Chapter 13

Development of Parlay-based Services Using UML and SDL

Markos Koltsidas, Ognjen Prnjat, Lionel Sacks

Departement of Electronic and Electrical Engineering,

University College London, Torrington Place, London WC1E 7JE, U.K.

Phone: +44 20 7679 3198 Fax: +44 20 7387 4350

Email: {oprnjat | lsacks}@ee.ucl.ac.uk

Key words: Service development approach, Parlay API, SDL, UML.

Abstract: Business and residential demands for sophisticated multimedia services are constantly on the increase, while the rapid introduction of new services and the user-tailored customization are emerging as the key requirements. A number of initiatives in the area of programmable network interfaces are paving the way for the future telecoms environment where new, customized services will be developed more quickly and efficiently by the third party providers and end-users, and deployed over the operators' network and support systems. Parlay API is one of the most notable examples of an open network programming interface, its development being carried out by the key players in the industry. Parlay specifications define a number of interfaces available to service developers, and some basic guidelines on the way of accessing these interfaces: however, no clear development approach is defined to guide the developers. Here we present and assess an approach for specification and design of the Parlay-based 3rd party applications, based on UML and SDL. We illustrate the approach with the case study of the “Conference between three parties” application.

1. INTRODUCTION

Nowadays the network operator controls all the telecoms network capabilities and is wholly responsible for the design, development and operation of all services. The existing environment is effectively made up of three layers: the bottom layer representing different network technologies,

the middle layer consisting of service components such as routing, configuration *etc.*, and the top layer featuring various applications. This architecture is sufficient for provision of services that satisfy common customer needs but the customized end-user services are not quickly and easily deployable. Overall time to market for new services is high since only the operator's service development team is involved in application development. High service deployment time consequently impacts the customer base and the operator's revenue: both residential and business customers are increasingly asking for new, sophisticated services that can be tailored to their demands. On the other hand, deregulation activities, such as the Open Network Provisioning in Europe, are prompting the established operators to allow the other licensed operators and service providers the access to their networks and support systems. Thus, operators are looking into the possibility of opening up their network and control systems at the considerably lower level of access than was available in the past. This would allow the 3rd party application developers to specify, design and deploy new services over the operator's infrastructure, using the network application programming interfaces (open network APIs). One of the main initiatives in this field is lead by the Parlay group, which is developing the Parlay API.

2. PARLAY API

Parlay API is an open technology-independent interface to generic network service components in the operator's domain: it is located between the application and the service component layers (*Figure 1*). Its goal is to allow applications to be developed and deployed by end users / value-added service providers outside the network domain. Parlay API has a revolutionary potential within the telecoms industry: software developers will have the ability to design and offer sophisticated services, by gaining access to operators' generic service components and networks.

Parlay API is object-oriented and consists of two categories of interfaces: service and framework interfaces. Service interfaces allow the application to access a wide range of network capabilities. Parlay phase 1 specification provides an interface to call control and messaging, while phase 2 gives access to wireless and IP services. Framework interfaces provide additional management support for service interfaces, such as discovery, notification, authentication, *etc.*

Figure 1. Future telecommunications network and service environment: the Parlay API

Parlay API Specification [Parlay99] consists of a set of documents. **Core Specification** defines the interfaces, methods, parameters and a few state models of the API. **Data Definitions** document specifies the data definitions. **Sequence Diagrams** document specifies the basic services, through the use of sequence diagrams, that can be derived from the API. **Class Diagrams** document describes the class diagrams used by the API specification and recommends how these should be presented. **MIDL Specification** document describes Microsoft (SOM) Interface Definition Language for the Parlay API. **OMG-IDL Specification** document describes OMG Interface Definition Language for the API. Core Parlay API interface specifications are technology-independent (specified in Unified Modeling Language [UML]) and thus potentially support a wide range of distribution middleware such as CORBA, DCOM and JAVA/RMI. Combining Core Specifications with the MIDL specification will support DCOM, while combining the Core Specifications and the OMG-IDL specification will support CORBA.

Introducing Parlay API brings many benefits to all the players in the market. For the operators, the API can become the key factor for the progress of the market through the increasing demand for new services, which can be created and operated in short time by collaboration with service providers. Service providers have the opportunity to create new services or make improvements to existing ones in relation to the API capabilities. As a result, the service reaches the market faster. Application developers also benefit from the Parlay API: applications that were either previously not technically feasible (network operator could not provide the necessary functionality) and/or not financially acceptable (high cost of development) can now be developed. End users have the ability to integrate new application solutions with other software. Competition between

developers results in the reduction of the price that is offered from an open market for customer applications [Parlay99].

For these benefits to be fully exploited, a clear and concise development approach for third party applications must be agreed between the developers and the network operators. Currently, the Parlay API specifications consist solely of the simple UML interface class definitions and a few basic examples of the interface use (via UML sequence diagrams). Behaviour of the interfaces is poorly specified through the use of a few simple UML state diagrams. The exchange of specification and design documents and a certain level of collaborative development between the application developers and network operators will be inevitable in the future. Moreover, operators will require a certain level of assurance that the applications developed outside their own domain will not in any way harm the correct and proper functioning of their service components and the networks. The operators will need a proof that the integrity of their domain will not be compromised. Integrity is defined as the ability of the system/network to retain its attributes in terms of performance and functionality [Ward95].

Thus, a development approach that will ensure both clear and concise mutual understanding, between the operators and developers, of the service application functionality, as well as allow a certain level of application integrity analysis, and verification and validation, is needed.

3. SERVICE DEVELOPMENT APPROACH

First step in the application development requires thorough understanding of the Parlay specification documents discussed above. Having consulted these, and prepared the textual description of the target application, the developer can use the following development phases:

- Application requirements analysis and high-level design using **Unified Modeling Language (UML)** notation [UML].
- Detailed application design using the **Specification and Description Language (SDL)** [Z.100].
- Application **integrity analysis** based on specification/design documents; **simulation and validation** using widely available SDL tools.

Application requirements analysis and high-level design using UML. UML is a third-generation object-oriented language, rapidly becoming a software industry standard. It is more expressive yet cleaner and more uniform than OMT [Rumb91] or other established visual modeling languages. UML is used to specify, construct and document structure and behaviour of a software system. It consists of a number of basic diagrams

and core concepts, so the users can easily learn and use it, but also extend it. Parlay API specification is based on UML: it uses class diagrams for interface descriptions, sequence diagrams for description of simple example applications, and state diagrams for depicting some basic behaviour. Here we advocate a more thorough use of UML, where the development would start by specifying the high-level application functionality through the use-case diagrams. Then, the classes needed to support this functionality would be chosen from the Parlay Core Specification document, and their relationships would be depicted through class diagrams, while class internal behaviour would be depicted through state diagrams. Interaction between the objects implementing the interfaces is depicted through collaboration and sequence diagrams, showing how objects collaborate to perform a use-case.

Detailed application design using SDL.

SDL is a formal description technique based on the concept of extended Finite State Machines. This sophisticated modeling and design technique allows complex behaviour to be expressed through asynchronous communication between a number of objects containing processes. UML, although clearer, cannot fully describe this complex behaviour. SDL is the most widely accepted formal language in the telecoms industry [Loge99], and is supported by a number of software tools. Detailed SDL design of the Parlay-based application is easily constructed from UML analysis and high-level design documents: Parlay interface objects map to the SDL objects/processes; behaviour of the processes is derived from UML state diagrams, and inter-process interaction from sequence and collaboration UML diagrams. This yields a detailed design of the Parlay-based application and the supporting objects, a design capable of depicting the complex behaviour by exploiting the powerful features of SDL. SDL design is easily transferable into executable code through the use of widely available tools.

Application simulation, validation and integrity analysis.

SDL is supported by a number of tools, which provide the means of simulating the design so as to produce Message Sequence Charts (MSCs). MSCs visually depict the behaviour of the detailed design, as a response to the external stimuli. MSCs produced in this manner can be compared to the analysis UML sequence diagrams so as to verify the conformance of the detailed design to the requirements specification. Moreover, the availability of the detailed behavioural design in SDL allows both the network operators and service providers to conduct the application and supporting Parlay objects' integrity analysis [Prnj99][Prnj00], in terms of a number of factors. Using SDL validations tools, the correct behaviour of the SDL design can be conducted by exploring the state space of the application. Reachability analysis can be performed, and livelocks and deadlocks can be detected. Moreover, conducting the service design in SDL can greatly help in feature

interaction detection prior to service launch. Network operators having access to a number of SDL designs could compare and simulate them so as detect feature interactions [Came93] that might arise due to the unexpected interactions between a number of correctly designed applications.

4. CASE STUDY

This section illustrates our development approach using the “Conference between three parties” example, which provides an audio conference between three users. The application allows user A, who is in full control, to establish through the Parlay API a conference call with user B, and then, at any point of the call, user A can add user C to the existing conference.

4.1 UML requirements analysis and high-level design

This phase involves analyzing the Parlay Core Specification and elaborating on it through a number of UML diagrams. Application high-level requirements and functionality are first captured via a **use-case diagram**. The diagram shows the structure of the system as a graph of actors, a set of use-cases enclosed by the system boundary, communication associations between the actors and the use-cases, and generalizations among use-cases.

Figure 2. "Conference between three parties" use-case diagram

Figure 2 depicts the use-case diagram for the example application. First, user A accesses the Parlay API (the framework interface). Then, A is using the application in order to request connection with user B and then with user C. Use-case “Request Connection” extends in “Connection Establish B” and “Connection Establish C” use-cases. An "extends" relationship from use-case “Request Connection” to these use-cases indicates that an instance of

these use-cases may include the behaviour specified by the parent use-case. When A has established a connection with B, he/she requests and establishes connection with C, thus creating a three-party conference call.

Figure 3. "Conference between three parties" class diagram

Having captured the application requirements, we must consult the Parlay API Core Specification document for the choice of interface classes needed to support the use-cases. A number of classes were chosen to support our application. *IparlayAppCall* handles call request responses and state reports. *IparlayCallControlManager* creates call objects. *IparlayCall* provides a structure to allow complex call behaviour to be used. *IparlayUICall* provides functions to send information to, or gather information from, the user to which a call leg is connected. *IparlayAppUICall* handles call user interaction request responses and reports. *IparlayCallLeg* tracks its own state and allows charging summaries to be accessed. *IparlayAppCallLeg* handles responses and errors associated with requests on the corresponding callLeg. These interface classes, the class operations, and the relationships between the classes, are shown on the **class diagram** on *Figure 3*. There are three instances of *IparlayCallLeg* and *IparlayAppCallLeg*, corresponding to the three users involved in the conference call. The arrows in the diagrammatic

representation indicate the way across which the communication flows: a line without arrows indicates that the communication takes place both ways.

The next step is to depict how the instances of these classes interact in order to fulfil a use-case. This can be done using two types of UML interaction diagrams: collaboration and sequence diagrams. **Collaboration diagram** shows the relationship among object instances which are related in the context of the use-case, and a set of messages exchanged between these objects. Arrow next to a line indicates a message flowing in the given direction. Messages are numbered in chronological order. *Figure 4* shows the object instances that compose our example model and messages that are exchanged when user A is establishing connection with users B and C. Not all messages are shown, for clarity. **Sequence diagram** captures the same information as the collaboration diagram, but it also includes time, by showing the objects participating in the interaction by their "lifelines" and the messages that they exchange arranged in a time sequence. Sequence diagram has two dimensions: the vertical represents time, and the horizontal different objects. In modelling our example scenario sequence diagrams, we assumed that the messages are exchanged between objects at the correct time and no that errors occur. If an error occurs (message corrupt, message never arrives at destination) then the message is re-sent.

Figure 4. "Conference between three parties" collaboration diagram

Now that the developer has captured requirements in terms of use-cases, depicted static structure of the relationships between classes via class diagrams, and specified the interaction between object instances that fulfil certain functionality, the last step in the high level design is to specify the

internal behaviour of the classes by developing a state diagram for every class chosen for the target application. **State diagram** shows the sequence of states that an object traverses during its lifetime in response to received stimuli. It thus represents a state machine: a graph of states and transitions that describes the response of an object to the outside input. A state is a condition in the life of an object during which it satisfies some condition, performs some action, or waits for some event [Fowl97]. *Figure 5* illustrates one of the state diagrams for our example application. This state machine for the IparlayCallLeg interface shows all the possible states of the object as it receives different messages, while the arrows represent state transitions.

Figure 5. IparlayCallLeg state diagram

4.2 Detailed application design using SDL

On the basis of the UML requirements analysis and high-level design, the detailed SDL design can be developed. SDL model consists of concurrent objects, described by processes, which define the control structure in terms of states and transitions. Objects are grouped into blocks, and communicate via signals, which are sent through channels. The UML to SDL mapping is straightforward: UML object instances map to SDL objects, each described by a process diagram derived from the UML state diagram. Signals are derived from UML collaboration and sequence diagrams.

Starting from the high-level UML design, our example application is designed in SDL. The SDL system design consists of a number of entities. The *Parlay* **system diagram** defines the high-level architecture of the system (Figure 6). The *ParlayAPIObjects* **block** is effectively the container of **objects**. **Processes** (finite state machines extended with data) then describe the objects in terms of states and transitions. **Signal lists** define groups of signals that are exchanged between the objects/processes. The objects, which make up the the SDL system, communicate through **channels**.

Figure 6. "Conference between three parties" SDL system diagram

Our design, developed using the Telelogic Tau tool SDT [Tau3.5], uses a single SDL block containing all objects/processes, and communication with the environment is achieved only through this block (Figure 6). By using only one block and one signal list on the system level the developer can easily add or delete signals and thus avoid signal duplication, that can occur when a number of blocks are used, and that can lead to an error. The objects contained in the SDL block are mapped directly from UML object instances: *IparlayAppCall*, *IparlayCallControlManager*, *IparlayCall*, *IparlayUICall*, *IparlayAppUICall*, *IparlayCallLegA*, *IparlayCallLegB*, *IparlayCallLegC*, *IparlayAppCallLegA*, *IparlayAppCallLegB*, *IparlayAppCallLegC*. Additionally, an *Error_Log* procedure was created to handle the unexpected signals: if a process receives an unexpected signal, with the help of the exception-handling *Error_Log* procedure it stays in the same state. All processes have been designed so as to react to a correct incoming signal by moving to a relevant state. The sound behaviour of the SDL design depends on the correct design of the processes and their states. Enriching the SDL model can be achieved through the consideration of new scenarios and creation of new states for the processes to enable them to handle various incoming signals. As a result, the SDL model becomes behaviorally rich and is capable of accommodating a wide range of scenarios.

4.3 Simulation, validation and integrity analysis

The final phase in the development approach is the detailed SDL design simulation, validation and integrity analysis. Simulation means executing the system under developer's control: signals can be sent to the system in order to examine its behaviour. Through simulation, developer creates Message Sequence Chart (MSC) diagrams, which depict the details of the interaction between the processes and the states that the processes go through.

Figure 7 illustrates an MSC diagram of our example application design simulation, representing the End_of_Conference. This diagram is only a part of the overall MSC diagram that was created with the help of the Telelogic Tau simulator tool. This MSC represents the exchange of signals during the end of conference between users A, B and C. In this case user A decides to first interrupt the conversation with user C and then with user B.

Figure 7. Message Sequence Chart depicting the end of conference

MSCs produced in this fashion are then compared with the initial requirements capture UML sequence diagrams. If the diagrams match for all the scenarios, the developer can have confidence that the detailed design conforms to the specification. In the case of our example application, the diagrams are nearly identical: discrepancies occur in the cases where the important signals in the SDL design were made to be acknowledged by a special message, a feature that was not included in the requirements analysis.

Moreover, the SDL tool used for our design could perform an in-depth validation of the application, with the *SDL validator tool*. This tool allows the exploration of the overall state space of the SDL system and is capable of detecting a range of run-time errors, by performing reachability analysis and livelock and deadlock detection. We did not use this feature, since it required the inclusion of timers for every signal exchange, while our application was not designed as a real-time one. However, this is a powerful feature, which should be fully exploited by application developers and network operators so as to detect and resolve any integrity risks during application development. Apart from performing livelock and deadlock detection, the use of validation techniques can ensure robustness and, if carried out in conjunction with the integrity analysis of other potentially interacting applications, can also resolve feature interaction problems in the operator domain. This does not guarantee the fully integral operation of the Parlay-based applications and the network operators' support systems, but it does partially ensure the required level of some integrity features [Prnj99][Prnj00].

5. DISCUSSION AND CONCLUSIONS

Future prospects of the use of network interfaces such as the Parlay API raise the issue of establishing a standardized approach for the third-party application development that can be understood and deployed by both service developers and network operators. This approach should support clear development guidelines and use meaningful and widely accepted notation schemes, and also give a solid basis for the analysis of the integrity features of the newly developed applications and the way these interact with the operator's control systems and the network. Here we presented a development approach for the Parlay-centered applications, based on UML and SDL. Development phases were presented, leading the developer from the basic application textual descriptions and Parlay specification documents, through the requirements capture and high-level design in UML, to the detailed and behaviourally rich design in SDL. The approach was illustrated using the "Conference between three parties" example application.

The main advantages of this approach are clear and straightforward mappings from the Parlay specifications to UML high-level design, and the UML high-level design to the detailed SDL design. Thus, a developer has a clear guideline how to achieve final service design starting from basic Parlay interface specifications. UML and SDL are widely used languages both in telecoms and IT industries: UML is a de-facto standard, while SDL is increasingly gaining wider acceptance [Loge99]. Incorporating these languages in the approach could help collaborative service development between third-party developers or between developers and network operators themselves. Moreover, UML and SDL can be seen as documentation tools aimed to help the service maintenance: lack of clear service documentation in the operator's domain is more frequent than believed [Ward95].

Another important consideration is that the service modeling techniques used should facilitate close analysis of the design to ensure, at early stages, that the system is complete in meeting its requirements and consistent in its operation. This should yield robust, well-engineered products. Our approach, by providing rich SDL behaviour models, allows the developers and network operators to analyze, validate and test the new applications with respect to their integrity features, with the help of widely available SDL support tools. Service verification and validation can be performed, and deadlock, livelock and feature interaction detection can be conducted prior to service launch. Service validation and integrity analysis are seen to be of paramount importance to the network operators, who will require strong assurance from the third party developers that the applications deployed will not harm their network and supporting systems. The benefits of the proposed approach will have to be traded off against one clear drawback: the SDL development might be time consuming and thus slow down the time to market which can be seen as of significant importance in the future open network scenarios.

REFERENCES

- [Came93] E. J. Cameron and H. Velthuisen, "Feature Interactions in Telecommunications Systems", *IEEE Communications Magazine*, August 1993, pp. 18-23.
- [Fowl97] M. Fowler, S. Kendall, "UML Distilled", Addison-Wesley, 1997.
- [Loge99] X. Logean et. al., "On Applying Formal Techniques to the Development of Hybrid Services", *IEEE Comms Magazine*, Vol. 37, No. 7, July 1999, pp. 132-138.
- [Parlay99] Parlay Group, "Parlay API Specification 1.1", Issue 1.1, 7th May 1999.
- [Prnj00] O. Prnjat, L. Sacks, "Inter-domain Integrity Management for Programmable Network Interfaces", *Management of Multimedia Networks and Services' 2000*, September 2000.
- [Prnj99] O. Prnjat, L. Sacks, "Integrity Methodology for Interoperable Environments", *IEEE Comms Magazine*, Vol. 37, No. 5, May 1999, pp. 126-139.
- [Rumb91] J. Rumbaugh et. al., "Object-Oriented Modeling and Design", Prentice-Hall, 1991.
- [Tau3.5] Telelogic Tau 3.5, SDT Tool, <http://www.telelogic.se>

- [UML] Rational Software, "UML Semantics", Version 1.1, 1st September 1997; "UML Notation Guide", Version 1.1, 1st September 1997.
- [Ward95] K. Ward: "Impact of Network Interconnection on Network Integrity", British Telecommunications Engineering, Vol. 13, January 1995, pp. 296-303.
- [Z.100] ITU Rec. Z.100, "Specification and Description Language (SDL)", 1993.
- [Z.120] ITU Rec. Z.120, "Message Sequence Charts (MSCs)", 1994.