Formal Service-Oriented Development of Fault Tolerant Communicating Systems

Linas Laibinis*, Elena Troubitsyna*, Sari Leppänen**, Johan Lilius*, Qaisar Malik*

*Åbo Akademi University, Department of Computer Science, Lemminkaisenkatu 14 A, 20520, Turku, Finland
** Nokia Research Center, Mobile Networks Laboratory, P.O. Box 407, 00045, Helsinki, Finland

1. Introduction

Majority of engineering methods for building complex systems is based on system decomposition. In the software engineering the decomposition-based development methods are often referred to as the service-oriented methods. The notion of a *service* provides a convenient mechanism for modelling and reasoning about system interactions and functionality.

In the telecommunicating systems, a service is usually understood as a coherent piece of functionality that the system delivers to its users. Since telecommunicating systems are distributed by their nature, a service is usually provided by several collaborating service components. Often communication between service components relies on an unreliable media, such as, e.g., radio-based mobile network. Hence communication failures are an intrinsic part of system behaviour. Therefore, the correct service provision is unfeasible without integrating the fault tolerance mechanisms in the system design.

In this paper we propose a formal approach to service-oriented development of fault tolerant communicating distributed systems. Our approach is based on formalization of the service-oriented methodology Lyra [4] developed in the Nokia Research Center. The design flow of Lyra is based on concepts of decomposition and preservation of the externally observable behaviour. The system behaviour is modularised and organized into hierarchical layers according to the external communication and related interfaces so that the distributed network architecture can be derived from the functional system requirements via a number of model transformations. This approach coincides with the stepwise refinement paradigm adopted in the B Method [1].

In this paper we describe our work on the formalizing Lyra in B. We propose general specification and development patterns according to which the services can be specified and decomposed into communicating service components. The patterns generalise the existing practice of the communicating system engineering. Hence our approach provides the basis for automating the process of development of communicating systems correct by construction. It is illustrated by a case study – development of a Third Generation Partnership Project (3GPP) positioning system.

2. Overview of Lyra

The Lyra design method consists of four phases corresponding to the classical design phases: Service Specification, Service Decomposition, Service Distribution and Service Implementation. These phases correspond also to the conventional phases of standardization [2]. In the Service Specification phase we define the services provided by the system (standardization Phase 1). In the Service Decomposition phase we specify the functional architecture of each the system level service (standardization Phase 2). In the Service Distribution phase logical entities of the functional architecture, i.e. service components, are distributed over a given network architecture and signalling protocols are defined for communication between the network elements (standardization Phase 3). In the Service Implementation phase we adjust the functionality to the target environment. The program code for a specific platform is generated automatically from the resulting implementation.

Next we describe the general idea of the methodology with a running example. We model part of a Third Generation Partnership Project (3GPP) positioning system [7,8]. The positioning system provides positioning services to calculate the physical location of a given user equipment (UE) in a Universal Mobile Telecommunication System (UMTS) network. We focus on Position Calculation Application Part (PCAP) – a part of the positioning system allowing communication in the Radio Access Network (RAN). PCAP manages the communication between the Radio Network Controller (RNC) and the Stand-alone Assisted Global Positioning System Serving Mobile Location Centre (SAS) network elements. The functional requirements for the RNC-SAS communication have been specified in [7,8].

The Service Specification phase defines the service in terms of its communication with the service consumer as shown in Fig. 1. The service consumer requests the positioning calculation and receives the result of the service execution via the provided service access point (the upward interface). At this development stage we abstract away from the details of the position computation and merely observe that the service execution can result either in the position calculated with the requested accuracy or in a failure.

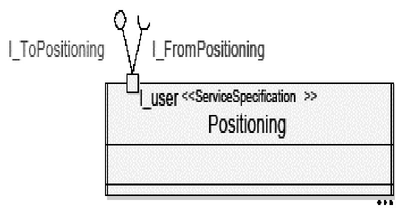

Fig 1. Service specification

At the next stage of the development process – Service Decomposition we take into account that a service is provided in co-operation with service components. The initial model presented in Fig.1 is augmented with the used service access points (the downward interfaces) via which the communication with service components is conducted. The model obtained as a result of the Service Decomposition phase is presented in Fig.2.

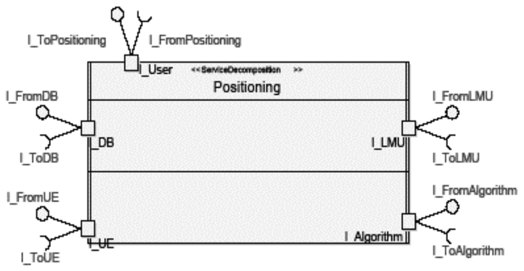

Fig 2. Service decomposition

At the Service Decomposition phase we also design the functional architecture of the service, as shown in Fig 3. Usually the functional architecture is constructed according to the following pattern: a service director orchestrates the service execution by requesting certain services from service components. For instance, to execute the position calculation service, the service director first requests an approximate location of the UE from the network database, then it requests the UE to send additional radio measurements, then it requests several local measurement units (LMU) to provide some local measurements, and finally the data collected during all these stages are sent to a location algorithm server which invokes a certain algorithm for position calculation. After executing the algorithm, the calculated position is returned to the service director. Let us observe that any of the requested components can fail (either because of communication or some other failure). When a request to a service component fails, the service director diagnoses the failure and decides on the fault tolerance measures to be undertaken.

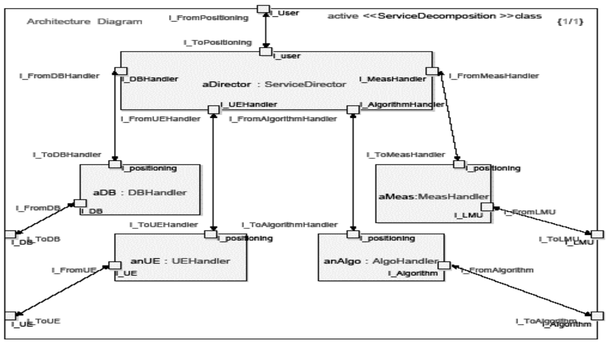

Fig 3. Functional architecture

To manage complexity of communicating systems, at the Service Decomposition phase the communication between components still remains on a virtual level – the realistic communication protocols are introduced upon completing the next (Service Distribution) stage. At the Service Distribution phase we map the functional system architecture to the platform architecture. For instance, in our example we decompose the system in such a way that communication with the network database and UE is performed by the service director allocated on RNC, while communication with LMU devices and the algorithm server is performed by the service director allocated on SAS. The service directors communicate via a certain (predefined by PCAP) protocol. The result of service distribution is shown in Fig. 4.

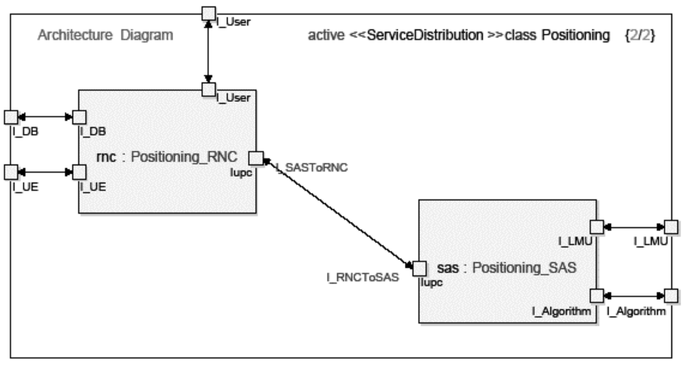

Fig 4. Platform architecture

In the final (Service Implementation) phase we adjust the model to fit a specific platform. We omit the detailed discussion of this stage. In the next section we give a brief introduction into our formal framework – the B Method, which we will use to formalize the development flow described above.

3. The B Method

The B Method [1] (further referred to as B) is an approach for the industrial development of highly dependable software. The method has been successfully used in the development of several complex real-life applications [5]. The tool support available for B provides us with the assistance for the entire development process. For instance, Atelier B [6], one of the tools supporting the B Method, has facilities for automatic verification and code generation as well as documentation, project management and prototyping. The high degree of automation in verifying correctness improves scalability of B, speeds up development and, also, requires less mathematical training from the users.

The development methodology adopted by B is based on stepwise refinement [1]. While developing a system by refinement, we start from an abstract formal specification and transform it into an implementable program by a number of correctness preserving steps, called *refinements*. A formal specification is a mathematical model of the required behaviour of a (part of) system. In B a specification is represented by a set of modules, called Abstract Machines. An abstract machine encapsulates state and operations of the specification and as a concept is similar to a module or a package.

Each machine is uniquely identified by its name. The state variables of the machine are declared in the VARIABLES clause and initialised in the INITIALISATION clause. The variables in B are strongly typed by constraining predicates of the INVARIANT clause. All types in B are represented by non-empty sets.

The operations of the machine are defined in the OPERATIONS clause. In this paper we use Event B extension of the B Method. The operations in Event B are described as guarded statements of the form SELECT cond THEN body END. Here cond is a state predicate, and body is a B statement. If cond is satisfied, the behaviour of the guarded operations corresponds to the execution of their bodies. However, if cond is false, then the execution of the corresponding operation is suspended, i.e., the operation is in waiting mode until cond becomes true.

B statements that we are using to describe a state change in operations have the following syntax:

```
S  ==  x := e | IF cond THEN S1 ELSE S2 END | S1 ; S2 |  x :: T  |
         S1 || S2 | ANY z WHERE cond THEN S END  |  ...
```

The first three constructs – assignment, conditional statement and sequential composition (used only in refinements) have the standard meaning. The remaining constructs allow us to model nondeterministic or parallel behaviour in a specification. Usually they are not implementable so they have to be refined (replaced) with executable constructs at some point of program development. The detailed description of the B statements can be found elsewhere [1].

The B method provides us with mechanisms for structuring the system architecture by modularisation. A module is described as a machine. The modules can be composed by means of several mechanisms providing different forms of encapsulation. For instance, if the machine C INCLUDES the machine D then all variables and operations of D are visible in C. However, to guarantee internal consistency (and hence independent verification and reuse) of D, the machine C can change the variables of D only via the operations of D. In addition, the invariant properties of D are included into the invariant of C.

To illustrate basic principles of specifying and refining in B, next we present our approach to formal service-oriented development.

4. Formal Service-Oriented Development

We start to formalize the Lyra development by creating a specification pattern modelling a communicating component. This pattern is used throughout the entire development process. The pattern is called Abstract Communicating Component (ACC). ACC consists of a “kernel”, i.e., the provided functionality, called Abstract

Calculating Machine (ACAM), and a "communication wrapper", i.e., the communication channels via which data are supplied to and consumed from the component, called Abstract Communicating Machine (ACM).

The specification of an abstract communicating component (ACC) consists of operations specifying ACM and ACAM. The variables inp_chan and out_chan model the input and output channels. The environment places requests for the service by assigning to inp_chan and receives the results of the service via out_chan. Data transferred to and from ACC are modelled abstractly. We reserve the abstract constant NIL to model the absence of data, i.e., the empty channel. The variables input and output are one-place data buffers internal to the ACC. The variable input stores the data read from inp_chan. It is used as a temporal data storage in calculating the required service. The variable output stores the final result of calculations which is consequently put into the output channel out_chan for the server consumer.

The operations env_write and env_read model the behaviour of the service consumer: placing the request to execute a service and reading the results of its execution. The operation read models reading the service request by ACC from the input channel. The symmetric operation write writes the results of service provision into the output channel. These operations specify the "communication wrapper" (i.e., ACM) part of ACC.

In the initial specification, ACAM is modelled abstractly by the operation calculate, which non-deterministically assigns the variable output either the result of a successful service provision or a failure. The machine ACC (presented below) specifies the described behaviour instantiated for the position calculation case study.

```
MACHINE  ACC

VARIABLES
  inp_chan, input, out_chan, output

INVARIANT
  inp_chan : INPUT_DATA  &
  input : INPUT_DATA  &
  out_chan : POS_DATA  &
  output : POS_DATA

INITIALISATION
  inp_chan, input := INP_NIL, INP_NIL ||
  out_chan, output := POS_NIL, POS_NIL

OPERATIONS

env_write =
  SELECT inp_chan = INPUT_NIL
  THEN
    inp_chan :: INPUT_DATA - {INPUT_NIL}
  END;

read =
  SELECT not(inp_chan = INP_NIL) &
             (input = INP_NIL)
  THEN
    input,inp_chan := inp_chan,INP_NIL
  END;

calculate =
  SELECT not(input = INP_NIL) &
             (output = POS_NIL)
  THEN
    CHOICE
      output ::
          POS_DATA - {POS_NIL,POS_FAIL}
    OR
      output := POS_FAIL
    END ||
    input := INP_NIL
  END;

write =
  SELECT not(output = POS_NIL) &
(out_chan = POS_NIL)
  THEN
    out_chan,output := output,POS_NIL
  END;

env_read =
  SELECT not(out_chan = POS_NIL)
  THEN
    out_chan := POS_NIL
  END

END
```

The next phases in the Lyra development are the functional and the platform-based decomposition. In our approach they correspond to two consequent refinement steps – first refining the algorithmic part of ACC, i.e. ACAM, and then introducing the peer entities distributed over the given platform. The result of the first refinement is graphically represented in Fig.5.

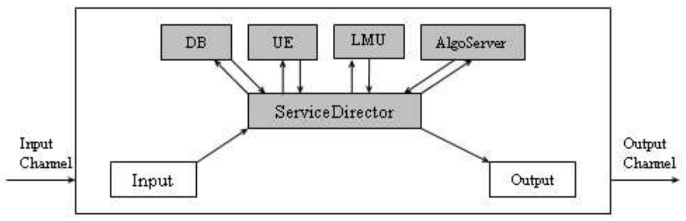

Fig.5. Functional decomposition of ACAM

ACAM is refined by introducing the service director modelled by the operation director which orchestrates the execution of the whole positioning service, and the operations db, ue, lmu, and alg modelling execution of the corresponding service components.

Basically we decompose ACAM to model stages of the positioning service. Additionally, we introduce the variables that model results obtained at these intermediate stages from the corresponding service components. At this refinement step, these variables are non-deterministically updated in the operation director. We model not only successful execution of the intermediate stages by the service components but also possible failures. Moreover, we abstractly model error recovery – upon detecting an error, the service director can retry (up to the predefined number of attempts) to execute a certain stage of the service. However, if error recovery fails, the service director terminates the service execution and returns the error as the final result. The refined specification of the ACC – ACC1 instantiated for the positioning service is presented in Fig.6.

The second refinement that we perform models the mapping of the functional decomposition into the given target platform. We introduce communication with the service components into the specification of the service director. The service components are specified according to the proposed pattern ACC. The service director plays now a role of the service consumer for the service components. Namely, it sends the requests to execute the services required for the corresponding stages and receives the obtained results. Let us observe that at this refinement step we replace non-deterministic updates to the variables storing the results of the intermediate stages by assigning them the values obtained from the communication channels. The graphical representation of this stage for the positioning system is given in Fig.7.

```
REFINEMENT  ACC1

REFINES  ACC

VARIABLES
  curr_service, handling_flag …

INVARIANT
  curr_service : SERVICE  &
  handling_flag : BOOL & …

INITIALISATION
  curr_service, handling_flag := SD,FALSE
  || …

OPERATIONS

env_write =  …

read = …

db =
  SELECT curr_service = DB
  THEN
    handling_flag := TRUE
  END;

ue = …

lmu = …

alg = …

director =
  SELECT handling_flag = TRUE
  THEN
    IF curr_service = SD
    THEN
      curr_service := DB
    ELSIF curr_service = DB
    THEN
      dbdata :: DB_DATA-{DB_NIL};
       IF DB_Eval(dbdata) = OK
       THEN
         curr_service := UE
       ELSIF DB_Eval(dbdata) = RECOV &
                (n_db > 0)
       THEN
         n_db := n_db-1
       ELSE
         posdata,curr_service :=
                     POS_FAIL,CALC
       END
    ELSIF curr_service = UE …
    ELSIF curr_service = LMU …
    ELSIF curr_service = ALG …
    END ||
    handling_flag := FALSE
  END;

calculate =
  SELECT (curr_service=CALC) & …
  THEN
    output,input := posdata,INP_NIL ||
    curr_service := SD
  END;

write = …

env_read =  …

END
```

Fig.6. ACC1 refinement

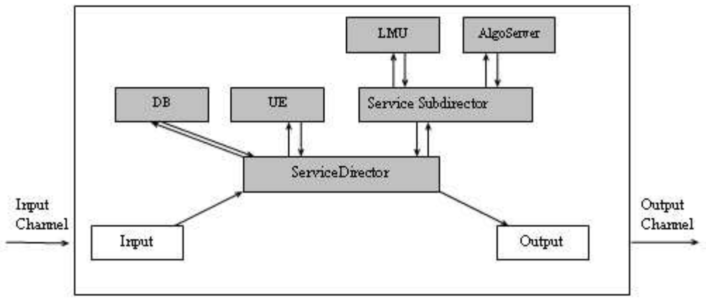

Fig.7. Service distribution

The excerpt from the refinement of ACC1 – the refinement machine ACC2 is given below.

```
director =
  SELECT handling_flag = TRUE &
        (((curr_service = DB) & not(db_out_chan = DB_NIL)) or ...)
  THEN
    IF curr_service = SD
    THEN
      curr_service := DB
    ELSIF curr_service = DB
    THEN
      dbdata <-- db_read_ochan;
      IF DB_Eval(dbdata) = OK
      THEN
        curr_service := UE
      ELSIF DB_Eval(dbdata) = RECOV & (n_db > 0)
      THEN
        n_db := n_db-1
      ELSE
        posdata,curr_service := POS_FAIL,CALC
      END
    ELSIF curr_service = UE ...
    ELSIF curr_service = ALG
    THEN
      posdata <-- sas_read_ochan; ...
    END ||
    handling_flag := FALSE
  END;
```

At the consequent refinement steps we will focus on particular service components and refine them (in the way described above) until the desired level of granularity is obtained. Once all external service components are in place, we can further decompose their specifications by separating their *ACM* and *ACAM* parts. Such decomposition will allow us to concentrate on the communicational parts of the respective components and further refine them by introducing details of required concrete communication protocols.

5. Conclusions

In this paper we proposed an approach to formal modelling of communicating distributed systems. We derived the specification and refinement patterns that can be used to automate the development of such systems. The patterns define the formal semantics of UML diagrams usually used in the development process. Hence UML modelling can be used as a syntactic sugaring of the formal development. Because of the wide acceptance of UML in industry our approach can therefore be easily integrated into existing development practice.

In this paper we demonstrated how to model faulty behaviour and fault tolerance features of communicating systems. Unreliable communication is an intrinsic part of communicating distributed systems. Hence addressing the fault tolerance issues in the development process is an important merit of the proposed approach.

The formalisation of the UML-based development of communicating distributed systems has been undertaken in the Lyra approach. Lyra is based on model checking and enables reasoning about preservation of the externally observable behaviour throughout the development process. However, the model checking techniques are prone to the state explosion problem since telecommunicating systems tend to be large and data intensive. Our approach helps to overcome this limitation.

As a future work, we will continue to develop the proposed approach to address issues of concurrency and verification of the temporal properties of communication protocols between network elements. Moreover, we are planning to develop a tool support to automate the proposed approach.

Acknowledgements
The work reported in this paper is financially supported by IST-511599 EU Project RODIN.

References

1. J.-R. Abrial. *The B-Book*. Cambridge University Press, 1996.
2. ITU-T. Rec. I.30 (1993). Method for characterization of telecommunication services supported by an ISDN and network capabilities of an ISDN.
3. L.Laibinis and E.Troubitsyna. *Fault Tolerance in a Layered Architecture: A General Specification Pattern in B*. Proceedings of 2nd International Conference on Software Engineering and Formal Methods (SEFM 2004), Beijing, China, September 2004. IEEE Press, pp.346-355.
4. S.Leppänen, M.Turunen, and I.Oliver. *Application Driven Methodology for Development of Communicating Systems*. FDL'04, Forum on Specification and Design Languages. Lille, France, September 2004.
5. MATISSE Handbook for Correct Systems Construction. 2003. http://www.esil.univ-mrs.fr/~spc/matisse/Handbook/
6. Steria, Aix-en-Provence, France. *Atelier B, User and Reference Manuals*, 2001. Available at http://www.atelierb.societe.com/index uk.html
7. 3GPP. Technical specification 25.305: Stage 2 functional specification of UE positioning in UTRAN. See http://www.3gpp.org/ftp/Specs/html-info/25305.htm
8. 3GPP. Technical specification 25.453: UTRAN Iupc interface positioning calculation application part (pcap) signalling. See http://www.3gpp.org/ftp/Specs/html-info/25453.htm