Introduction to the User Requirements Notation: Learning by Example

Daniel Amyot [⋆]

SITE, University of Ottawa
800 King Edward
Ottawa (ON), Canada, K1N 6N5

Abstract

Recognizing the need for a notation that would be used in the very first and often informal stages of the development cycle, the International Telecommunication Union (ITU-T) initiated a question on a User Requirements Notation (URN), which will be standardized as the Z.150 series of Recommendations. URN supports the development, description, and analysis of requirements for telecommunications systems and services, as well as for other types of complex reactive, distributed, and dynamic systems. Through a wireless telephony example, this paper gives an overview of the core elements and typical usage of the two complementary notations comprised in URN. The Goal-oriented Requirement Language (GRL) is used to describe business goals, non-functional requirements, alternatives, and rationales, whereas Use Case Maps (UCM) enable the description of functional requirements as causal scenarios. This paper also briefly explores methodology elements and the complementarity between URN and the existing ITU-T languages.

Key words: Goals, GRL, ITU-T Languages, Requirements Engineering, Scenarios, UCM, User Requirements Notation

1. Introduction

Requirements engineering has become an essential part of all development processes, especially in the area of complex reactive and distributed systems, which includes telecommunications services. However, few standardized notations and techniques can address the needs specific to visualizing and analyzing functional (behaviour) and non-functional requirements (NFRs, such as performance, cost, security, and usability). The User Requirements Notation (URN), to be published by the International Telecommunications Union (ITU-T) in 2003, pioneers work in the standardization of visual notations that address these needs [16]. URN will allow software engineers and requirements engineers to discover or specify requirements for a proposed system or an evolving system, and review such requirements for correctness and completeness. Standardization of a formally defined notation used for capturing and analyzing user requirements aims to make requirements engineering activities more rigorous and predictable and the results yielded by these activities clearer, more consistent, correct, and complete. These results should lead to a reduction of development costs, earlier delivery of products to market, and increased customer satisfaction.

URN focuses on *user requirement* (desired goals or functions that users or other stakeholders expect the system to achieve) but it also enables the

[⋆] Supported financially by NSERC, Nortel Networks, and the University of Ottawa.
Email address: `damyot@site.uottawa.ca` (Daniel Amyot).

description of their refinement as *system requirements* (expression of ideas to be embodied in the system or application under development). Like most notations in ITU-T's family of languages, URN is graphical, because graphical presentations are often compact and evocative.

The URN is intended for use in requirements descriptions in specifications developed by national and international standards organizations. In ITU, requirements descriptions are often called Stage 1 descriptions (e.g. Q.65 [13]). The URN is also intended for use by commercial organizations developing requirements specifications for new products and product extensions; these specifications are not necessarily governed by standards.

In order to cope with the ever-increasing complexity of requirements engineering activities for emerging telecommunications services, reactive systems, and distributed systems, URN is defined to have the following capabilities [16]:

(1) capture user requirements when very little design detail is available;

(2) describe scenarios as first class entities without requiring reference to system subcomponents, specific inter-component communication facilities, or sub-component states;

(3) facilitate the transition from a requirements specification to a high-level design involving the consideration of alternative architectures and the discovery of further requirements that must be vetted by the stakeholders;

(4) have dynamic refinement capability with the ability to allocate scenario responsibilities to architectural components;

(5) be applicable to the design of policy-driven negotiation protocols involving dynamic entities;

(6) facilitate detection and avoidance of undesirable interactions between services (also called *features* in this paper);

(7) provide insight at the requirements level that enables designers to reason about feature interactions and performance trade-offs early in the design process.

(8) provide facilities to express, analyze and deal with goals and non-functional requirements;

(9) provide facilities to express the relationship between goals and system requirements;

(10) provide facilities to capture reusable analysis and design knowledge related to know-how for addressing non-functional requirements;

(11) provide facilities to trace and transform requirements to other languages (especially ITU-T notations and UML);

(12) provide facilities to connect URN elements to external requirements objects;

(13) provide facilities to manage evolving requirements.

Since it is next to impossible to find a single notation that could satisfy all these ambitious objectives, the current proposal for URN combines two complementary notations: the *Goal-oriented Requirement Language* (GRL) for goals and NFRs, and *Use Case Maps* (UCMs) for scenarios. The nature and content of GRL and UCMs will be illustrated in Sections 3 and 4 respectively. As the standard is still evolving and not yet completed, this tutorial focuses on the core notation elements that are the most stable and the most useful. More advanced concepts and notation elements may be tackled at times, but they will not be systematically covered.

In order to illustrate typical applications of the notations, we will use an example from the wireless telephony domain. Though inspired from previous work on the application of UCMs to 2.5G and 3G wireless standards in TIA, 3GPP, and 3GPP2 [2,4,12,27], and especially from the work done by Andrade and Logrippo [6,7], this partial example is artificial and is not a case study as such. It is meant to illustrate a variety of notation elements in a context simpler than real wireless systems and standards, which would need to address many more concerns. In order to help the readers unfamiliar with background terminology and

structural concepts commonly found in wireless telephony, a brief introduction to TIA's *Wireless Intelligent Network* (WIN) is given in Section 2.
URN does not impose any process or methodology on its users. However, elements of typical processes and relations to other languages are explored in section 5, followed by our conclusions. As a convenience to the reader, appendix 6 provides a summary of the main notation elements found in GRL and UCM.

2. WIN Concepts and Terminology

Our URN examples will be presented in the context of the Wireless Intelligent Network standard, which has been developed to drive Intelligent Network capability into ANSI-41-based wireless networks [27]. WIN separates call processing intelligence and feature functionality from network switches, includes mobility management functions, and offers a diversity of enhanced services to subscribers. WIN was developed in multiple phases and covered various features from call screening to flexible billing to location-based services.
Figure 1 depicts WIN's *Distributed Functional Model* with computational objects, called functional entities (FEs), and their relationships. A grouping of actions across one or more FEs, when coordinated by communication flows, provides the required WIN service execution. Additional FEs for service management also exist but are not shown here. It is not necessary to understand all these FEs, but some of them will get used in our URN example and they will be explained further where appropriate.
Additionally, WIN possesses a *Network Reference Model*, which defines network entities (NEs) similar to those found in typical wireless telephony systems (e.g. GSM and UMTS). Figure 2 illustrates interfaces connecting common NEs. The core elements include the mobile switch named MobSC [1], the home and visiting profile databases (HLR, VLR), and the service control point (SCP) where additional service logic can be found. Usually, FEs are allocated to NEs, but such mapping is vendor-specific and is not dictated by the WIN standard. The reader is invited to use these two figures as references when wireless terminology and acronyms are used in our URN examples.

[1] Usually called *MSC* in the literature, but this paper uses MobSC to avoid conflicts with the MSC acronym used for Message Sequence Charts

3. Goals, Non-Functional Requirements, and GRL

In software development practice, many non-functional requirements (NFRs) are stated only informally, making them difficult to analyze, specify and enforce during software development and to get validated by customers and user once the system has been built. Goal-oriented modelling, which aim to address such issues, has been used in the requirements engineering community for a number of years. A *goal* is an objective or concern used to discover and evaluate functional and non-functional requirements.
The Goal-oriented Requirement Language (GRL) is a recent addition to the growing list of goal-modelling techniques built on the well-established *NFR framework* [11]. GRL captures business or system goals, alternative means of achieving goals (either objectively or subjectively), and rationales for contributions and decisions. GRL intends to provide some of the capabilities (7 to 10) identified for URN in the introduction.

3.1. Basic GRL Notation

In our ongoing example, we plan to discover and analyze various requirements for a generic wireless telephony system. Such a system has obvious objectives such as high performance, evolveability, scalability, interoperability, security, reliability, and so on. Business concerns also include low cost, fast time to market, high customer satisfaction, etc. For a wireless telephony system, several concerns such as usability are much less important for the network than for the mobile device (telephone) itself.
GRL offers graphical means of describing and structuring such concerns. *Softgoals*, which represent goals that are somewhat fuzzy in nature and can never be entirely satisfied, capture high-level objectives. They are shown graphically as clouds, and they can be connected to each other, in an

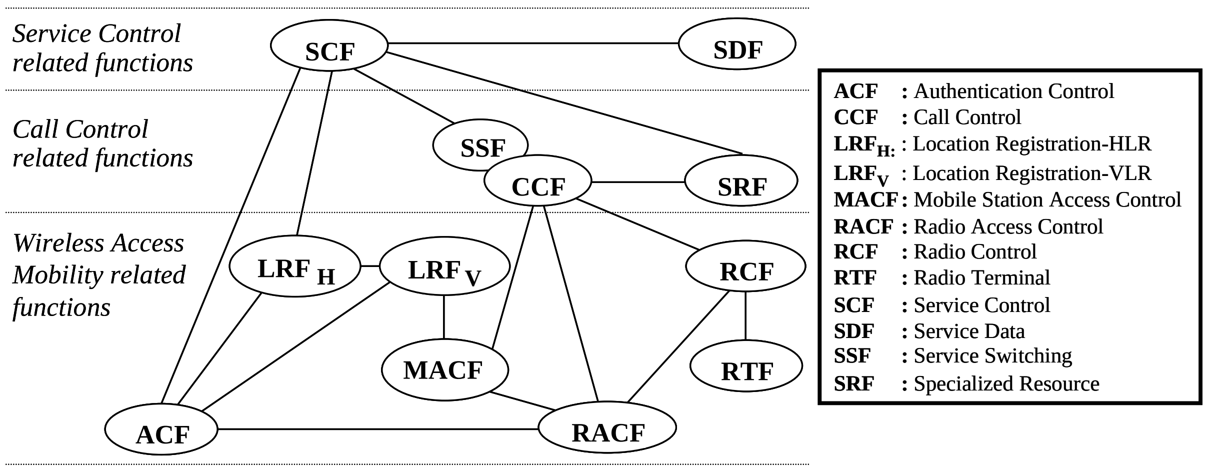

Service Control related functions
Call Control related functions
Wireless Access Mobility related functions
SCF
SDF
SSF
CCF
SRF
LRF H
LRF V
RCF
MACF
RTF
ACF
RACF
ACF : Authentication Control
CCF : Call Control
LRFH: : Location Registration-HLR
LRFV : Location Registration-VLR
MACF : Mobile Station Access Control
RACF : Radio Access Control
RCF : Radio Control
RTF : Radio Terminal
SCF : Service Control
SDF : Service Data
SSF : Service Switching
SRF : Specialized Resource

Fig. 1. Wireless Distributed Functional Model (adapted from [27])

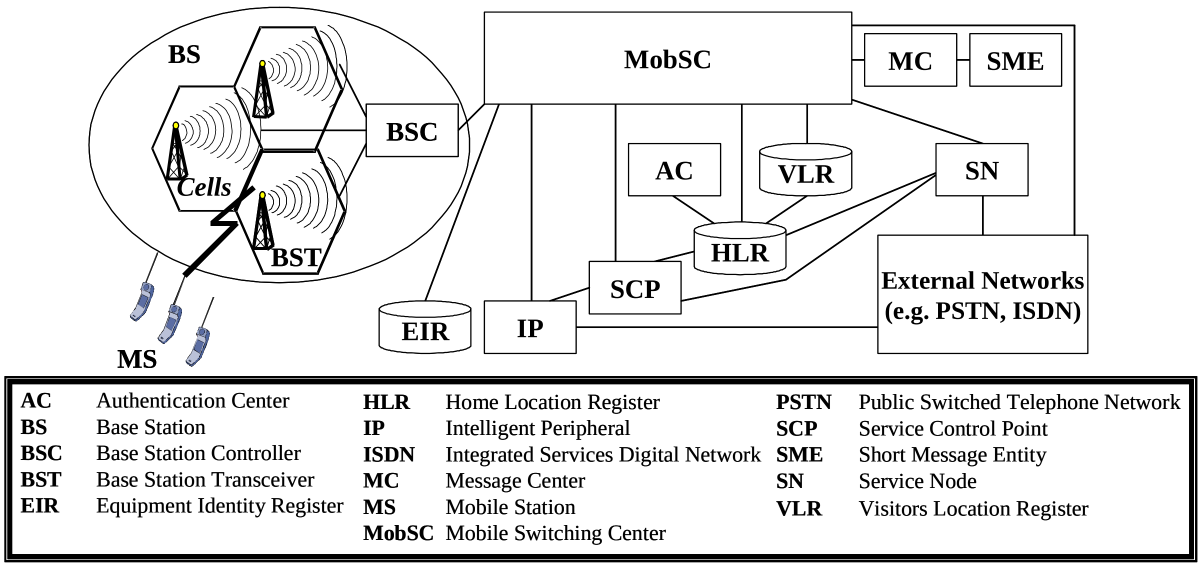

BS
Cells
BST
MS
BSC
MobSC
MC
SME
AC
VLR
SN
HLR
SCP
EIR
IP
External Networks (e.g. PSTN, ISDN)
AC Authentication Center
BS Base Station
BSC Base Station Controller
BST Base Station Transceiver
EIR Equipment Identity Register
HLR Home Location Register
IP Intelligent Peripheral
ISDN Integrated Services Digital Network
MC Message Center
MS Mobile Station
MobSC Mobile Switching Center
PSTN Public Switched Telephone Network
SCP Service Control Point
SME Short Message Entity
SN Service Node
VLR Visitors Location Register

Fig. 2. Wireless Network Reference Model (adapted from [27])

AND/OR graph, using *contribution links*. In the GRL diagram of Figure 3, both Maximum Hardware Utilisation and High Throughput contribute positively to the High Performance softgoal. Contributions links can be composed using AND (all sub-goals are needed) or OR (one sub-goal is needed) constructs. A short line crossing a contribution link near the arrow indicates an AND composition. Contributions may have various degrees of impact, including positive and sufficient (*make*), positive but insufficient (*help*), unknown positive (*some+*), and their corresponding contributions on the negative side (*break, hurt*, and *some-*). A summary of GRL contribution types and symbols can be found in Figure A.1(e). Contributions are qualitative in nature because we deal with rather fuzzy and unquantifiable objectives at this level of abstraction.

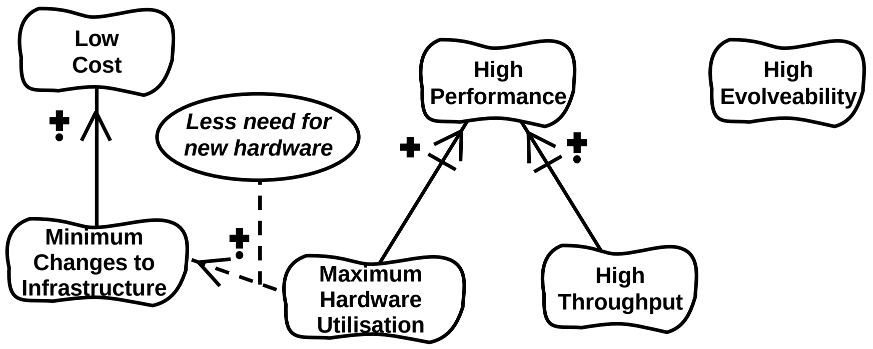

Low Cost
High Performance
High Evolveability
Less need for new hardware
Minimum Changes to Infrastructure
Maximum Hardware Utilisation
High Throughput
+
+
+
+

Fig. 3. Structuring Objectives with GRL Softgoals and Contribution Links

If we consider additional objectives, then more nodes and possibly more links are added to the graph. For instance, requirements for a wireless telephony system will be influenced by objectives such as low cost and evolveability, also identified in Figure 3. In order to minimize cost, the changes to the existing infrastructure should also be kept to a minimum. Using the hardware to its fullest extent not only contributes to the system performance, but this helps reducing the number of changes required in the system. Such side-effects are called *correlations* in GRL, and dashed arrows are used to illustrate them. Like contribution links, correlation links can be of different types (or weights). Justifications or explanations of GRL links can be added in the form of *beliefs*, shown as ellipses. Beliefs document the rationale behind various parts of a GRL model. Other stakeholders can support or argue against such beliefs while validating the model. Once validated and agreed upon, beliefs become very useful documentation items that can prevent stakeholders from having the same discussions over and over again.

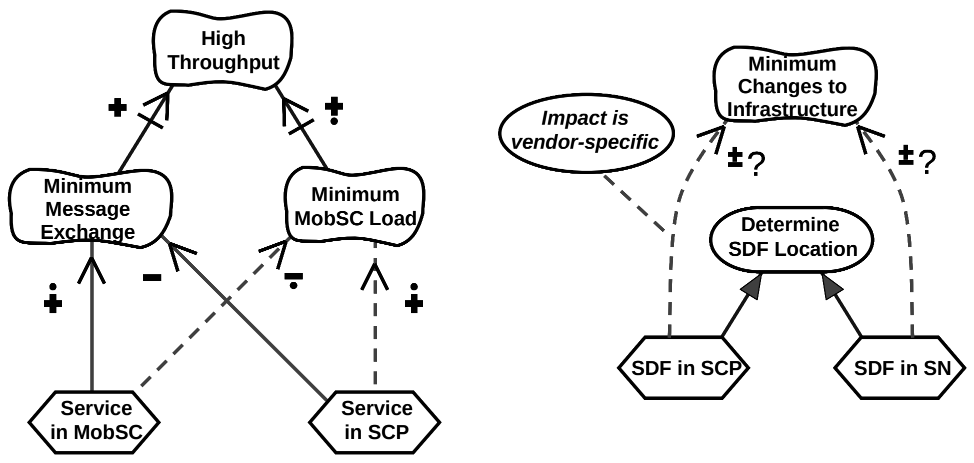
Softgoals can be decomposed and refined until a point where they become quantifiable goals or potential operational solutions. *Tasks*, shown as hexagons, are typically used to operationalize parent (soft-)goals. Figure 4a presents further refinement of the High Throughput softgoal. In particular, to minimize the exchanges of messages, two alternative solutions are foreseen: have the mobile switching center (MobSC) support the service, or leave that to the service control point (SCP). Packing additional services (independently of their nature) in the MobSC will obviously contribute very positively to the reduction of message exchanges, whereas an SCP-based option would do quite the opposite. However, these alternative tasks would have different side-effects on the additional load imposed on the MobSC.

Figure 4b presents the problem of determining a location for service data functions (SDF) as a regular *goal* with two alternative tasks. These tasks are connected to the goal with *means-end* links because they are both solutions that fully satisfy that goal. Again, these solutions could have side-effects on the existing infrastructure. However, the impact here is unknown because it is vendor-specific. For instance, one company could already have the SDF in a service node (SN), a second company would have them in a SCP, and a third one could have them elsewhere.

3.2. GRL Evaluations

The GRL models seen so far illustrate the usefulness of GRL for visualizing static relations existing between the various goals, the alternatives means to achieve these goals, their interactions, and accompanying rationales. GRL also supports an evaluation mechanism used to measure the impact of qualitative decisions on the level of satisfaction of high-level goals. Given initial degrees of satisfaction to some tasks and goals in a GRL model[2], a propagation algorithm (inspired from [11]) will guide the computation of the sat-

[2] For softgoals, we often use the term *satisficed*, which means satisfied within acceptable limits.



(a) Alternative locations for the service (b) Alternative Locations for the SDF

Fig. 4. Expressing Alternatives with GRL Tasks

isfaction level of all the other (soft-)goals in the model. Input from requirements engineers might be required to solve conflicting situations.

Figure 5 combines the GRL models from Figures 3 and 4 for a service provider whose existing wireless infrastructure has SDF in the SCP (the undecided contributions to the Determine SDF Location goal were updated accordingly). This evaluation starts with the selection of a location for services. In this example, we evaluate the option where services are located in the SCP. Different labels are added next to the tasks and (soft-)goals to show their level of satisfaction (see the legend in Figure A.1b). We usually start with the leaves of the diagrams, e.g. we first checkmark the selected tasks (SDF in SCP, and Service in SCP). Additionally, we do not yet know how utilised the hardware will be, and this is shown with the *undecided* label.

The propagation algorithm takes this initial configuration and propagates the satisfaction levels to the top through the weighted contribution links. The SDF location goal is satisfied because of the means-end link; it is sufficient for one of the alternative tasks to be satisfied for the parent to become satisfied. The Minimum Message Exchange takes the best degree of satisfaction offered by its two contributors, which means it becomes weakly denied (a high satisfaction going through a *some-* contribution amounts to a weakly denied score, which is still better than the fully denied score from the other alternative). The Minimum MobSC Load is satisficed because the service resides in the SCP. This, combined to the *some+* contribution, causes the High Evolveability to be weakly satisficed. Because of the AND composition, the High Throughput softgoal needs to take into consideration both contributions (one positive and one slightly negative). We can hence argue that the high throughput is weakly satisficed.

This goes on until all the high-level softgoals get labeled with a degree of satisfaction. Different alternative solutions can be quickly and systematically evaluated, which helps finding a global solution that maximizes the satisfaction of the highest-level goals, and hence leads to a good tradeoff between conflicting goals. In this particular GRL model, evaluating the option where the services are located in the MobSC would result in a lower satisfaction of the High Performance and of the High Evolveability. After the evaluation, the selected configuration of tasks becomes an initial set of requirements for the system, and these requirements are traceable to the business objectives.

The propagation in a GRL model is usually bottom-up, and the current techniques assume that there are no cycles caused by contributions

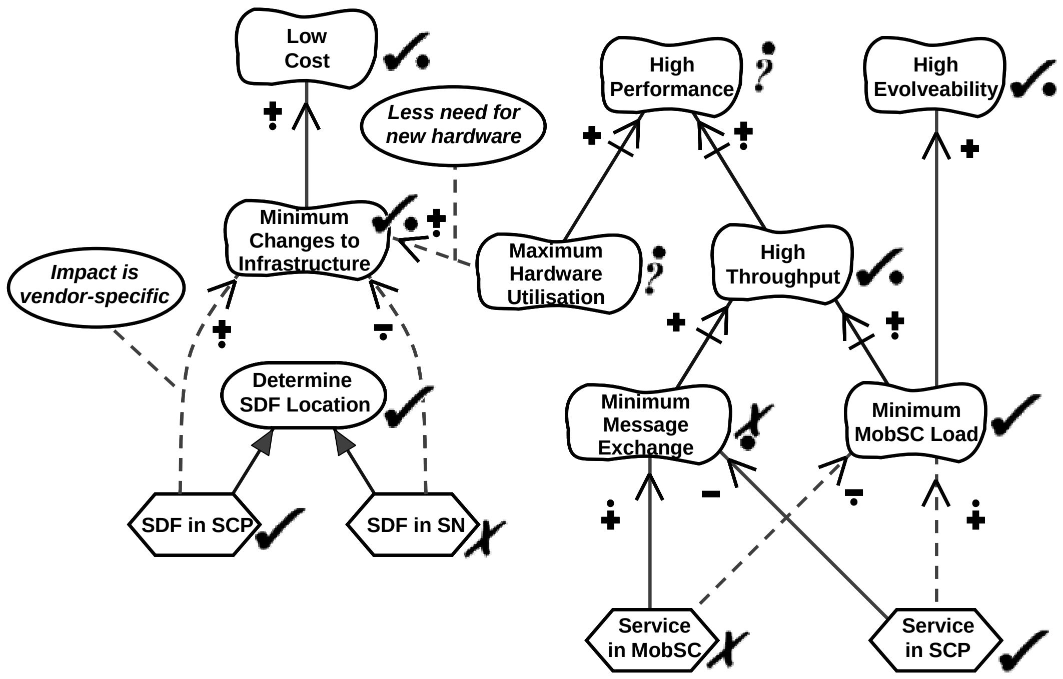


Fig. 5. Evaluation of a Candidate Combination of Solutions

links. The feasibility and usefulness of top-down propagation for root-cause analysis and of propagation in models with circular contributions (feedback) are left for future work.

3.3. Advanced GRL Notation and Tool Support

The concept of GRL *actor* is not illustrated here because its usage goes beyond the scope of this tutorial. Suffice it to say that actors are active entities that execute actions to achieve their goals. They can encapsulate GRL graphs and can have *dependencies* on other actors or on other GRL elements. Actors, which are based on Yu's agent-oriented modelling language i^* [29], are often used to do role-based analysis on social relationships. These concepts are further developed in the draft Z.151 document ([17])

GRL is currently supported by a modelling and analysis tool named OME (*Organization Modelling Environment*) [19]. OME offers a Java-based graphical environment for the creation, maintenance, and analysis of GRL models. An interactive propagation algorithm is also supported.

4. Scenarios, Functional Requirements, and Use Case Maps

A *functional requirement* is a requirement defining functions of the system under development. Modelling functional requirements of complex systems often implies an early emphasis on behavioural aspects, often captured in the form of use cases and scenarios. In a recent survey [3], it was observed that few scenario notations can offer capabilities such as those enumerated in Section 1 (1 to 7 in particular). Many notations are variants of MSCs and focus on messages and inter-component interactions. This is valuable for design, but this is often overcommitting for requirements.

As will be shown in the next sub-sections, the Use Case Map scenario notation avoids the problems cited above by focusing on causal relationships between responsibilities of one or more use cases. The relationships are said to be causal because they involve concurrency and partial orderings of responsibilities, because they link causes to effects, and because they abstract from component interactions expressed as message exchanges. UCMs are applicable to use case capturing and elicitation, use case validation, as well as high-level architectural design and test case generation. The combined, gray-box, view of behaviour and structure and the flexible allocation of responsibilities to architectural structures contribute to bridging the gap between requirements and design. UCMs provide a behavioural framework for evaluating and making architectural decisions at a high level of design, optionally based on performance analysis of UCMs. Moreover, UCMs provide their users with dynamic (run-time) refinement capabilities for variations of scenarios and structures, and they allow incremental development and integration of complex scenarios. The UCM Web Page [28] contains various applications of UCMs in the areas just mentioned.

UCMs share some commonalities with UML activity diagrams [24], but they have additional capabilities (e.g. dynamic stubs, multiple start points, hierarchical structure of components instead of swimlanes, and timers) which make them more suitable for expressing and analysing requirements for complex systems (see [3,5] for more detailed comparisons). Recently, the concept of scenario definition was also incorporated to UCMs, and it will be further explored in this section.

Note that the following UCM examples contain many abbreviations, whose meanings are summarized in Appendix B.1

4.1. Basic UCM Notation

In UCMs, a *scenario* is a partial description of system usage defined as a set of partially-ordered responsibilities a system performs to transform inputs to outputs while satisfying preconditions and postconditions. UCM *responsibilities* are scenario activities representing something to be performed (operation, action, task, function, etc.). A responsibility can potentially be associated or allocated to a *component*. In UCMs, a component is generic and abstract enough to represent software entities (e.g. objects, processes, databases, or servers) as well as non-software entities (e.g. actors or hardware).

The UCM in Figure 6 illustrates some of these concepts through the description of a simple handling procedure for cellular hand-off. Filled circles represent *start points*, which capture preconditions and triggering events (in this case, a hand-off request HOreq). *End points* capturing resulting events and post-conditions are illustrated with bars perpendicular to causal *paths*. Scenarios progress along paths from start points to end points. Paths also support responsibilities, shown as crosses. Paths can fork as alternatives (*OR-fork*) and may also join (*OR-join*). Alternative branches can be guarded by *conditions*, shown between square brackets. A condition needs to be true for the guarded path to be followed. In this hand-off example, after tuning to a new channel the signal quality might be better or worse. When it is better, the user profile is updated and the scenario may continue. Otherwise, the mobile station will tune to the previous channel. If it is still good enough ([OK]), then the scenario continues otherwise the hand-off fails (FailHO).

Although this UCM can be a standalone map, in our wireless system example it is only a submap of a larger collection of scenarios, whose root UCM is shown in Figure 7. This UCM contains many new notation elements such as direction arrows and concurrent paths. Concurrency and partial ordering of responsibilities and events are supported in UCMs through the use of (*AND-forks*) and (*AND-joins*) (see Figure A.2b). While a OR-join simply indicates overlapping of scenarios that share common paths, an AND-join is a synchronization between two or more paths which must all have been visited for the rest of the scenario to progress. Figure 7 also has multiple start points, which can be triggered independently as long as their preconditions are satisfied.

The diamond symbols are called *stubs* and are used as containers for sub-maps, which are then referred to as *plug-in* maps (because they are plugged into a stub). Any map can be a plug-in. The hand-off UCM in Figure 6 is in fact a plug-in for stub Handoff. A hand-off check is triggered periodically (CellCheck) to determine whether a new channel would result in a better communication quality. Stubs have identifiable input and output segments (*IN1*, *OUT1*, ...) connected to

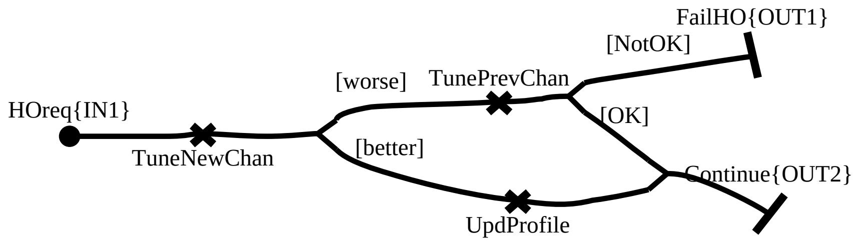


Fig. 6. UCM Model for Hand-off Handling

start points and end points in the plug-in. This *binding relationship* is also made visual in the plug-in, where the connections to the parent stub are shown between curly brackets (see Figure 6). Binding relationships ensure that paths flow from parent maps to sub-maps, and back to parent maps.

While static stubs contain only one plug-in map and are typically used for hierarchical decomposition (e.g. CM), *dynamic stubs* contain many plug-ins and are drawn with dashed diamonds (e.g. Update). Plug-ins in dynamic stubs have preconditions which are used to select among them at run-time. This collection of preconditions form the stub's selection policy. In our example, a second plug-in exists for Handoff (not shown here) and it connects *IN1* to *OUT1* when hand-offs are no longer required, e.g. when a Disconnect is initiated. The presence of multiple plug-ins in a dynamic stub enables the refinement of that stub into many potential sub-scenarios, one of which is selected at run-time according to the selection policy local to the stub.

After having authentified the call originator and updating its location record in the home database (UpdHLoc), the system needs to update visiting databases if the mobile user enters or leaves a visiting area. This can be expressed using two alternative plug-ins for stub Update, as shown in Figure 8. The first plug-in is selected when the mobile user is in the same area as before, and the visiting profile is updated if this area is not the home area. The second plug-in is selected when the mobile user has entered a different area. Different activities (deletion and creation of visitor profiles) are required to handle the various situations where the old and new areas are the home area or visiting areas (i.e. home → visiting, visiting → home, visiting → other visiting).

The UCM illustrated in Figure 9 is a plug-in for the communication management (CM) stub. The system must first acquire the resources needed to establish communication and then waits for a connection. In UCM, waiting can be done with *waiting places* (filled circles on a path) and *timers* (clock symbols). The waiting period ends when the waiting place or the timer receives an event coming from the environment or from another UCM scenario. This can be done asynchronously or synchronously (through juxtaposition of a path or of an end point, see Figure A.2e). When it is not triggered in a timely way, a timer stops and its timeout path (the path with a zigzag symbol) is taken. We have such a situation here when the terminating party does not answer within a certain amount of time, which causes the resources to be released and the plug-in to exit. The zigzag symbol used in Figure 9, generated by the UCM Navigator tool (to be discussed in section 4.4), is only an approximation of the official one.

4.2. UCM Component Structures

UCM responsibilities and other elements included within the boundaries of a component are said to be allocated to that component. Generic components are represented as rectangles. They have names and can have multiple attributes, which are not covered in this paper. Components can also contain sub-components. This is useful to capture various mappings of Functional Entities to Network Entities. For instance, NEs such as HLR, SCP, or SN (Figure 2) can all include various FEs such as SDF and SCF (Figure 1). These two degrees of freedom (responsibilities to components,

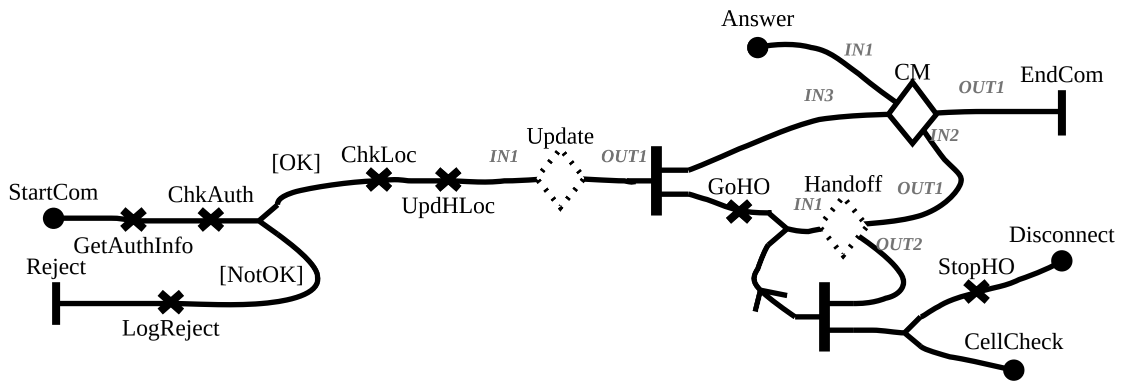


Fig. 7. Root UCM for the Simplified Wireless System

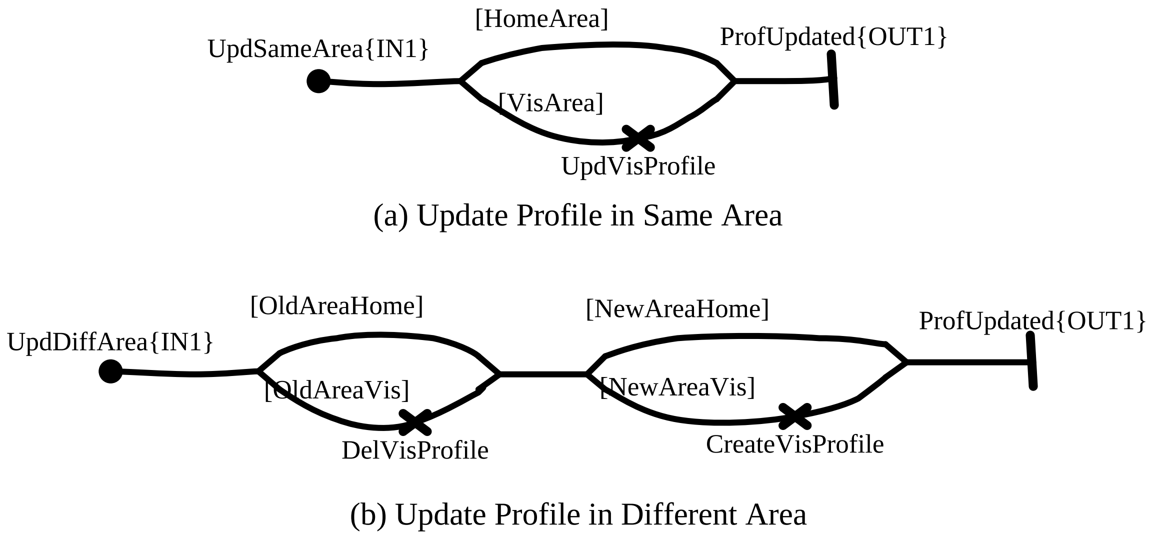


Fig. 8. Plug-ins for the Update Dynamic Stub

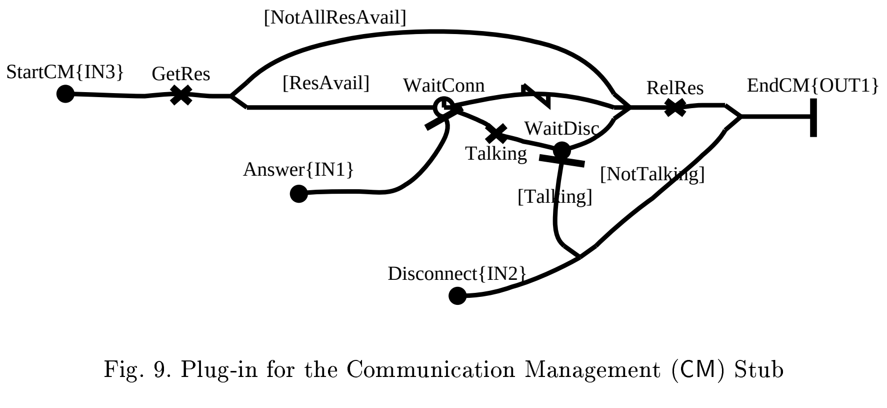


Fig. 9. Plug-in for the Communication Management (CM) Stub

and sub-components to components) greatly improve the ease with which alternative architectures can be evaluated and with which scenarios can be adapted and reused in different contexts.
For the sake of simplicity, we will focus on the very beginning of the root map from Figure 7, which handles the authentification of the originating party. The allocation of these responsibilities to FEs and NEs can be done in a number of ways, as suggested in the GRL models in Figure 4. Three options are illustrated in Figure 10. In the first one, the authentication is handled entirely by the MobSC. The second option also performs authentication in the MobSC, only this time the matching authentication information needs to be acquired from the SDF located in a Service Node. Similar scenarios where the SDF is in a SCP or in a HLR are also possible. In the third option, the service is located in the SCP (in its SCF functional entity), where we also find the SDF. Again, other variants exist. In particular, additional options not being considered here include the handling of the authentification by specialized functions (such as an ACF, see Figure 1), which could be located in various places (e.g. in the MobSC, or in a dedicated Authentication Center, as shown in Figure 2).
With UCM, different structures suggested by the alternatives identified in a GRL model or other means can be quickly expressed and evaluated by moving responsibilities from one component to another, or by restructuring the components. Other scenario notations usually require much more details and efforts to achieve this, and often this distracts engineers and other stakeholders from addressing core issues.

4.3. UCM Scenario Definitions, Path Traversal, and Transformations

Our UCM model integrates dozens of scenarios in a very compact form, which is not always suitable for understanding when used as is. *Scenario definitions* offer the possibility to describe and extract individual scenarios from a complex UCM model. Such scenarios can be used to explain and visually emphasize particularly interesting cases, to analyze potentially conflicting situations (for instance, between interacting services), or to generate other types of models (e.g. MSCs, UML sequence diagrams, and test cases).
A scenario definition is composed of four elements: a name, a list of starting points, a set of initial conditions, and (optionally) a set of post-conditions. Conditions are expressed using an abstract path data model comprised of global Boolean variables. These variables are also used in guarding conditions, timers, and selection policies, and their values can be modified inside responsibilities. A UCM model with variables can be traversed using scenario definitions in combination with a *path traversal* mechanism. Guidelines for such mechanisms are provided in the draft UCM standard ([18]).
Any Boolean expression can be used to formalize our conditions. Given a Boolean variable Authentified, the conditions [OK] and [NotOK] found in Figure 10 could be formalized as Authentified and !Authentified respectively. In the Update stub of Figure 7, the selection policy would use another variable (NewArea) that indicates whether the mobile user has moved to a new area. This policy would be:

- NewArea $\rightarrow$ use plug-in UpdateDiffArea (Figure 8b)
- !NewArea $\rightarrow$ use plug-in UpdateSameArea (Figure 8a)

Individual scenarios can be highlighted in a UCM tool, or they can be transformed into a different formalism. Figure 11 shows the result of one potential transformation of two scenario definitions to Message Sequence Charts. The first MSC corresponds to the successful scenario from Figure 10a and the second MSC to Figure 10c. Since UCMs do not contain any information relative to the message exchanges required to implement causal relationships across components, synthetic messages (m0, m1, m2, ...) have to be inserted. These abstract messages could be refined into more detailed and realistic protocol exchanges given the necessary information. As expected in Figure 4a, placing services in the MobSC minimizes the number of messages at the expense of additional load on the MobSC. These two MSCs provide further insight regarding the real impact of such solution.

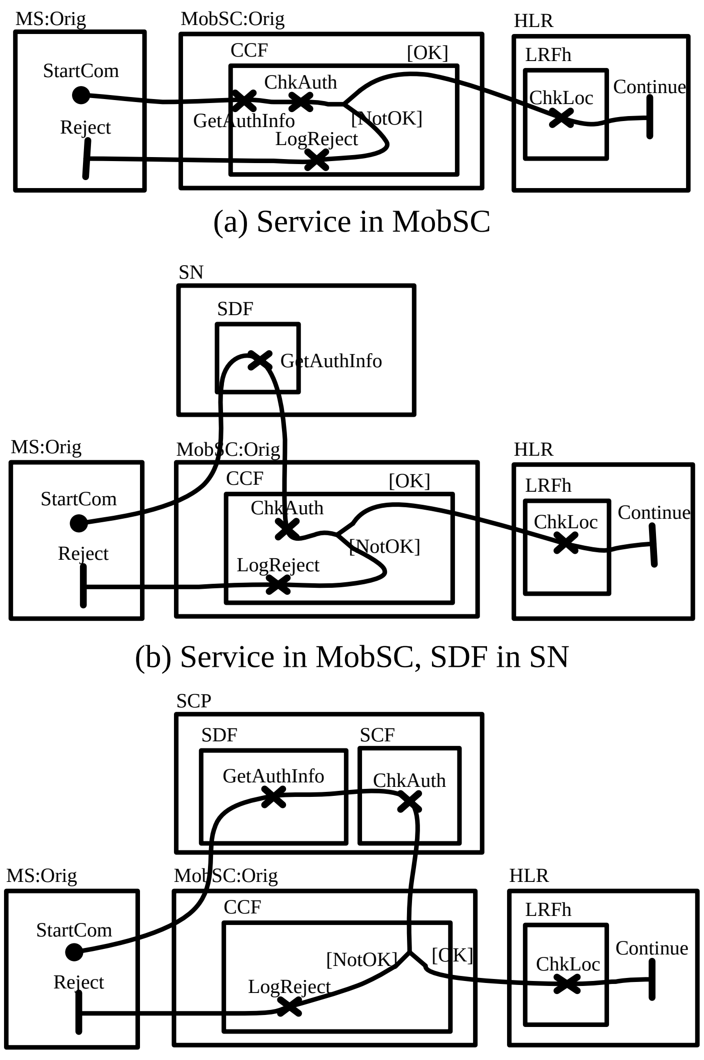
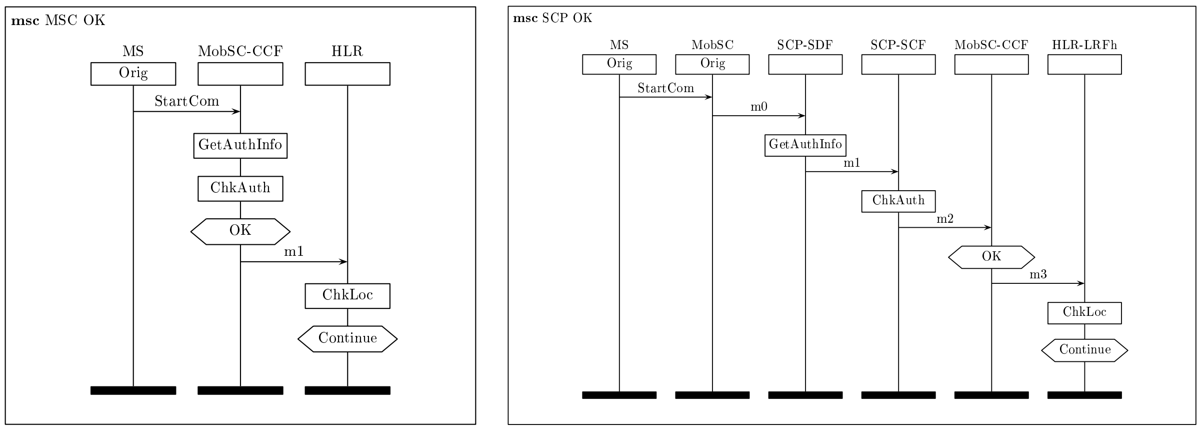


Fig. 10. Allocation of UCM Responsibilities to Alternative Component Structures

The path traversal mechanism permits us to find whether undesirable interactions, which often result from the composition of multiple plug-ins or complex conditions, can happen in a given context. It can report incomplete or ambiguous conditions (e.g. plug-ins with overlapping preconditions) while attempting to traverse them. Transformations to MSCs also help visualizing and understanding long scenarios that traverse multiple plug-in maps.

4.4. Advanced UCM Notation and Tool Support

In addition to the elements summarized in Figure A.2, the UCM notation includes symbols for various types of components (processes, objects, agents, and interrupt service requests) as well as component attributes (e.g. replicated, protected, formal, and anchored). Path elements and components can also be annotated with information used to generate performance models in the form



(a) MSC for Service in MobSC   b) MSC for Service in SCP

Fig. 11. Generation of MSCs from UCM Scenario Definitions

of *Layered Queueing Networks* (LQN). Such LQN models can be used to provide a quantitative analysis of alternative UCM architectures [25]. UCMs also support the concept of dynamic components, where the component structure can evolve at run-time. Components can be created, deleted, moved along UCM paths, stored in pools, and put in slots to assume particular roles. Dynamic responsibilities enable the manipulation of dynamic components, and together they support the description of dynamic systems using a static representation. Additional information on these concepts can be found in [9,10,18].

UCMNav is a free graphical tool for the edition and exploration of UCMs [21,28]. The edition is transformation-based and ensures that the UCMs drawn respect the syntax and static semantics of the language. The tool supports the whole notation and maintains many types of bindings (responsibilities to components, plug-ins to stubs, child components to parent components, performance annotations, etc.). The file format is in line with the XML concrete syntax proposed in [18] and is multi-platform, like the tool itself (Windows, Linux, HP/UX, and Solaris). The UCMs can be exported to various graphical formats (EPS, CGM, SVG, and MIF), and many types of reports can be generated (PostScript/PDF). UCMNav version 2.1 now supports scenario definitions, scenario highlights (traversed paths are colored), and the generation of MSCs in Z.120 format [22], of XML scenarios, and of Layered Queuing Networks [25].

5. Methodology Elements

The upcoming User Requirements Notation standard will not impose any development process. However, several methodology elements will be proposed in a companion document (Z.153), and some of the most typical ones are discussed in this section.

The first problem requirements engineers and designers are faced with when using URN is where to start. For new systems, one can start by identifying and analyzing system objectives and tentative corse-grained requirements (functional and non-functional). Evaluations of GRL models help making and justifying high-impact decisions very early in the development cycle. The various GRL tasks that are refined during this process can be connected to UCMs in a number of ways (e.g. task to UCM, task to UCM element, task to scenario definition, etc.). A UCM could also elaborate a collection of tasks (for instance, to explore ordering among these tasks) or be connected directly to goals or softgoals in the GRL model. Such traceability relationships are important, especially during the evolution of the system where they can be used for impact analysis. Alternatively, for legacy systems and for cases where stakeholders have difficulties expressing goals and requirements in an abstract way, UCMs can be used as a start point for the discovery of requirements. Individual UCMs can be gradually integrated together and structured using stubs and plug-ins. Both approaches should be iterative and incremental.

If the underlying structure of components is unknown, UCM scenarios can be documented without referencing it. Boolean conditions and scenarios can then be introduced, and validation through exploration and scenario highlight can then be used. The various alternatives identified in the GRL model contribute to the qualitative selection of a suitable structure of components. When necessary, quantitative analysis based on UCM performance annotations and the generation of LQN models can be done. Once the component structure becomes more stable, MSC scenarios can be generated to pave the way towards more detailed designs based on the UCM requirements. Functional test cases (e.g. in TTCN) can likely be created from UCMs in a similar way.

Checking traceability links between GRL and UCM models can also be beneficial. A goal not covered by any scenario is a symptom of an incorrect or overspecified GRL model or of an incomplete UCM model. Similarly, a scenario that does not contribute to any goal is either not necessary, or the goal model needs to be enhanced. The contribution levels in a GRL model can also indicate which scenarios will have a high impact on the system objectives, and hence these levels can help prioritize requirements and scenarios.

Many of the above methodology elements have been illustrated in this paper. The high-level relationships that exist between GRL, UCM, other ITU-T languages, and UML are summarized in Figure 12. URN represents a missing piece of the modelling puzzle that connects informal requirements and use cases to structural and behavioral models.

Many of the connections between UCM/GRL and the other pieces of this puzzle have been explored in the literature. Liu and Yu experimented the combined use of GRL and UCM for architectural design, and compared a URN-based approach with other related architectural approaches [20]. A related proposition where UCM scenarios support the selection of appropriate architectures is described by de Bruin and van Vliet [8]. Sales presented an approach that bridges informal requirements and formal specifications using UCMs and SDL [14] which has been successfully applied to the description and analysis of IETF's Open Shortest Path First routing protocol [26]. Petriu and Woodside did many successful experiments involving the generation and analysis of LQN-based performance models from UCMs [25]. Amyot and Mussbacher have explored the relationships that exist between UCMs and UML activity diagrams [5]. Andrade has developed a pattern language for mobile wireless systems using UCMs [6,7], while Mussbacher and Amyot provided patterns of UCM styles for describing complex systems [23]. Guan recently implemented a tool that synthesizes LOTOS specifications from UCMs [12], following previous work by Amyot and Logrippo in the same area [1,4].

The development of UCM-to-MSC generators, initiated by Miga *et al.* in [22], is still ongoing. A recent addition to the UCMNav tool decouples the traversal of UCMs (whose results are stored in an XML file) from the generation of a target scenario model such as MSC. This separation of concerns contributed to the resolution of many problems in the previous mechanism (which was attempting to do too many things at the same time), and leads to more flexible generation of MSCs (e.g. through XSLT transformations). We expect the same mechanism to be reusable for the generation of UML sequence diagrams and of TTCN test goals. Additionally, we started experimenting with the generation of SDL models from UCMs, for early validation purposes. Instead of doing a direct UCM-to-SDL transformation, we are studying a UCM-MSC-SDL synthesis process that combines the MSC generation capabilities of UCMNav with the MSC-to-SDL synthesis supported by commercial tools.

Relations between URN and ASN.1, and between URN and eODL, remain to be explored. URN has little need for data, except in th UCM path data model, which could be aligned with (subset of)

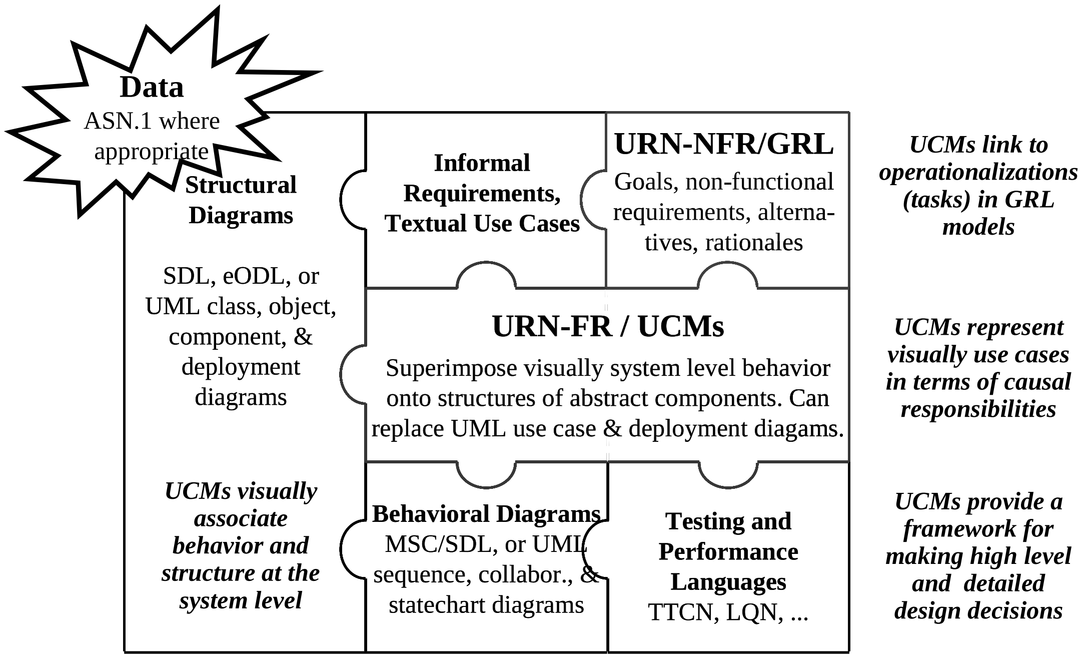

Data
ASN.1 where appropriate
Structural Diagrams
SDL, eODL, or UML class, object, component, & deployment diagrams
UCMs visually associate behavior and structure at the system level
Informal Requirements, Textual Use Cases
URN-NFR/GRL
Goals, non-functional requirements, alternatives, rationales
URN-FR / UCMs
Superimpose visually system level behavior onto structures of abstract components. Can replace UML use case & deployment diagams.
Behavioral Diagrams
MSC/SDL, or UML sequence, collabor., & statechart diagrams
Testing and Performance Languages
TTCN, LQN, ...
UCMs link to operationalizations (tasks) in GRL models
UCMs represent visually use cases in terms of causal responsibilities
UCMs provide a framework for making high level and detailed design decisions

Fig. 12. GRL, UCM, and Other Pieces of the Modelling Puzzle

ASN.1. UCM components could also the bridging point between URN scenarios and eODL concepts.

6. Conclusions

This paper gave an overview of the core URN concepts and notation elements, illustrated with examples from the wireless telephony domain. URN supports the discovery and analysis of requirements at multiple levels. It targets early phases of the development of complex reactive, distributed and dynamic systems, but is also applicable to many other domains, both in industrial setting and in standardization organizations. URN helps bridging the gap between informal and formal concepts, and between requirements models and design models. URN combines two complementary notations, both with a history of applications to requirements engineering activities.

The Goal-oriented Requirement Language focuses on incomplete, tentative, requirements (especially non-functional ones). GRL models describe business and system goals, alternatives, and rationales that are refined into requirements. Evaluations are used to determine whether goals are satisfied by a given solution. Evaluations of alternatives can be achieved in a qualitative way using initial levels of satisfaction and propagation rules.

Use Case Maps are most useful for specifying operational scenarios and functional requirements. UCM scenarios can be independent from underlying components, or they can be allocated to them in order to evaluate alternative architectures. Scenario definitions and path traversals enable the analysis of complex UCMs and the generation of more detailed representations, including MSCs. Transformations to performance models support early performance engineering activities, at the requirements level.

Together, GRL and UCM cover most of the language objectives for URN identified in Section 1. Using URN, even in a lightweight and informal way, can bring major benefits for little modelling investment. URN also fills a void in the ITU-T family of languages and has many potential connections to these languages and to UML. These relationships are expected to get tighter as UML profiles are created for the ITU-T languages. Such a profile for URN (Z.159) is planned for 2004.

The author is indebted towards Rossana Andrade, Gunter Mussbacher, and the members of the URN Focus Group for all their contributions. A special thank goes to Ognian Kabranov, Misbah Islam, Rossana Andrade, Don Cameron, and anonymous reviewers for comments on an earlier version of this paper. This work was supported financially by Nortel Networks, NSERC, and the University of Ottawa, Canada.

References

[1] D. Amyot, *Specification and Validation of Telecommunications Systems with Use Case Maps and LOTOS*. Ph.D. thesis, SITE, Univ. of Ottawa, Canada, 2002.

[2] D. Amyot and R. Andrade, Description of Wireless Intelligent Network Services with Use Case Maps. *SBRC'99, 17° Simpósio Brasileiro de Redes de Computadores*, Salvador, Brazil (May 1999) 418–433.

[3] D. Amyot and A. Eberlein, An Evaluation of Scenario Notations and Construction Approaches for Telecommunication Systems Development. To appear in: *Telecommunication Systems Journal*, 2003.

[4] D. Amyot and L. Logrippo, Use Case Maps and LOTOS for the Prototyping and Validation of a Mobile Group Call System. *Computer Communication*, 23(12) (May 2000) 1135–1157.

[5] D. Amyot and G. Mussbacher, On the Extension of UML with Use Case Maps Concepts. *<<UML>>2000, 3rd International Conference on the Unified Modeling Language*, York, UK (October 2000), LNCS 1939, 16–31.

[6] R. Andrade, *Capture, Reuse, and Validation of Requirements and Analysis Patterns for Mobile Systems*. Ph.D. thesis, SITE, Univ. of Ottawa, Canada, May 2001.

[7] R. Andrade and L. Logrippo, Reusability at the Early Development Stages of the Mobile Wireless Communication Systems. *Proceedings of the 4th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2000)*, Vol. VII, Orlando, Florida, 2000, 11–16.

[8] H. de Bruin and H. van Vliet, Scenario-Based Generation and Evaluation of Software Architectures. *Generative and Component-Based Software Engineering (GCSE'01)*, LNCS 2186, 2001.

[9] R.J.A. Buhr and R.S. Casselman, *Use Case Maps for Object-Oriented Systems*, Prentice-Hall, 1996.

[10] R.J.A. Buhr, Use Case Maps as Architectural Entities for Complex Systems. *IEEE Transactions on Software Engineering*, 24(12) (Dec. 1998) 1131–1155.

[11] L. Chung, B.A. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishers, 2000.

[12] R. Guan, *From Requirements to Scenarios through Specifications: A Translation Procedure from Use Case Maps to LOTOS*. M.Sc. thesis, Univ. of Ottawa, Canada, September 2002.

[13] ITU-T, *Recommendation Q.65, The unified functional methodology for the characterization of services and network capabilities including alternative object-oriented techniques*. Geneva, 2000.

[14] ITU-T, *Recommendation Z.100, Specification and Description Language (SDL)*. Geneva, 2000.

[15] ITU-T, *Recommendation Z.120, Message Sequence Chart (MSC)*. Geneva, 2001.

[16] ITU-T, URN Focus Group, *Draft Rec. Z.150 - User Requirements Notation (URN)*. Geneva, November 2002. `http://www.UseCaseMaps.org/urn/`

[17] ITU-T, URN Focus Group, *Draft Rec. Z.151 - Goal-oriented Requirement Language (GRL)*. Geneva, February 2002.

[18] ITU-T, URN Focus Group, *Draft Rec. Z.152 - UCM: Use Case Map Notation (UCM)*. Geneva, February 2002.

[19] L. Liu et al., *GRL and OME* (since 2001). `http://www.cs.toronto.edu/km/GRL/`

[20] L. Liu and E. Yu, From Requirements to Architectural Design - Using Goals and Scenarios. *From Software Requirements to Architectures Workshop (STRAW 2001)*, Toronto, Canada, May 2001.

[21] A. Miga, *Application of Use Case Maps to System Design with Tool Support*. M.Eng. thesis, Dept. of Systems and Computer Engineering, Carleton Univ., Canada, October 1998.

[22] A. Miga, D. Amyot, F. Bordeleau, D. Cameron, and M. Woodside, Deriving Message Sequence Charts from Use Case Maps Scenario Specifications. *Tenth SDL Forum (SDL'01)*, Copenhagen, Denmark, 2001.

[23] G. Mussbacher and D. Amyot, A Collection of Patterns for Use Case Maps. *First Latin American Conference on Pattern Languages of Programming (SugarLoafPLoP 2001)*, Rio de Janeiro, Brazil, 2001.

[24] OMG, *Unified Modeling Language Specification*, Version 1.4, May 2001.

[25] D. Petriu and M. Woodside, Software Performance Models from System Scenarios in Use Case Maps. *12th Int. Conf. on Modelling Tools and Techniques for Computer and Communication System Performance Evaluation*, London, U.K., April 2002.

[26] I. Sales, *A Bridging Methodology for Internet Protocols Standards Development.* M.Sc. thesis, SITE, Univ. of Ottawa, Canada, August 2001.

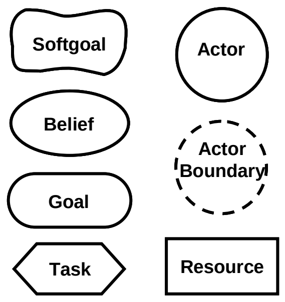
[27] Telecommunication Industry Association,*TIA / EIA/ IS-771: Wireless Intelligent Network*, July 1999.

[28] *Use Case Maps Web Page* (since March 1999) `http://www.UseCaseMaps.org`

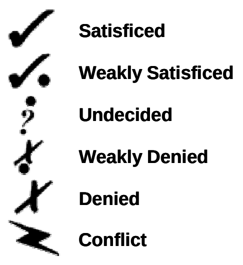
[29] E. Yu and J. Mylopoulos, Why Goal-Oriented Requirements Engineering. *Proceedings of the 4th REFSQ*, Pisa, Italy, 1998, 15–22.

A. Summary of the Main URN Notation Elements

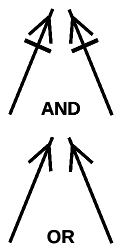
B. Abbreviations Used in the UCM Examples



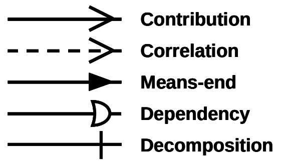
(a) GRL Elements



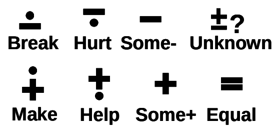
(b) GRL Satisfaction Levels



(c) Link Composition

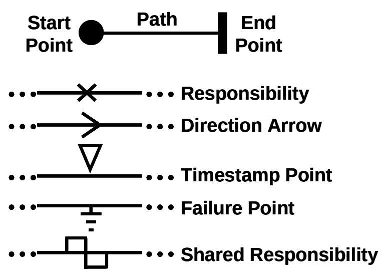


(d) GRL Links

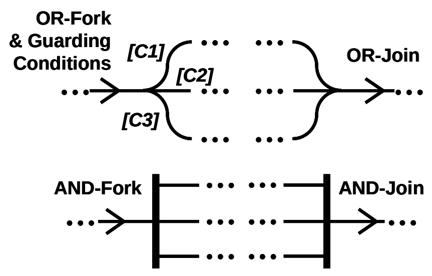


(e) GRL Contributions Types

Fig. A.1. Summary of the GRL Notation

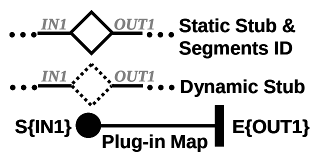


(a) UCM Path Elements

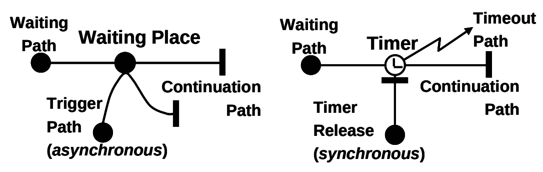


(b) UCM Forks and Joins

(c) UCM (Generic) Component



(d) UCM Stubs and Plug-ins



(e) UCM Waiting Places and Timers

Fig. A.2. Summary of (a Subset of) the UCM Notation

UCM	Label	Description
Figure 6	Continue	Scenario can continue
	FailHO	Failed handoff
	HOreq	Handoff request
	TuneNewChan	Tunes to a new wireless channel
	TunePrevChan	Tunes to the previous wireless channel
	UpdProfile	Updates the user profile
	[better]	The new channel is better than the previous one
	[NotOK]	The previous channel no longer available
	[OK]	The previous channel is still available
	[worse]	The new channel is worse than the previous one
Figure 7	Answer	The communication request was answered
	CellCheck	Checks the need for a cell handoff
	ChkAuth	Checks (verifies) the authorization
	ChkLoc	Checks (determines) the user location
	Disconnect	The user has disconnected
	EndCom	The communication ended
	GetAuthInfo	Gets the authorization information
	GoHO	Starts the handoff procedure
	LogReject	Adds the rejected attempt to the log
	Reject	The attempt was rejected
	StartCom	Initiates a communication attempt
	StopHO	Stops the handoff procedure
	UpdHLoc	Updates the user's home location
	[NotOK]	The user is not authorized
	[OK]	The user is authorized
Figure 8a	ProfUpdated	The user profile was updated
	UpdSameArea	Starts the profile update in the same area
	UpdVisProfile	Updates the profile of the visiting user
	[HomeArea]	The user is in his/her home area
	[VisArea]	The user is in a visiting area
Figure 8b	CreateVisProfile	Creates a new visiting profile for the visiting user
	DelVisProfile	Deletes the profile of the visiting user
	ProfUpdated	The user profile was updated
	UpdDiffArea	Starts the profile update in a different area
	[NewAreaHome]	The new area is the user's home area
	[NewAreaVis]	The new area is a visiting area
	[OldAreaHome]	The previous area was the user's home area
	[OldAreaVis]	The previous area was a visiting area
Figure 9	Answer	The communication request was answered
	Disconnect	The user has disconnected
	EndCM	The communication management has ended
	GetRes	Gets the necessary resources
	RelRes	Releases the allocated resources
	StartCM	Starts the communication management procedure
	Talking	The users can talk to each other
	WaitConn	Waits for a connection to be established
	WaitDesc	Waits for a disconnection
	[NotAllResAvail]	The requested resources are not all available
	[NotTalking]	The users are not talking to each other
	[ResAvail]	All the requested resources are available
	[Talking]	The users are talking to each other

Fig. B.1. Abbreviations Used in Figures 6-9