Abstract – Rapid yet diverse service creation and deployment will ensure customer acceptance of the Next Generation Network (NGN). Softswitch architectures, such as Parlay, utilise service logic decomposition to achieve this. These architectures decompose service logic into service-dependent and service-independent parts. In this paper, service logic decomposition is applied to the functionally rich TINA service components resulting in generic and re-usable software sub-components that enable rapid service creation. The benefit of this decomposition approach is the ability to re-use service logic during service creation thus ensuring rapid deployment of third party services in the NGN or TINA compliant networks.

Index terms – Next Generation Network, Parlay API, TINA, service composition, service creation and deployment.

I. INTRODUCTION

The Next Generation Network (NGN) is a multi-service bearer network capable of supporting multiparty, multimedia, real-time and information services. Commercial success of the NGN is seen to rely on the ability to attract multiple service providers, outside of the telco’s domain, offering services that use the telco’s network [1]. This competitive environment makes rapid service creation and deployment essential. Architectures for service control and management have been developed to support the above business model, most notably Parlay and TINA.

The Parlay Group defines a rich set of open APIs enabling service provision by third party service providers [2]. The Java APIs for Integrated Networks (JAIN) also defines an API, which makes use of proprietary components like Java Call Control (JCC) and protocol adapters. The decomposition of service logic into service-dependent and service-independent parts results in these APIs. TINA on the other hand, provides functionally rich re-usable service components suitable for multiparty, multimedia service creation and provision in Distributed Processing Environment (DPE) based networks. These functionally rich components however, encompass both service-dependent and service-independent logic making it difficult to identify re-usable software below the service component level. Additionally, the creation of TINA services is a time consuming complex task requiring specialist knowledge of TINA’s service architecture.

In this paper we propose that a combination of aspects of Parlay and TINA will facilitate rapid service creation and provision in the NGN. This combination of Parlay and TINA enables the identification of re-usable software below the service component level, thus ensuring rapid service creation and provision. Section II describes the NGN business model and focuses on service composition. Section III discusses the Parlay API and illustrates the decomposition of service logic via the use of a simple service example. Section IV presents the implementation of re-usable TINA service components based on this decomposition, followed by the conclusion in Section V.

II. NGN SERVICE ARCHITECTURE

We assume an underlying Quality of Service (QoS) enabled packet switched network and concentrate on the service architectural layer. We envisage services, call and connection control in the NGN to be implemented in application servers and softswitch platforms based on a DPE. Service providers provide services directly to customers in a manner consistent with the NGN business model.

A. NGN Business Model

Figure 1 shows the envisaged NGN business model supporting the rapid creation and deployment of services. In this model the network operator is a combination of connectivity provider and TINA retailer. The stakeholders in this model fulfil the following functions:

1. The connectivity provider provides the retailer access to network resources. The retailer is able to manipulate network resources using call and connection control.
2. The network operator provides to service providers the ability to utilise this call and connection control in service provision.
3. Service providers are responsible for service creation and provision directly to the customer.

---

1 This work is supported by Telkom SA Limited, Siemens Telecommunications and the THRIP programme of the Department of Trade and Industry.
This model requires relationship 2 in figure 1 to be secure yet open to multiple service providers [3]. Additionally, the network operator’s presentation of call and connection control functionality to the service provider must be transparent of underlying networking technologies, easy to use, quick to implement and most importantly standard for all service providers.

The NGN business model is composed of four stakeholders common to both the Parlay and TINA business models. Parlay enables the service provider to offer services directly to the customer while TINA, views the service provider as the provider of services to the retailer, which then offers these services to customers. We adopt the NGN business model because it exploits the advantages of both the Parlay and TINA business models. The NGN business model enables the rapid creation and provision of services by service providers utilising the re-usable call and connection control capabilities offered by the retailer or network operator.

B. Service composition in the NGN

Examples of services that might be provisioned in the NGN include Video on Demand (VoD), Streamed Audio Conferencing (AC) and Shared Whiteboarding (SWB). We envisage these services to be hierarchical in nature. Complex NGN services developed by service providers are composed of simpler services which themselves are composed of even simpler services. The lowest level of this hierarchy encapsulates service-independent logic that is generic, stable and re-usable.

Figure 2 shows the composition and hierarchy of services in the NGN. Each node in this figure represents a service. In the service provider’s domain, complex NGN services that are composed of simpler services are developed and executed. Complex NGN services, for example VoD, AC and SWB, encapsulate the service-dependent logic that represents the capabilities of the services provisioned to the customer.

![Figure 2. Service Composition in the NGN](image)

The simple services are re-usable software sub-components that encapsulate service-independent logic. Simple services are combined to create more complex NGN services. The VoD and SWB services both require an authentication mechanism and we call this mechanism a simple service. The lowest level of this hierarchy encompasses the generic, stable and re-usable service-independent logic for session or call control, service management and connection control. The service-independent logic in the network operator’s domain supports different capability levels but are nonetheless the same for any service. The call control logic for example, could have different capability levels supporting services requiring simple single party call or session control to complex multiparty, multimedia call or session control. The SWB service in figure 2 requires multiparty, multimedia call or session control while the VoD service requires only simple two party call or session control. The Parlay API presented in the next section utilises the separation of service logic into service-dependent and service-independent parts to enable rapid third party service provision.

III. Characteristics of the Parlay API

The Parlay Group defines a standardised set of open APIs enabling service creation and provision by multiple service providers. The Parlay API is composed of two types of interfaces, namely Service Interfaces or Service Capability Features (SCFs) and Framework Interfaces [4]. These interfaces reside in Parlay gateways in the network operator’s domain. The next sub-section elaborates on the Parlay Group’s definition of these interfaces.

A. Parlay API Architecture

Each SCF is a logical grouping of interfaces that provide the core functionality useful to Parlay Applications i.e. third party NGN services. The SCFs perform functions in response to operations issued by the Parlay Application or third party NGN service, for example, the routeReq() operation in the Generic Call Control (GCC) SCF routes a call to the destination specified by the Parlay Application. The GCC SCF has tiered functionality from basic call control to complex multiparty, multimedia call control [5]. This tiered functionality supports the creation and provision of simple two party services to more complex multiparty, multimedia services and is analogous to the capability levels we envisage NGN services to be composed of. The Parlay API defines other SCFs such as Messaging Management, Mobility Management and User Interaction.

The Framework Interfaces provide support functionality useful to both third party NGN services and SCFs. The Framework Interfaces include Trust & Security Management, Service Discovery, Event Notification, Service Subscription, Service Lifecycle Management and Integrity Management. The next sub-section uses a simple number translation service to illustrate interactions between third party NGN services or Parlay Applications and SCFs.

B. Interaction between NGN services and SCFs via the Parlay API

Figure 3 shows a message sequence chart for a simple NGN number translation service, illustrating, the use of callback interfaces and the decomposition of service logic. The number translation service uses the GCC SCF to perform simple two party call control. In figure 3, App Logic is an abstract representation of the NGN number translation service. Each entity in this figure represents instances of the computational objects defined by Parlay’s GCC SCF.
The following messages are used in the provision of the NGN number translation service:

1. App Logic requests the creation of an object implementing the AppCallControlManager (App CCM) object by invoking the new() method. App CCM passes its interface reference to App Logic in the return parameter of this message.

2. App Logic then invokes the enableCallNotification() method on the CallControlManager (CCM) in the Parlay gateway. Since the service is a number translation service, the enableCallNotification() method sends a reply to App Logic in the event that a number requiring translation has been dialled. In this message the callback interface that CCM uses to inform App Logic of the event is passed via the in parameter i.e. the interface reference to App CCM (CCM’s callback interface).

3. When the number requiring translation has been dialled, CCM invokes the callEventNotify() method on its callback interface App CCM.

4. The callEventNotify() message is passed to App Logic.

5. App Logic requests the creation of a new object implementing AppCallObject (App CO). The return parameter of this message contains the interface reference to App CO.

6. App Logic translates the number.

7. App Logic invokes the routeReq() method on the CallObject (CO) in the Parlay gateway and passes CO the translated number and App CO’s interface reference in the in parameter of this message. CO is part of the GCC SCF and is responsible for call control. CO uses this number to route the call to the destination.

8. CO invokes the routeRes() method on its callback interface, App CO, indicating that the call has been answered.

9. The routeRes() message is passed to App Logic.

10. App Logic decides that it isn’t interested in controlling the call and sends a release() message to CO. The release message allows the call to continue but releases the CO so that there is no further communication between CO and App Logic.

In this example, the NGN number translation service utilises the call control capabilities provided by the GCC SCF for service provision. The NGN number translation service encapsulates service-dependent logic i.e. the method for translating a particular number (message 6) and the ability to decide which operations to invoke on the Parlay gateway along with the parameters to be passed within each operation (messages 2, 7 and 10). The GCC SCF encompasses service-independent logic, namely call control. The mechanism used to route the dialled number (message 8) or to inform the service that it needs to translate a number (message 3) is part of the capability provided by the GCC SCF.

Service logic decomposition in the Parlay API ensures rapid third party service provision. The service developer need only develop the service-dependent logic and re-use the service-independent logic in the Parlay gateway i.e. the service developer re-uses the capabilities provided by the SCF’s or Framework Interfaces in service provision.

IV. BUILDING RE-USABLE TINA SERVICE COMPONENTS

In this section a method for decomposing TINA service components into re-usable service-dependent and service-independent sub-components is discussed. This decomposition is similar to the Parlay API and results in re-usable sub-components for rapid service creation.

A. TINA service component description

TINA’s service architecture is composed of four important service components, namely the Service Session Manager (SSM), User Service Session Manager (USM), Service Factory (SF) and Service Session User Application (ssUAP) [6]. Service development on the South African TINA platform (SATINA) shows that TINA bundles service-dependent and service-independent logic into these components [7] [8]. The next sub-section describes a way of unbundling this logic resulting in re-usable service-independent sub-components.

B. Re-usable generic service components

The service-independent logic in any service is session or call control, service management and connection control. TINA is a session based service architecture. Thus call control is not applicable in this discussion. TINA’s service architecture uses the concept of stream binding to reduce complexity by abstracting connection control in the service layer. Thus, connection control has no meaning in this discussion. For the remainder of this discussion we concentrate only on the service-independent session control logic or generic session control logic as we call it. The SSM provides Global Session Control (GSC) functionality i.e. supports session control capabilities that are global to the service session [9]. Examples of this functionality include support for [10]:

- Starting, stopping, suspending and resuming sessions;
- Addition, removal or modification to parties in an existing session;
- Creation and deletion of stream bindings; and
- Addition, removal or modification of stream flows within a stream binding;

The USM provides User Session Control (USC) functionality i.e. supports session control capabilities applicable or local to a particular party in a service session. The USC encapsulates similar functionality to that of the SSM, the only difference being that the USM performs session control operations only for the party to which it belongs and not for other parties in the session.
The service-independent management functionality includes billing mechanisms, fault management and load management. The SSM and USM contain interfaces encapsulating this functionality but are outside the scope of this discussion. The level of generic session control available for service creation and provision is supported by different Feature Sets (FS), similar to the capability levels we envisage in composing NGN services. Service Group IDs (SGID) are used to determine the FSs that make up the generic session control logic for a group of services. Figure 4 shows a method for creating re-usable generic SSM and USM sub-components using service logic decomposition and FSs.

**Figure 4. Service Logic Decomposition in TINA**

We identify the decomposed service components as \( X_{\text{generic}} \), \( X_{\text{group}} \) and \( X_{\text{Service}} \). \( X_{\text{generic}} \) is an abstract service component encapsulating the generic session control logic supporting all TINA FSs. \( X_{\text{group}} \) is a concrete service component encapsulating the generic session logic for a particular group of FSs (required FSs specified by SGID) and \( X_{\text{service}} \) is a concrete service component encapsulating service-dependent logic. The logic for using the decomposed service components is as follows:

1. A customer’s User Agent (UA) requests the start of a service session and passes \( S_{\text{generic}} \) two parameters, the Service ID (SID) and SGID. SGID is used to determine which of the generic session control interfaces to use for session control. For example, for the SWB service the generic session control interface interfaces created as part of \( S_{\text{SMGroupA}}/S_{\text{USMGroupA}} \) might include TINA’s MultipartyFS, VotingFS, SflowFS and ParticipantSBFS. For the VoD service, on the other hand, the generic session control interfaces created as part of \( S_{\text{SSMGroupA}}/S_{\text{USMGroupA}} \) might include TINA’s Basic FS, SflowFS and ParticipantSBFS.
2. SID and SGID are used to search a SF Repository for the correct service factories to use for that particular service.
3. The interface references to both factories is passed back to \( S_{\text{generic}} \).
4. \( S_{\text{generic}} \) uses the CreateSS interface, i.e. CreateSS::createSession(SID) on \( S_{\text{Service}} \).
5. \( S_{\text{generic}} \) then uses CreateSS interface, i.e. CreateSS::createSession(SSID) on \( S_{\text{Group}} \).
6. \( S_{\text{Service}} \) and \( S_{\text{Service}} \) are instantiated by \( S_{\text{Service}} \) containing the service-dependent logic.

7. \( S_{\text{USMGroupA}} \) and \( S_{\text{SSMGroupA}} \) are instantiated by \( S_{\text{Group}} \) containing only those generic session control interfaces specified by SGID.

\( S_{\text{Group}} \) creates service management interfaces for accounting, fault management, load management etc in a similar fashion. The next sub-section discusses the interaction of these generic sub-components with the service specific or service-dependent SSM and USM sub-components.

**C. Interaction between service components**

Figure 5 shows the generic SSM and USM in the network operator’s domain along with the service specific SSM and USM in the service provider’s domain. The generic SSM and USM are the stable and re-usable components developed and provided by the network operator. The service specific SSM and USM components are developed by the service provider and are used in service provision.

**Figure 5. Service Component Interactions**

Interactions at interfaces 1 and 2 are the permitted interactions between SSM and USM in TINA’s service architecture. Separation of the decomposed SSM and USM in figure 5 imitates the Parlay API approach. \( S_{\text{SMspecific}} \) and \( S_{\text{USMspecific}} \) map to Framework Interfaces and SCFs in Parlay. \( S_{\text{SMgeneric}} \) and \( S_{\text{USMgeneric}} \) utilise callback interfaces on \( S_{\text{SMspecific}} \) and \( S_{\text{USMspecific}} \) respectively for third party service provision. Interactions at interfaces 3 and 4 map to the unspecified Third Party Reference Point (3Pty) in TINA. We propose that the Parlay API is a logical substitute for this Reference Point. In the next sub-section we present an example of the interactions at interfaces 3 and 4 for a SWB service.

**D. Interaction across 3Pty**

Figure 6 shows a message sequence chart for a Shared Whiteboarding (SWB) service, illustrating the transfer of messages when a particular user suspends participation in the service. The example illustrates the use of service logic decomposition and callback interfaces in the SWB service. The re-usable service-independent logic exists in the network operator’s domain and is identified as \( S_{\text{SMGroupA}} \) and \( S_{\text{USMGroupA}} \). The service-dependent logic exists in the SWB service provider’s domain and is identified as \( S_{\text{SSMGroupA}} \) and \( S_{\text{USMGroupA}} \).

The assumption in this example is that the user is currently participating in a SWB service. The generic and specific service components exist (SSM and USM). The
user then decides to go into silent mode, that is it wants to temporarily suspend participation in the SWB service.

![Figure 6. SWB Service Component Interactions](image)

The following messages are used to suspend a user’s participation in the SWB service:

1. The user’s ssUAP (not shown in figure 6) informs USM_SWB that it would like to go into silent mode. USM_SWB invokes the goSilent() method on the SWBControl interface of SSM_SWB. The SWBControl interface is just one of the many service-dependent interfaces developed by the service provider and encapsulates operations that are specific to the SWB service.

2. SSM_SWB then invokes the suspendPartyReq() method on the ProviderMultiPartyReq interface of SSM_group. The in parameter of this message contains the PartyID, i.e. an ID for the party requesting suspension from the SWB service. This PartyID is passed in all messages used in the network operator’s domain.

3. SSM_SWB then invokes the suspendPartyInd() method on the ProviderMultiPartyInd interface of USM_group. This method informs the USM of the requesting party that its participation in the SWB service session is about to be suspended (not shown in figure 6).

4. SSM_group then invokes the suspendParticipation() method on the SSMManage interface of SF_group. The return parameter of this message will contain an interface reference to the Resume interface. If the party requesting suspension from the service decides that it would like to continue with the SWB service the Resume interface on the SSM or USM will be used.

5. SSM_group then invokes the suspendPartyExe() method on the PartyMultiPartyExe interface of USM_group. This method starts the process of suspending the stream flow belonging to the requesting party from the SWB service’s stream binding.

6. SSM_group then invokes the deactivatePartySBindInd() method on the PartySBindInd interface of USM_group. This method informs other parties in the SWB service that the requesting party is going to suspend participation in the service. After this message has been invoked SSM_group requests the relevant modification to the communications session (represented by the Communications Session Manager (CSM)) i.e. the CSM deactivates the requesting parties stream flow from the stream binding essentially suspending the connection between the requesting party and other parties in the SWB service.

7. After the CSM has made the necessary modifications it informs USM_group of these modifications (not shown in figure 6). USM_group then invokes the leavePartySBExe() method on the PartyPaSBExe interface of SSM_group informing SSM_group that the party has been suspended from the SWB service.

8. SSM_group then invokes the goSilentExe() method on the SWBControl interface of SSM_SWB informing SSM_SWB that the party has been suspended from the SWB service.

9. The goSilentExe() method is then passed on to the SWBControl interface of USM_SWB essentially informing USM_SWB that the party has been suspended from the SWB service.

In this example, the SWB service utilises the generic session control capabilities provided by SSM_group and USM_group. The generic session control capabilities are represented by TINA interfaces. The SWBControl interface in the SWB service provider’s domain is an example of service-dependent logic, i.e. the goSilent() and goSilentExe() methods of messages 1, 2, 8 and 9 are methods specific to the SWB service. Additionally, the service-dependent logic has the ability to decide which operations to invoke on the service-independent logic in the network operator’s domain along with the parameters to be passed within each operation (messages 2 and 8). The callback interface mechanism discussed in the NGN number translation service example is used to ensure communication between SSM_SWB and SSM_group.

The SWB service example indicates the separation of service-dependent logic from the service-independent session control logic. We propose that the service-independent logic in the network operator’s domain is generic, stable and re-usable by other service providers. Let’s assume that a user in a VoD service would like to pause a movie currently being watched. The mechanism for suspending the users participation in the VoD service (essentially pausing the movie) would be the same as the mechanism shown in the network operator’s domain in figure 6. The service-dependent logic associated with pausing a movie in the VoD service is a pauseMovie() method on a VODControl interface. The generic, stable and re-usable service-independent logic enables the rapid creation of third party NGN services.

V. Conclusion

This paper responds to the need for rapid yet effective service creation for external service providers. We have defined and mapped generic service functionality to certain TINA service components, most notably the SF, SSM and USM. We have also shown how the Parlay API enables rapid yet simplistic service creation through service logic decomposition. This approach has been adopted to show that the generic session control logic in the SSM and USM is re-usable in the rapid creation and deployment of third party services. The decomposition provides the benefits of the Parlay API approach and results in re-usable software below the service component level, thus enabling rapid TINA or NGN service creation. Further work to be done includes applying service logic decomposition to the ssUAP in the customer’s domain and identifying re-usable service
management logic. Ultimately, the aim is to identify re-
usuable software within service components and to develop a
system that supports the rapid creation and provision of
third party services in the NGN.

VI. REFERENCES

Multimedia Service Creation and Deployment
Platform”, Bell Labs Technical Journal, vol. 4 no. 2,
June 1999, pp. 57-70.

specification”, British Telecomms Technical Journal,
vol. 18, no. 2, April 2000, pp 57-58.

Development and Delivery of Converged Services using
API’s”, Bell Labs Technical Journal, vol. 5 no. 3,

Overview, Version 1.1.1, December 2001, ETSI
Standard ES 201 915-1.

[5] The Parlay Group, Open Service Access: API; Part4:
Call Control SCF, Version 1.1.1, December 2001, ETSI
Standard ES 201 915-4.

[6] TINA Consortium, Service Architecture, Version 5.0,
June 1997.

Hanrahan, “Video on Demand Provision using TINA”,
Proceedings of SATNAC 2000, SATCAM CD-Rom,
ISBN 0-620-26494, Somerset West, South Africa,
September 2000.

for controlling Streamed Voice Services”, Proceedings
of SATNAC 2000, SATCAM CD-Rom, ISBN 0-620-
26494, Somerset West, South Africa, September 2000.

5.0, June 1997.

[10] TINA Consortium, Service Component Specification,
Computational Model and Dynamics, Version 1.0b
Final, January 1998.

BIOGRAPHIES

PARESH NANA {p.nana@ee.wits.ac.za} received his B.Sc.
degree in Electrical Engineering from the University of the
Witwatersrand, Johannesburg, in December 2000. He is currently a
full-time M.Sc. student at the University of the Witwatersrand
where he conducts research on the Next Generation Network
(NGN). This research focuses on the development of re-usuable
service components enabling rapid third party service provision in
the NGN. Concepts and principles from both the Parlay and TINA
architectures are utilised in the development of re-usuable service
components.

SETUMO MOHAPI {s.mohapi@ee.wits.ac.za} is a Research
Officer and PhD Student with the Centre for Telecommunications
Access and Services (CeTAS). His current research interests are in
the areas of load management in distributed systems. Other fields
of interests include distributed computing and mathematical
analysis.

HU HANRAHAN {h.hanrahan@ee.wits.ac.za} is Professor of
Communications Engineering at Wits University. He leads the
Centre for Telecommunications Access and Services (CeTAS), a
research and advanced teaching centre devoted to improving
knowledge and practice in evolving telecoms access networks and
telecoms services.