelligent Networks for CS-2.