Formal Specification and Validation using a Scenario-Based Approach: The GPRS Group-Call Example

Daniel Amyot, Neil Hart, and Luigi Logrippo
School of Information Technology and Engineering
University of Ottawa
150 Louis-Pasteur, Ottawa, Ontario
K1N 6N5, Canada
+1 613 562 5800 (ext. 6704)
{damyot | nhart | luigi}@csi.uottawa.ca

Pascal Forhan
Institut National des Télécommunications
5 rue Fourier
91000 Evry
France
pforhan@worldnet.fr

ABSTRACT
Deriving a formal specification and a detailed design from informal requirements can be a tedious and error-prone task unless a methodical or rigorous approach is used. An increasing number of designers are interested in scenario-driven approaches that allow them to focus on the main functional aspects of the system to be specified. We present an approach where informal requirements are described with a use case notation called Use Case Map, and formal specifications and test cases are written in the process-algebraic language LOTOS. We present the approach by using an example: a Group Call service of the mobile data system GPRS. This work also aims to present ideas on how to integrate LOTOS and Use Case Maps in the ROOM methodology, and to discuss several strategies for scenario-based validation.

Keywords
Scenarios, Requirements Engineering, Formal Specification, Tests, Standards, GPRS, LOTOS, Use Case Maps, ROOM.

INTRODUCTION

The Standardization Challenges

Under the auspices of standardization bodies such as ITU, ISO, ANSI, ETSI, TIA, etc., standardization committees are constantly at work to produce standards for telecommunications products, for which the main logic is meant to be implemented in software. In the early stages of this processes, many features, services, and functionalities are described using informal operational descriptions, tables and visual notations such as *Message Sequence Charts* (MSCs) [12]. These descriptions evolve dynamically, and their drafting and validation quickly become difficult to manage. MSCs are also used in the ROOM methodology [18] as the primary scenario notation.

In this context, the following issues should be addressed:

- While designing systems and services at the initial stages, the discussion might focus at a level of detail that is too low with respect to the knowledge (about data, messages, components, etc.) available at the time. Descriptions that include irrelevant details tend to obscure the main idea behind a feature/service/functionality, especially when the latter needs further modifications or refinements. Several levels of detail (abstraction) are often mixed in a single description.
- A focus on message exchanges (using MSCs) is necessary for detailed design, but this can be cumbersome while defining the functionalities in the initial design steps. A simpler visual notation that abstracts from messages would help focusing on the real issues while providing for more manageable and reusable scenario descriptions.
- There are possibly ambiguities, inconsistencies or interactions inside or between service descriptions, or between levels of abstraction of a given service. These remains difficult to detect with conventional inspection methods, and often remain hidden until errors are discovered after implementation, at which point correction can be very costly.

Need for Scenarios in a Rigorous Approach

The process of going from informal functional or operational requirements to a high-level formal specification is a research subject where much work has been done. However, many challenges still remain, especially regarding newer techniques for defining requirements and formal specifications. *Formal Description Techniques* (FDT), such as LOTOS [11], were created in order to formally express functional requirements. In particular, they are well suited for the precise definition of telecommunication systems.

Over the last few years, there has been a strong interest, from both academia and industry, in the use of scenarios for system design. The introduction of *use cases* [13] in the OO world confirmed this trend. Many methodologies are now available. However, many different meanings were associated to the word “scenario”. They are related to traces (of internal/external events), message exchanges between components, interaction sequences between a system and its user, to a more or less generic collection of such traces, etc. Numerous notations are also used to describe them: grammars, automata, and message exchange diagrams similar to MSCs. The approaches available differ on many aspects, depending on the definition and the notation used. It should be noted also that this work relates mostly to software that is sequential in nature, while we concentrate on concurrent software.

Several techniques can be used to address the issues related to standardization processes and scenarios. We use LOTOS to describe the specification obtained from high-level scenarios, described as *Use Case Maps* (UCM) [7][8]. These designs are also documented with tables describing the activities. We use tools and techniques developed or used within our research group for verification, validation, scenario-based testing, and coverage measurement, in order to detect inconsistencies, ambiguities, incompleteness, and other problems as soon as possible. The goal is to produce documentation that is more easily understood and also is validated as far as possible, and to produce a validated test suite that can be reused at later stages, including implementation. We illustrate this process on a case study involving the Point-to-Multipoint Group Call service of General Packet Radio Services. Later, we present ideas on how such an approach could leverage the design and validation processes in the ROOM methodology.

BACKGROUND

General Packet Radio Services (GPRS)

GPRS [10] allows the service subscriber to send and receive data in an end-to-end packet transfer mode. This service is a set of *Global System for Mobile Communications* (GSM) [14] bearer services that provides packet transfer in interworking with external networks and within a *Public Land Mobile Network* (PLMN).

The service is divided in two main branches: *Point-To-Point* (PTP) and *Point-To-Multipoint* (PTM), based on existing and standardized network protocols.

Typical applications for PTP services are: retrieval services (Web), messaging services (mailbox), real time conversations (Telnet), and short tele-actions (credit-card validation). Typical applications for PTM services are: unidirectional distribution services (newsgroups), bidirectional dispatching services (taxi fleet), and multi-directional conferencing services, without store and forward, between multiple users.

The PTM services have several capabilities, including geographical routing to restrict distribution and scheduled delivery. In this project, we focus on the *PTM-Group Call* (PTM-G). This service allows transmissions to specific groups of users in specific geographical areas. At any point in time, the network has the knowledge of the number of users and their location.

Language Of Temporal Ordering Specification (LOTOS)

Formal methods have proven their usefulness in capturing descriptions of complex, concurrent, and communicating systems. LOTOS is an algebraic specification language and a standardized Formal Description Technique [11]. Using LOTOS, the specifier describes a system by defining the temporal relations along the interactions that constitute the system's externally observable behavior. Data abstractions can also be described by using *Abstract Data Types* (ADTs).

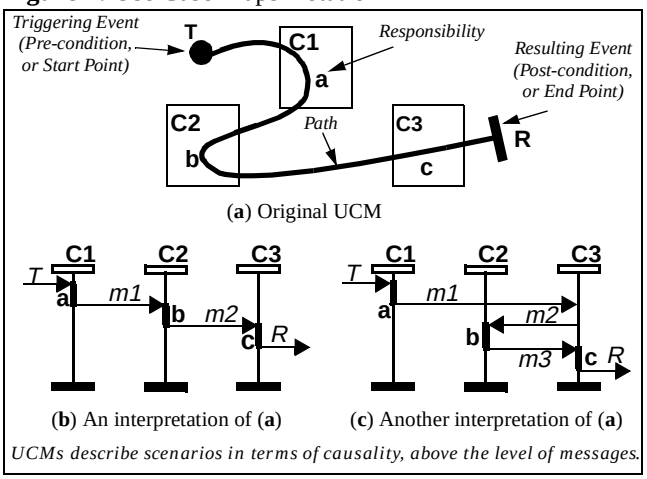
LOTOS is powerful at describing and prototyping communicating systems at many levels of abstraction through the use of *processes*, *hiding*, and *synchronization*. LOTOS is suitable for the integration of behavior and structure in a unique executable model. LOTOS models allow the use of many validation and verification techniques such as step-by-step execution (simulation), random walks, testing, expansion, model checking, and goal-oriented execution. Many tools can be utilized for the automation of these techniques, and several development cycles based on stepwise refinement are available [5].

Use Case Maps (UCMs)

UCMs are a visual notation we utilize for capturing the requirements of reactive systems. They describe scenarios in terms of *causal relationships* between *responsibilities*. They can have internal activities as well as external ones. Usually, UCMs are abstract (generic), and could include multiple traces. With UCMs, scenarios are expressed above the level of messages exchanged between components; hence they are not necessarily bound to a specific structure.

To illustrate several UCM concepts, Figure 1(a) shows a very simple UCM that contains only one *route*, linking a cause to an effect. A scenario starts with a triggering event or a pre-condition (filled circle) **T** and ends with one or more resulting events or post-conditions (bar) **R**. Intermediate responsibilities (**a**, **b**, **c**) have been activated along the way. In this picture, the activities are allocated to abstract components ($\mathbf{C_1}$, $\mathbf{C_2}$, $\mathbf{C_3}$). We call such superposition a *bound map* (and respectively an *unbound map* when there are no components). The notation also allows for alternative and concurrent paths, and for interactions between paths. For a detailed description of the notation, refer to [8]. A textual format (linear form), where UCMs can be defined using a formal grammar, is also described in [4]. Such linear form is generated automatically by our UCM graphical editor for further processing by other tools (compilers, code generators...).

Figure 1: Use Case Maps Notation



A causal relationship can be refined in many ways in terms of exchanges of messages, depending on the components structure, the available communication channels, and on the chosen protocols. Several MSCs could be considered as valid implementations of a UCM. For example, the MSC in Figure 1(b) represents a straightforward interpretation of Figure 1(a). The MSC in Figure 1(c) could result from an architecture where there is no channel directly linking $\mathbf{C_1}$

and C_2, and hence the causality between **a** and **b** would have to be refined by, for instance, messages *m1* and *m2*.

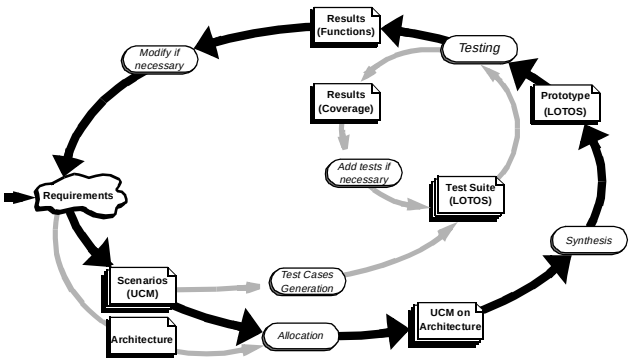
The construction of a UCM can be done in many ways. Usually, one starts by identifying the activities that are to be performed by the system. They can then be allocated to scenarios and/or to components. Components can be discovered along the way. Eventually, the two views are merged to form a UCM.

A first attempt at formalizing UCMs was done in [1]. That technique was illustrated with a small telephony example in [2]. As this method tried to derive a complete model as one single UCM, it appeared to be not sufficient for describing complex systems composed of multiple views. Our new approach addresses this issue.

RIGOROUS APPROACH BASED ON SCENARIOS

We believe that the usage of UCMs in a scenario-based approach represents a judicious choice for the description of reactive and communicating systems. They fit well in the design approach that we propose in Figure 2, where we intend to bridge the gap between informal requirements and the first system design.

Figure 2: Scenario-Based Approach Used in this Project



Requirements are usually dynamic; they change and are adapted over time. This is why we promote an iterative and incremental process (in spiral) that allows rapid prototyping and test cases generation directly from scenarios. Figure 2 introduces an approach where the main cycle is concerned with the description of the scenarios and of the architecture (which can be done independently). They are then merged in order to (manually) synthesize a LOTOS specification, our prototype. Concurrently, test cases can be generated from these scenarios and then be used to test the specification. The results we obtain from those tests allow us to see whether or not additional test cases are necessary in order to achieve the desired specification coverage. We then consider that the prototype corresponds to the requirements.

We observed several advantages to this rigorous approach:

- ***Separation of the functionalities from the underlying structure***: since scenarios are formalized at a level of abstraction higher than message exchanges, different underlying structures or architectures can be evaluated with more flexibility. The scenarios then become highly reusable entities. As mentioned below, they can be used again to test the implementation.
- ***Fast prototyping***: once the structure and the scenarios are selected and documented, a prototype can then be generated rapidly.
- ***Test cases generation***: scenarios facilitate the generation of test cases that relate directly to the operational requirements. The test suite can itself be validated using structural coverage criteria on the model.
- ***Design documentation***: the documentation is done as we go along the design cycle. Very often, designers document their design only when they have to; we believe this approach encourages designers to methodically produce useful documentation.

SPECIFICATION OF GPRS GROUP CALL (PTM-G)

Informal Requirements and Assumptions

Six operations are defined in [10] for the implementation of the PTM-G service: *Initiate Call*, to create a call; *Terminate Call*, to delete a call; *Call Status*, to get the attributes of a call; *Join Call*, to join an existing call; *Leave*, to quit a joined call; *Data Transfer*, to send messages and data.

In order to generate the Call Terminate and Leave operations invoked by the network, we defined three artificial operations that we could trigger at will. Two of them are located in the underlying services: *Attach GPRS* and *Detach GPRS*. The third one, *Change Zone*, emulates the routing operation triggered by the physical layer.

In this paper, we will consider only one operation, namely *Initiate Call*. Upon such request (*Req_Init*), two outcomes are possible: the acceptance (*Ack-C-Init*) and the rejection (*Err-Rejinit*). In case of acceptance, the indication *Ind_Init* is multicast to the receivers only if the Initiate Call Notification (*call_not*) attribute defined for the call says so. Beside the usual member identification number (*M_ID*) and the International Mobile Group Identity (*IMGI*), the different parameters provided with this request are: the Data Transfer Mode (*DTM*), the quality of service (*QoS*), the geographical area (*GeoZone*), the *join_leave* indication (to inform when someone joins or leaves the call), and the Initiate Call Notification (*call_not*).

We also included a last parameter named *send_to_all*. It is not defined in [10], but it seemed convenient to consider it as it allows the multicast of a join/leave indication to all members in the call. In the draft standard, the initiator was the only one allowed to receive this indication.

The main reasons for rejecting an Initiate Call request include: incorrect geographical zone, invalid IMGI, or insufficient privilege access.

Architecture

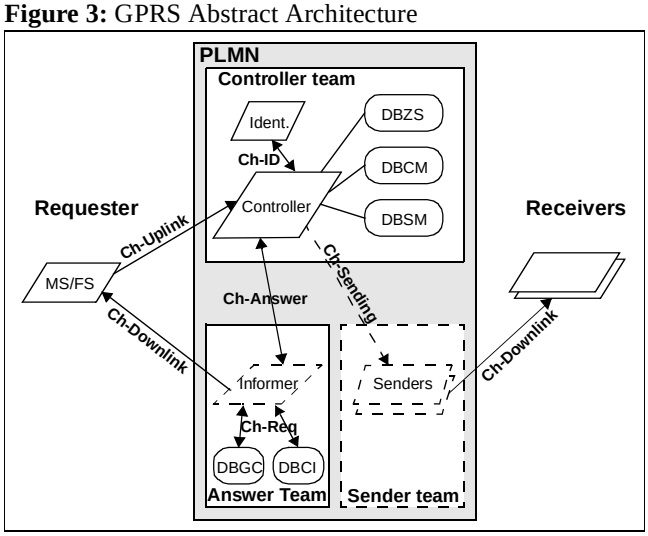
Our approach allows designers to specify the system architecture independently of scenarios, i.e., before, during, or after the specification of the scenarios. In our case, we first defined an abstract architecture, independent of GPRS. Once the scenarios are validated, they can be mapped on a more concrete architecture. A detailed architecture (e.g., a ROOM structure) could also be used from the beginning. However, this is not the goal of the current work, and this

step is postponed to a later stage of the development cycle, possibly in the detailed design.

Figure 3 presents our GPRS abstract architecture. It shows several kinds of objects: active processes as parallelograms, passive objects (e.g., databases) as rounded-corner rectangles, and containers (teams) are represented by rectangles. Arrows represent channels. The components in dotted lines are dynamically instantiated when required. Stacks of processes show that multiple concurrent instances may coexist.

We have identified three teams in the *PLMN*, each with a specific role. The *Controller Team* manages the mechanisms of control and reception of requests. The *Answer Team* sends responses back to the *Requester*. The *Sender Team* sends to the other participants, called *Receivers* in this case, the indications resulting from a request. A client can accumulate both the roles of requester and receiver as a Mobile Station, or even a Fixed Station (*MS/FS*) since GPRS can support connections with non-mobile clients.

Figure 3: GPRS Abstract Architecture



We defined several databases that contain the information required by our scenarios: *DBZS* for the localization of stations, *DBCM* for the list of members who joined a call, *DBSM* for member characteristics, *DBGC*, for the list of Call-ID of each group, and *DBCI* for the parameters of each call.

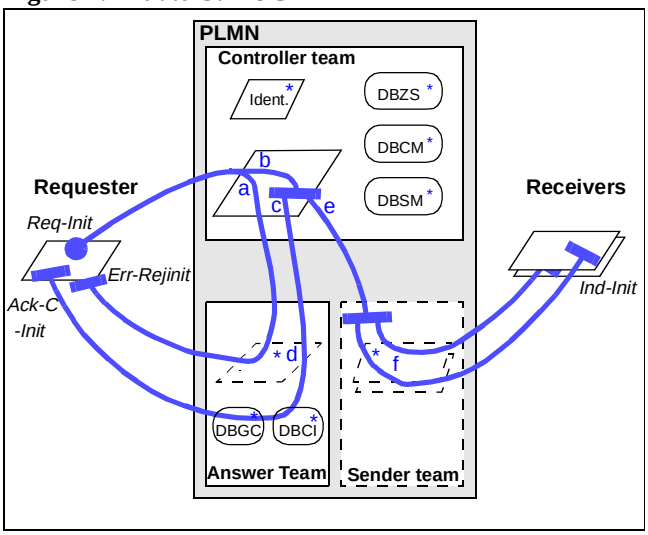
Several unidirectional and bidirectional channels allow for communication between pairs of components.

Scenarios

We described nine scenarios for the PTM-G service: six for the regular operations and three for the artificial ones. We obtained them fairly easily since GPRS services are described rather operationally, although very informally, in [10]. Our scenarios usually start with a single triggering event, leading to one or possibly many resulting events.

Figure 4 presents the bound map of our Initiate Call example. Channels have been removed from the picture to make it simpler. The stars (*) indicate that the information within these components has changed. This UCM has used one *OR-Fork* to express a choice between responsibilities a and b, and two *AND-Fork* to introduce concurrent paths. For instance, c and e are performed concurrently after b. This scenario captures the causal relationships expressed in the informal requirements.

Figure 4: *Initiate Call* UCM



Additional information on the responsibilities is provided in Table 1. At this point, it is possible to detail their type (request, error, acknowledgement, indication, condition, or internal activity), parameters, allocation to a component, and additional comments. Conditions can be refined more formally once the data types and data structures are known.

Table 1: Responsibilities information for Initiate Call UCM

Resp.	Type	Parameters	Alloc.	Comments
Req-Init	Request	M_ID, IMGI, DTM, QoS, GeoZone, join_leave, send_to_all, call_not	Requester	Request creation of a call, requester attached
Err-Rejinit	Error	IMGI, M-ID, cause	Requester	Request rejected (wrong GeoZone, wrong IMGI,...)
Ack-C-Init	Ack.	IMGI,C-ID, Cipher_Key	Requester	Acknowledgement of call creation
Ind-Init	Indic.	IMGI, C-ID	Receivers	Indication of call creation
a	Cond.		Controller	Request not accepted
b	Cond.		Controller	Request accepted
c	Internal		Controller	Update DBCM, DBSM
d	Internal		Informer	Update DBGC, DBCI
e	Cond.		Controller	Call Notification needed
f	Internal		Senders	Sending notification

LOTOS Specification

The process structure of the specification is derived from the architecture found in Figure 3. Each process, object, and team is mapped to a LOTOS process (except for DBZS, DBCM, and DBSM, which become parameters of process Controller in order to minimize the number of messages). Containment relationships are also maintained (e.g., Answer_Team is defined within PLMN). Channels become gates on which processes synchronize.

Several data types, mainly enumerations and lists of complex data structures, were defined to specify our databases, variables, and parameters.

We synthesized the behavior of each component from all the responsibilities that have been allocated to them. We had to make sure that the causality relationships defined in the UCMs were preserved. We therefore considered, for each component, all the paths that were going through it. Finally, causal relationships across components were refined as exchanges of messages.

This resulted in a LOTOS specification with 1140 commented lines of code for the ADTs and 1400 lines for the behavior part (the processes).

VALIDATION

LOTOS Testing Theory

Test cases derived from scenarios can be composed with the specification to detect possible errors. The verdict of this composition falls into one of the following categories:

- ***Must pass***: all the possible executions were successful.
- ***May pass***: some executions were successful.
- ***Reject***: all executions failed.

LOLA [5] is the tool we used to automate the testing of our specification. The tests themselves are expressed as LOTOS processes.

Test Cases Derivation

The ultimate goal of testing is to detect errors as soon as possible, especially in the specification. Our aim was to validate the specification against the functional requirements by using a test suite derived from the UCMs.

We could not generate an exhaustive test suite because of the state explosion problem we almost always encounter with concurrent systems (especially when we also consider data). We could, however, derive a sound test suite from each scenario. The test selection is based on UCM path exploration (all end-to-end paths, all combinations, all concurrent executions, etc.) and on predicate coverage (all conditions, all sub-expressions of a condition, etc.), similarly to white-box testing in traditional software engineering [15]. The level of coverage depends on the critical nature of some paths, or their cost. However, in our formal prototyping approach, cost issues are not very relevant, as the cost of testing is very low.

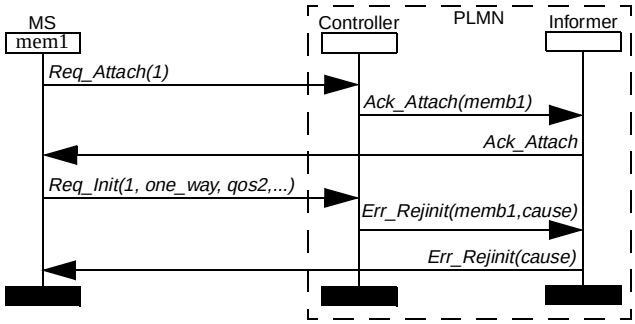
Usually, test cases include preambles and verification steps. In a typical test case for the Initiate Call operation, we must first have a sequence of events ensuring that the requester is attached to GPRS. We could then use a Call Status operation to verify that the call was correctly initiated.

The following LOTOS process is a short black-box test case. It checks that the Initiate Call request is rejected when the MS does not have sufficient privileges. In this case, database DBSM (not shown here) initially contains information stating that *mem1* is not an initiator. As a preamble, *mem1* has to attach to GPRS. *Test_Init_Call4* corresponds to the route <Req-Init; a; Err-Rejinit> in Figure 4.

```
process Test_Init_Call4[ch_Uplink, ch_Downlink, Success]: noexit :=
   ch_Uplink !mem1 !Req_Attach !Encode(1 of Geozone);
   ch_Downlink !mem1 !Ack_Attach !nomsg;  (* End of preamble *)
   ch_uplink  !mem1 !Req_Init !encode(1, one_way, qos2, NoGeozone,
                              join_leave_ind, send_to_all_ind, no_call_not);
   ch_downlink !mem1 !Err_Rejinit !encode(cause); (* End of path *)
   Success; stop
endproc
```

LOLA can verify that all possible executions of this test terminate successfully. One possible trace is illustrated as a MSC in Figure 5, where relevant components and internal messages within the PLMN are shown.

Figure 5: MSC from Test_Init_Call4 Execution Trace



35 test cases were generated for our system, including 5 test cases for Initiate Call, resulting in a total of 780 lines of code. While the length of these test cases varied between 2 and 31 events, the length of the 1933 execution traces varied between 3 and 155 events (including internal events).

Coverage Measurement

Once errors that have been detected are corrected, we would like to assess the coverage of our test suite in order to check that it is sufficient according to some criteria. In our case, we want to achieve a *functional* coverage based on the functionalities expressed in the requirements and based on the *structure* of the specification (and its underlying *labeled transition system*—LTS).

The insertion technique we used is different from probe insertion methods for structured sequential programs [16]. We basically insert probes at strategic places in the processes of the LOTOS specification (before **stop**, **exit**, and process instantiations) that are candidates for being leaves in their respective LTS.

When the coverage is achieved, the test suite can be reused for regression testing (when we modify the requirements) or for testing further refinements leading to the implementation, and ultimately for testing the implementation itself.

Observations

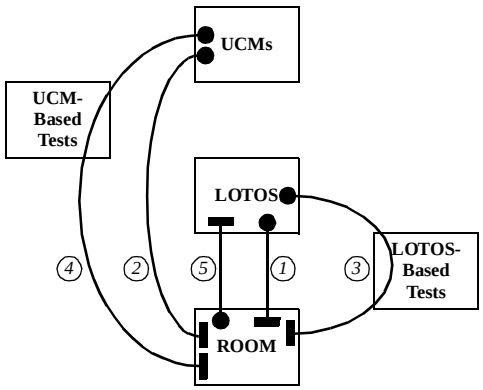
We successfully executed all 35 test cases (1933 execution traces) in 3 minutes on a Sparc Ultra. We first tested the PLMN alone, and then we specified a typical mobile station and tested its composition with the PLMN. Our test cases led to unexpected traces that were used to diagnose bugs in the specification (due to the ADTs, to the conditions, or to unfeasible synchronizations between processes). Probes helped uncover unreachable code in the specification, and they supported the improvement of the test suite.

CONNECTIONS BETWEEN LOTOS, USE CASE MAPS, AND ROOM

Several Integration Strategies

The ROOM methodology [18] and ObjecTime, the tool that supports it, offer powerful concepts and features for the detailed design, at a level of abstraction lower than the one used for our LOTOS prototypes. We envisage a software development strategy that includes UCM (for use case description and design), LOTOS (for formal specification and validation), and ROOM (for implementation). In further research, we aim at exploring connections between these three notations. Possible areas of research include the five strategies illustrated in Figure 6 and described below.

Figure 6: Five Integration Strategies



1. *Implementing a LOTOS specification using ROOM as an intermediate representation.*

The scenario-based approach, described above, allows us to iterate towards a LOTOS prototype which satisfies the requirements. Furthermore, the application of validation techniques to the LOTOS prototype offers considerable confidence that it doesn't exhibit undesirable behaviour, such as deadlocks, and that it does exhibit the desired behaviour. We would like to transfer this confidence to the final implementation, and deriving code in some automatic or semi-automatic way from the specification offers a way of doing this.

Earlier work on implementing formal specifications [5] have tended to use C as a target language, creating ordinary functional code. For reasons of reusability and maintainability, among others, it would be desirable if we could target an object-oriented implementation, perhaps using C++ or Java as the target language. Targeting an object-oriented language is not enough, however, to ensure that the resulting code offers the benefits of object-orientation. We suggest that using the ROOM modeling language as an intermediate step will help us to focus on encapsulation and possible inheritance of behaviour, producing better-quality code. It would also help to bridge the relatively large gap between high-level specification languages and implementations, while at the same time allowing for the reuse of the available code generation features of ObjecTime.

2. *Creation of a ROOM model from UCMs.*

Earlier work by Francis Bordeleau and Ray Buhr [6] illustrated a methodology for creating ROOM models from scenarios modeled as UCMs. Each UCM was converted into a high-level Message Sequence Chart (hMSC) and a collection of ordinary Message Sequence Charts (MSCs). The latter are then bound to a ROOM structure, where the behaviour of components is defined as hierarchical finite state machines. Architectural elements used in this paper, such as the ones in Figure 3, would therefore be substituted with elements from the ROOM graphical notation.

This approach provides useful thinking tools and traceability between the detailed design, the scenarios, and the requirements, but no formal validation strategy has been proposed.

3. *Testing a ROOM model using test cases derived from a LOTOS specification.*

Concurrent with the creation of a LOTOS prototype, we may create test cases which describe the correct behaviour of the specification. They can also be derived from the prototype according to a selected test derivation strategy [5]. Having created a LOTOS test suite, we may wish to convert these test cases into a form suitable for application to the ROOM model or the final implementation, to ensure that these conform to the specification.

4. *Testing a ROOM model using test cases derived from UCMs.*

Because UCMs describe scenarios which we wish the system to support, we may derive test cases from UCMs, which may then be converted into a form suitable for application to the ROOM model or the final implementation. This would help validating the ROOM model against the requirements.

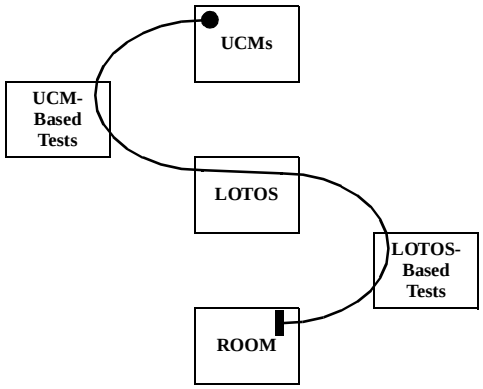
Our approach allows the definition of a test suite, from UCMs, that may be used for systematic validation (according to selected coverage criteria) of the model with respect to requirements. This could represent an important contribution to the ROOM methodology.

5. *Validation of a ROOM model using LOTOS.*

Deriving a LOTOS specification from a ROOM model would allow the developer to take advantage of the V&V theory (including testing) available in the former to formally verify the properties of the model. We are aware of some previous work by Olivier Basset and Francis Bordeleau in the automated generation of LOTOS specifications from ROOM models.

The current LOTOS standard and tools focus on reactive systems. However, real-time issues could also be considered with LOTOS extensions such as RT-LOTOS [9], for which a simulation tool is already available, and the upcoming E-LOTOS standard [17]. An interesting research direction could be the verification of ROOM models with time using such new specification techniques.

Figure 7: Combining Strategies 3 and 4



Obviously, these strategies are not mutually exclusive. In Figure 7, strategies 3 and 4 are merged to reflect an immediate use of the approach suggested in this paper. Combining the test case generation from UCMs with the creation of a LOTOS prototype (generated from UCMs) increases confidence in the test suite as it is validated with respect to the requirements, the UCMs, and the LOTOS prototype, when measuring the achieved functional and structural coverage. Such test suite can then be used to validate the ROOM detailed design, and later on to validate the implementation. The ROOM model could be derived from a LOTOS specification (strategy 1), derived from UCMs (strategy 2), or even generated using ObjecTime.

Whenever two or more metholodogies are combined, a major challenge is to use each of them for best mutual support. This may require finding new ways of using them.

Structuring Specifications for Implementation through ROOM Models.

LOTOS specifications may be written in a number of different styles, each being applicable to different needs [5]. An integral part of our research on the connections between LOTOS, UCMs and ROOM will be the determination of appropriate styles of specifications. For example, a LOTOS specification could be written with the intention that ultimately be implemented through the ROOM modeling technique to an object-oriented language. Important issues to be considered in the creation of this specification style include:

- *Encapsulation of Actors*. Actors should be clearly identified, and should hide their internal structure and behaviour, communicating through relatively few, well-defined ports. This can be done in LOTOS' resource-oriented style.
- *Hierarchical Organization of Behaviour.* ROOMcharts allow behaviour to be specified at varying levels of abstraction. LOTOS similarly allows us to specify behaviour in terms of a few processes, whose internal structure is revealed later in the specification.
- *Synchronous vs. Asynchronous Semantics*. LOTOS is based on a notion of synchronous communication, in which processes must agree to synchronize. This contrasts with Message Sequence Charts and ROOM, in which asynchronous communication is typical. Synchronous communication may be more appropriate for high-level design and formal verification. However an implementation-oriented specification may have to deal with issues of unreliable media, etc. This amounts to a refinement of the LOTOS specification by means of intermediate processes.
- *Multiway Rendezvous vs. Two-Way Communication*. LOTOS allows more than two processes to synchronize on a message. This may be regarded as an abstraction from implementation behaviour which will typically involve message passing between two entities. During the process of deriving an implementation from a formal specification, instances of multi-way synchronization must be converted to some form of two-way message passing. Earlier work [5] has indicated a number of methods by which this may be done.
- *Directionality of Communication*. The synchronous nature of LOTOS semantics abstracts away from the directionality of message passing. The structuring of LOTOS specifications with a view to implementation must make more explicit which entity is the sender of a message and which is the recipient. This may be achieved through well-defined message patterns or event structures.
- *Abstract Data Types*: The use of ADTs, a powerful yet abstract technique for the description of data, should be restricted so that automated or semi-automated translation towards programming languages (such as C++ and Java) is possible. Some LOTOS tools allow to replace ADTs by their implementation [9].
- *Distributed Systems*: ROOM and LOTOS both target, among other things, distributed systems. Mappings of distributed entities, and also of components that support this distribution (ports, communication channels, etc.), need to be well established between the two models.
- *Real-Time Systems*: Current LOTOS specifications tend to abstract from quantitative time either by not specifying it, or by using specific actions to express timely events. If hard real-time constraints need to be described, a LOTOS style would have to reflect them systematically. LOTOS enhancements that support time exist, however tool support currently is weaker.

CONCLUSION

We have demonstrated an iterative and incremental design approach based on a visual scenario notation (UCM) and a FDT (LOTOS) that leads to the generation of validated system prototypes and test suites. Although we could not discuss this in the text, it helped us unveil several ambiguous and incomplete descriptions in a draft standard document of the GPRS Group Call service.

Based on the current experiment and on previous ones with an artificial *Group Communication Server* [3] and several *Wireless Intelligent Network* (WIN) features, we observed several interesting points. Scenarios described using UCMs focus on causality instead of message exchanges, and consequently they can be developed independently of the underlying architecture. Such scenarios tend to be very reusable, and they can serve as a basis for the generation of functional test cases. These tests can be used to validate a

formal prototype that can help detecting interactions between scenarios. The prototype and the test suite can further be validated through probes inserted in the specification and structural coverage measurement.

The Group Call prototype and test cases were mainly done by one of us (P. Forhan, *stagiaire* from INT) who initially was not familiar with any of GPRS, LOTOS, UCM, this methodology, or LOTOS tools. Nevertheless, it took him less than 5 months to gain a better understanding of the Group Call service, and to produce useful documentation, a validated specification, and a test suite in which we have a high level of confidence.

We believe that our approach can facilitate the early stages of typical requirements engineering and design within standardization processes. We intend to pursue the approach in several directions, including early consistency and completeness checking of UCMs through some data dictionary, formal test case generation (for implementation testing), semi-automated synthesis through message exchange patterns associated to causality relationships, and better traceability between the models.

Finally, the LOTOS FDT offers a powerful means of formally verifying the behaviour of a design, and provides a means for test case generation. Deriving an implementation from the formal prototype can increase confidence in the correctness of the behaviour of the final code. Using ROOM as an intermediate step structures the implementation in an object-oriented style, but this might require the specification to be written in a style suitable for conversion. Deriving implementation tests from the test cases produced from the LOTOS specification or from UCMs further enhances our confidence in the detailed design and the implementation, and may provide a regression test suite for future development. Many promising research directions related to the integration of LOTOS, UCMs, and ROOM have been identified in this document. Further investigation is underway.

ACKNOWLEDGEMENTS

We kindly acknowledge FCAR, NSERC, ObjecTime, and Motorola for their support, the LOTOS research group (especially Jacques Sincennes, Brahim Ghribi, and Laurent Andriantsiferana) for their usual yet very appreciated collaboration, and Ray Buhr for having developed and taught us the Use Case Maps methodology.

REFERENCES

1. Amyot, D. *Formalization of Timethreads Using LOTOS*. M.Sc. Thesis, Dept. of Computer Science, University of Ottawa, Canada (1994). http://lotos.csi.uottawa.ca/~damyot/phd/mscthese.pdf
2. Amyot, D., Bordeleau, F., Buhr, R. J. A., and Logrippo, L. “Formal support for design techniques: a Timethreads-LOTOS approach”. In *FORTE VIII, 8th International Conference on Formal Description Techniques* (Montréal, October 1995), Chapman & Hall, 57-72. http://lotos.csi.uottawa.ca/~damyot/phd/forte95/forte95.pdf
3. Amyot, D., Logrippo, L., and Buhr, R.J.A. “Spécification et conception de systèmes communicants : une approche rigoureuse basée sur des scénarios d’usage”. In *CFIP 97, Ingénierie des protocoles* (Liège, September 1997), Hermès, 159-174. http://lotos.csi.uottawa.ca/~damyot/phd/cfip97/cfip97.pdf
4. Amyot, D. *Annotated UCM Grammar — Working Document* (December 1997). http://www.sce.carleton.ca/rads/agents/grammar/
5. Bolognesi, T., van de Lagemaat, J., and Vissers, C. *LOTOSphere: Software Development with LOTOS*. Kluwer Academic Publishers, The Netherlands (1995).
6. Bordeleau, F. and Buhr, R.J.A.“The UCM-ROOM Design Method: from Use Case Maps to Communicating State Machines”. In: *Proceedings of the Conference on the Engineering of Computer-Based Systems*, Monterey, USA (1997). http://www.sce.carleton.ca/ftp/pub/UseCaseMaps/UCM-ROOM.ps
7. Buhr, R.J.A. and Casselman, R.S. *Use Case Maps for Object-Oriented Systems*, Prentice-Hall, (1995).
8. Buhr, R..J.A., *Scenario-Path Signatures as Architectural Entities for Complex Systems* (December 1997). http://www.sce.carleton.ca/ftp/pub/UseCaseMaps/ucmUpdate.pdf
9. Courtiat, J.-P., and de Oliveira, R.C. “A Reachability Analysis of RT-LOTOS Specifications”. In *FORTE VIII, 8th International Conference on Formal Description Techniques* (Montréal, October 1995), Chapman & Hall.
10. ETSI, Digital Cellular Telecommunications system (Phase 2+); *General Packet Radio Service (GPRS); Service Description Stage 1 (GEM 02.60), Version 2.0.0* (November 1996).
11. ISO, Information Processing Systems, Open Systems Interconnection, *LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour*, IS 8807 (1988).
12. ITU, *Recommendation Z. 120: Message Sequence Charts (MSC)*, Geneva (1996).
13. Jacobson, I., Christerson, M., Jonsson, P., and Övergaard, G. *Object-Oriented Software Engineering, A Use Case Driven Approach*. Addison-Wesley, ACM Press (1993).
14. Mouly, M. and Pautet, M.-B. *The GSM System for Mobile Communications*. Cell & Sys (1992).
15. Pressman, R. S. *Software Engineering — A Practitioner’s Approach*. McGraw-Hill, USA (1987).
16. Probert, R.L. “Optimal Insertion of Software Probes in Well-Delimited Programs”, *IEEE Transactions on Software Engineering*, Vol 8, No 1 (January 1982), 34-42
17. Quemada, J. (editor). *Working Draft on Enhancements to LOTOS*. ISO/IEC JTC1/SC21/WG1. Project 1.21.20.2.3 (January 1997).
18. Selic, B., Gullekson, G., and Ward, P.T. *Real-Time Object-Oriented Modeling*, Wiley & Sons, 1994.