URN: Towards a New Standard for the Visual Description of Requirements

Daniel Amyot[1] and Gunter Mussbacher

[1] SITE, University of Ottawa
800 King Edward
Ottawa, Ontario, Canada, K1N 6N5
damyot@site.uottawa.ca

Abstract. In November 1999, the International Telecommunication Union (ITU-T, SG 17) initiated a question on the standardization of a User Requirements Notation (URN) for complex reactive, distributed, and dynamic systems and applications. URN is intended to be standardized by September 2003. This paper presents the motivations behind URN, its objectives, and the current proposal that combines two complementary languages. The first one, GRL (Goal-oriented Requirement Language), is used to describe business goals, non-functional requirements, alternatives, and rationales. The second one, UCM (Use Case Maps), enables the description of functional requirements as causal scenarios. The introduction of URN is likely to impact the development ans use of other SG 17 languages (especially MSC and SDL) as well as OMG's UML. This paper briefly explores several relations between these languages as well as potential for synergy and coordination.

1 Introduction

Requirements engineering has become an essential part of development approaches to systems, applications, and protocols in general. However, few standardized notations and techniques can address the needs specific to visualizing and analyzing functional and non-functional requirements (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 research in the standardization of visual notations used to describe requirements for complex dynamic systems [19]. Such systems, of various sizes and natures, include wireless or IP-based telecommunication systems, e-commerce and Web applications, and other types of distributed, embedded, or reactive systems.

The creation of a standard such as URN is not without challenges. Among others, three important aspects of future requirements engineering techniques will be *a*) the ability to capture goals and decision rationale which shape the resulting system, *b*) the ability to seamlessly move from analysis models to design models, and *c*) the ability to model dynamic systems where behaviour and structures may change at run-time. The first aspect, universally applicable, will allow tentative, ill-defined, and ambigu-

ous requirements to be expressed and clarified, thus reducing development costs caused by requirements problems and discussions that are repeated over and over again. The second aspect, important to a wide range of problem domains, will allow front-end software development activities to be tied more closely to the subsequent activities. This promises more complete life cycle round-trip engineering, a reduction of development time, and an increase in the software quality. Telecommunication systems based on recent and upcoming technologies such as agents, mobility, and IP are representative of dynamic systems as mentioned by the third aspect. These systems raise new modelling issues because of complex and sometimes unpredictable policy-driven negotiations between communicating entities and lead to protocols and entities more dynamic in nature and evolving at run time.

In its current form, URN attempts to answer these challenges by combining two complementary languages. The *Goal-oriented Requirement Language* (GRL) addresses the first aspect (*a*). Goal-oriented modelling has been proposed in the requirements engineering community for a number of years and several approaches have been published [24][35]. GRL is a rather new addition to this growing list of techniques built on the well-established NFR (*Non-Functional Requirements*) framework [10]. GRL captures business or system goals, alternative means of achieving goals (either objectively or subjectively), and the rationale for goals and alternatives. The notation is applicable to non-functional as well as functional requirements.

Use Case Maps (UCMs) are a scenario-based notation and software engineering technique that address the latter two aspects (*b* and *c*) [8]. UCMs have a history of application to the description of object-oriented systems and reactive systems in various areas, including software architecture, telephony, wireless, mobile, and agent domains [2][4][5][7][28]. UCMs are used as a visual notation for describing causal relationships between responsibilities of one or more use cases. UCMs are most useful at the early stages of software development and 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 structure and allow incremental development and integration of complex scenarios.

This paper describes the motivations behind URN, its objectives, and an overview of the two complementary notations. In addition to illustrative examples on GRL and UCMs, this paper discusses the role of URN in development methodologies [12][25] and its relationships to existing notations from OMG (UML) [29], ITU-T (MSC [17], SDL [16], TTCN [18], etc.), and performance engineering (Layered Queuing Networks [33]). Several remaining challenges and work items for URN are also presented. This paper is based on contributions to the standardization of URN and industrial experience gained with URN by the authors.

2 Objectives and Structure of the URN Standard

As suggested by the I.130 [14], Q.65 [15], and SDL+ methodologies [16], as well as by several UML-based approaches, the specification/design of distributed systems can

often be decomposed into three major stages. At Stage 1, services are described from the user's point of view in prose form, and sometimes with use cases, tables, and informal diagrams. The focus of the second stage is on control flows between the different entities involved, represented using Message Sequence Charts or UML sequence diagrams. Finally, Stage 3 aims to provide specifications of component behaviour, protocols and procedures, sometimes using (formal) languages such as SDL or UML Statecharts.

Acknowledging the need to improve this type of approach to properly answer the challenges raised by the modelling of dynamic, evolving complex systems, ITU-T Study Group 17 (which results from the merger of the former SG 10 and SG 7) approved a question for study on URN (Q.18/17) in September 2000. URN intends to fill a void in the ITU-T family of languages by allowing the visual description and analysis of requirements, both in standardization bodies and in industry. Although its prime application domain remains telecommunication services, URN also aims to cover a wide range of reactive, distributed, and dynamic systems beyond this domain.

2.1 URN Objectives

These objectives were part of the initial question for study (Q10/12). Since scenarios are well understood and already used by many stakeholders, these objectives focus on scenarios as a means to express functional requirements and to enable early analysis. URN is hence meant to have the following capabilities:

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

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

c) 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;

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

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

f) facilitate the detection and avoidance of undesirable interactions between features (or services);

g) provide insights at the requirements level to enable designers to reason about feature interactions and performance trade-offs early in the design process.

To address the void in dealing with business goals and non-functional requirements, additional objectives were proposed later by the URN Focus Group:

h) provide facilities to express, analyze and deal with non-functional requirements;

i) provide facilities to express the relationship between business objectives and goals to system requirements expressed as scenarios and global constraints over the system, its development, deployment, maintenance and evolution and operational processes;

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

k) Support traceability and transformations to other languages of the ITU-T (including UML).

All these objectives are refined as more precise and detailed requirements in the URN draft proposal [19].

2.2 Structure of the URN Family of Standards

The URN Focus Group already produced several draft versions of the URN documents, structured as follows:

- **Z.150 – URN** [19]: Provides motivations, scope, and objectives for URN, as well as basic terminology and basic requirements engineering concepts. Many specific requirements also refine the list of URN objectives presented in section 2.1. These fall into one of the three following categories: URN-NFR (URN - Non-Functional Requirements), URN-FR (URN - Functional Requirements), and others (relationship between URN-NFR and URN-FR, traceability, testing, performance analysis, round-trip engineering, etc.).
- **Z.151 – GRL** [20]: This document proposes GRL as a notation for URN-NFR. Notation constructs are defined, together with their visual representation and concrete grammars (textual and XML). A conformance table (GRL to URN-NFR) is provided, and a mini-tutorial and references are included.
- **Z.152 – UCM** [21]: This document proposes UCM as a notation for URN-FR. Again, visual and concrete XML representations are defined, together with a conformance table, a mini-tutorial, and references.

A fourth document (Z.153), which is not yet available, will focus on relationships between GRL and UCM, and between URN and other languages (from ITU-T, UML, Layered Queuing Networks, etc.). Z.153 addresses the last URN objective cited in the previous section.

3 GRL: Goal-oriented Requirements Language

The GRL graphical language is used to support goal and agent-oriented modelling and reasoning, providing guidance to the design process. Goal-oriented modelling has been proposed in the requirements engineering community for a number of years and several approaches have been published [22][35]. GRL is a rather new addition to this growing list of techniques and builds on the well-established *NFR framework* [10] (used for modelling non-functional requirements) and the agent-oriented language *i** [34] (used for the modelling, analysis, and reengineering of organizations and business processes).

Often, requirements from various stakeholders are initially provided as objectives or desired goals. Directly expressing such goals, rather than activities and entities that would refine these goals, enables designers to reason about various alternatives while avoiding early commitments to a particular solution. Such goals are also more stable and long-lived than small-grain activities during system evolution. Goal-oriented modelling hence allows us to handle and trace non-functional requirements and stra-

tegic objectives while discovering functional requirements before they become operational (i.e. with tasks and scenarios). The incorporation of explicit goal representations also provides a criterion for requirements completeness as requirements can be considered complete if they are sufficient to establish the goals they are refining.

GRL addresses most of URN's additional objectives described in section 2.1 (h, i and j). At the core of GRL we find intentional elements, actors, and their relations. Intentional elements model the "why" of certain requirements (objectives, alternatives, rationales), not operational details. GRL can however be connected to scenario notations and hence enables one to reason about operational aspects.

3.1 Concepts and Notation

GRL supports four main categories of concepts, namely intentional elements, intentional relations, actors, and non-intentional elements. The first category contains five basic concepts:

- **Goal**: Quantifiable high-level (functional) requirement (illustrated as a rounded-cornered rectangle).
- **Softgoal**: Qualifiable but unquantifiable requirement, essentially non-functional (illustrated as a cloud).
- **Task**: Operationalized solution that achieves a goal, or that *satisfices* a softgoal which can never be fully achieved due to its fuzzy nature (illustrated as a hexagon).
- **Resource**: Entity whose importance is described in terms of its availability (illustrated as a rectangle).
- **Belief**: Rationale or argumentation associated to a contribution or a relation (illustrated as an ellipse).

There are also five categories of intentional relations, which connect elements:

- **Contribution**: Describes how softgoals, tasks, beliefs, and relations contribute to each other. Each contribution can be qualified by a degree: equal, break, hurt, some-, undetermined, some+, help, or make (see Fig. 1).
- **Correlation**: Contribution that indicates side-effects on other intentional elements (dashed line).
- **Means-end**: Link for tasks achieving goals. Different alternatives are allowed.
- **Decomposition**: Defines what is needed for a task to be performed (refinement), always AND.
- **Dependency**: Link between two actors depending on each other (half-circle).

Fig. 1 illustrates some of these concepts (in italic characters) with a partial example based on the access of a distributed accounting application. The emphasis is put on system security, which can be qualified but not precisely quantified. This softgoal is decomposed in terms of security of the terminal and of the host, both being required but with different contributions. The security of the terminal can be satisficed in two ways, using encryption or access authorization. This last option is further decomposed into two quantifiable goals, including the authentication which can be achieved in three ways (cardkey, password, or biometrics). Selecting one of these alternatives over the others will lead to different side-effects on the cost of the terminal. The

password solution has a slight positive impact whereas the use of biometrics has a negative impact because such technology would need to be developed in-house.

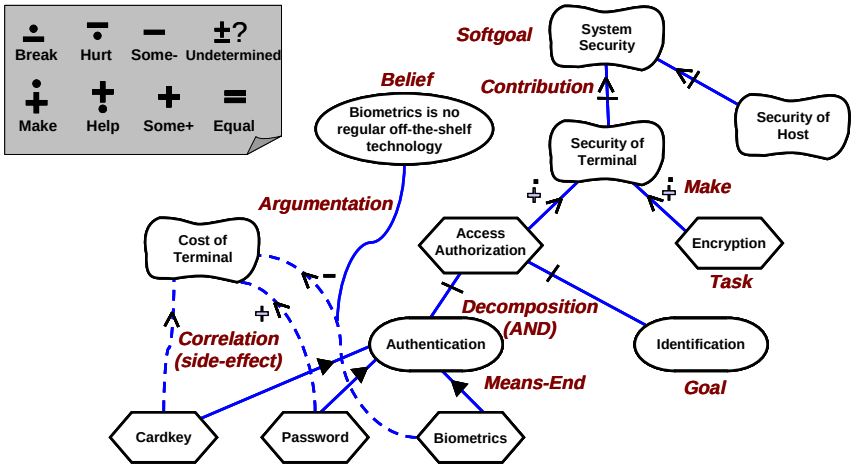


Fig. 1 Example of GRL model with relations and intentional elements

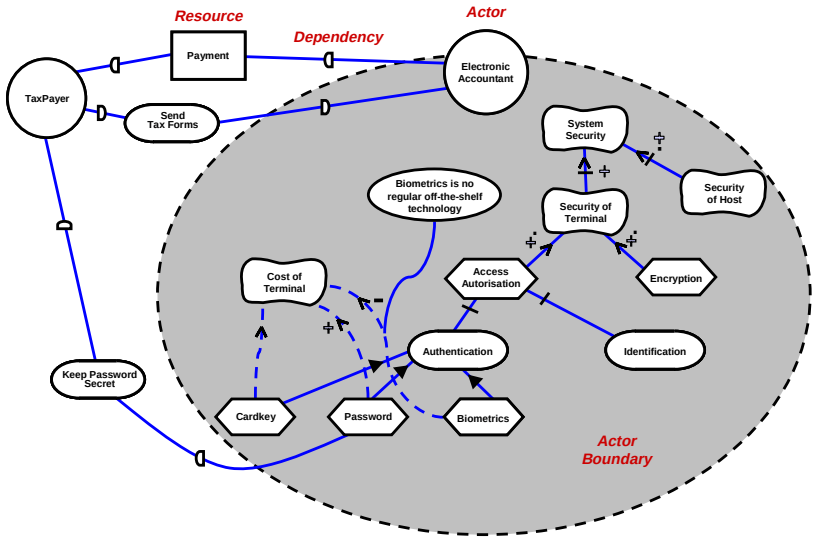


Fig. 2 Example of GRL model with actors

The last two categories of concepts include actors and non-intentional elements:

– **Actor**: Active entity with intentions that executes actions to achieve its goals (shown as a circle). Actors can be used to do role-based analysis on social relation-

ships (dependencies). A goal graph can be associated to an actor by circling the graph with a dashed line.
- **Non-Intentional Element**: Reference to a model external to GRL (3-D rectangle, not shown here).

In Fig. 2, the GRL model of Fig. 1 is associated to an electronic accountant actor, which depends on a tax payer actor for payments (resource). On his side, the tax payer depends on the electronic accountant system to process the tax forms. The password must also be kept secret by the tax payer. This dependency does not exist for the other two solutions (cardkey and biometrics).

3.2 Evaluation

The two previous GRL models illustrate the usefulness of GRL for visualizing static relations existing between the various goals, the alternatives meant 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. Such mechanism requires one to assign a qualitative degree of satisfaction or availability (see legend of Fig. 3) to tasks and goals in the graph (usually some of the leaves), and then to use a propagation algorithm (based on [10]) to compute how well parent goals are satisfied. In some cases, designer input might be required to solve conflicting situations.

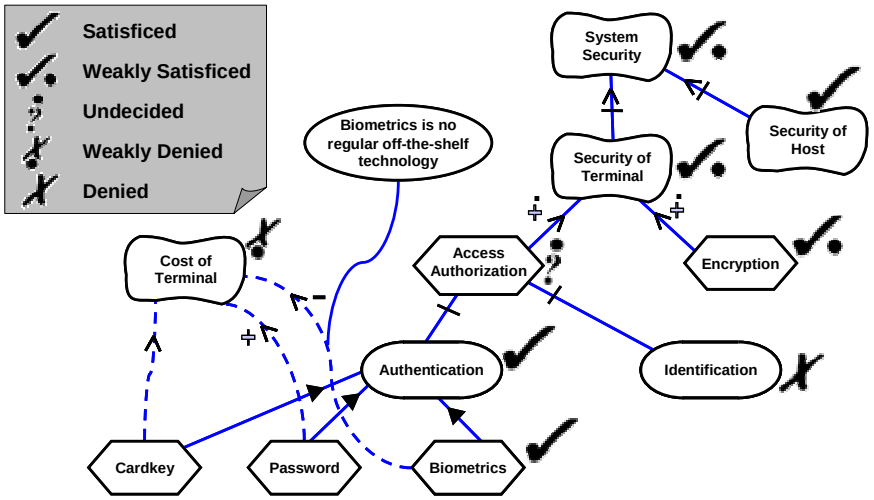


Fig. 3 Evaluation of candidate solution in a GRL model

Fig. 3 illustrates a candidate solution where biometrics is selected and the security of the host is initially satisficed. Also, the encryption is weakly satisficed and the identification is denied. The propagation to the higher-level goals goes as follows. Since the biometrics option is linked to the authentication through a means-end relation, the authentication will be achieved as well. If the identification is not possible, then one

can be undecided about whether the access authorization is satisfied or not. Note also that selecting biometrics will have a negative impact on the low cost of the terminal (because biometrics is an expensive authentication technology). The security of the terminal has a satisfaction level equivalent to the highest level amongst its candidate refinements, and hence becomes weakly satisficed. Finally, the system security depends on both the security of the host and that of the terminal, and hence is qualified as weakly satisficed.

Different alternative solutions can be quickly and systematically evaluated, which helps finding a global solution that maximizes the level of satisfaction of the highest-level goals, and hence leads to a good tradeoff between conflicting goals.

3.3 GRL Tool Support with OME

OME (*Organization Modelling Environment*) is a goal-oriented and agent-oriented modelling and analysis tool. It supports the GRL, NFR, and *i** notations [23]. OME offers a graphical environment for the creation, maintenance, and analysis of GRL models. Written in Java, OME now supports the export of GRL models in XML as well as the creation of catalogues containing instantiable GRL models, which increase the reuse of know-how and speed up the modelling process.

4 UCM: Use Case Map Notation

Modelling functional requirements of complex systems often implies an early emphasis on behavioural aspects such as interactions between the system and its environment (and users), on the cause to effect relationships among these interactions, and on intermediate activities performed by the system. Scenarios represent and excellent and usable way of describing these aspects.

Dozens of notations exist for scenario descriptions [13], but few can satisfy the objectives enumerated in section 2.1 (a to h). Most notations, similar to MSCs, require the presence of messages and components. Others do not have visual representations or do not support dynamic refinements well. A recent comparative study [1] indicates that the UCM notation and UML activity diagrams offer an abstraction level adequate for URN-FR because they both allow the description of related sets of scenarios without messages or even component structures. UCMs are however superior to activity diagrams in the context of URN because they support dynamic refinement at the behaviour and structure levels, they allow multiple start points in sub-maps, and they offer better support for scenario integration and for visualizing structures combined to behaviour [3]. Moreover, UCMs have a history of applications to the description of requirements for mobile, distributed, and agent systems, to the detection of undesirable feature interactions, to early performance analysis, to test generation, and to conversions to other types of models ([32] includes a comprehensive list of documents and references). More recently, the concept of *scenario definition* was also incorporated, and it will be further explored in this section.

4.1 Concepts and Notation

The UCM notation supports the modelling and analysis of systems described with scenarios, potentially combined to structural elements, at a high level of abstraction. UCMs have four basic concepts:

- **Start point**: Captures preconditions and triggering events (filled circle).
- **Responsibilities**: locations where computation (procedure, activity, function, etc.) is necessary (cross).
- **End point**: Represents resulting events and post-conditions (bar).
- **Paths**: Connects start points to end points and can link responsibilities in a causal way.

Alternative and concurrent paths that span an entire system may easily be captured. Operators for choice (**OR-fork**, **OR-join**) and parallelism (**AND-fork**, **AND-join**) are used to describe alternative paths (accompanied by guards, between square brackets), common segments, concurrent segments and their synchronization. Other basic elements indicate **waiting places**, with or without a **timer**. An event triggering the end of the waiting period can come from the environment or from another UCM scenario (through juxtaposition of an end point).

Optionally, scenario elements can be allocated to the components part of the system architecture and/or of its environment. A **component** represents an abstract entity (object, process, server, database, user, etc.). In general, a component is displayed as a rectangle containing scenario elements and other sub-components. The nature of components can vary (active/passive, composite or not, stacked, interrupt service, etc.), and different shapes and other graphical hints covered in [21] are then used.

Complex and lengthy scenarios can be decomposed and structured thanks to submaps called **plug-ins**, used (and reused) in map containers called **stubs** and displayed as diamonds. Most of the above concepts are illustrated in Fig. 4, which continues the GRL examples started before.

The UCM in Fig. 4a) represents a top-level scenario (commonly called *root map*), whereas the two other UCMs are plug-ins for the Authenticate stub. In this example, a tax payer wants to access the electronic accountant via the security system. After the tax payer identification (CheckID), the system needs to authenticate the requester, who can be accepted or rejected. The electronic accountant creates a new session in the first case and then becomes ready to process transactions (all of this in parallel with an acceptance notification).

Candidate alternatives for the authentication, considered in the GRL model, can now be operationalized to enable a more detailed analysis. For instance, the use of biometrics (Fig. 4b) or password (PW—Fig. 4c) can be described with plug-in maps. The stub input/output segments are then bound to the adequate start/end points of the plug-in. For biometrics, we have {<IN1, Bio>, <Out1, Yes>, <Out2, No>}. For the password solution, the binding relationship is {<IN1, PW>, <Out1, Yes>, <Out2, No>} (InputPW remains unbound).

Different structures, suggested by the alternatives identified in a GRL model, can also be quickly evaluated by moving responsibilities from one component to another, or by restructuring the components.

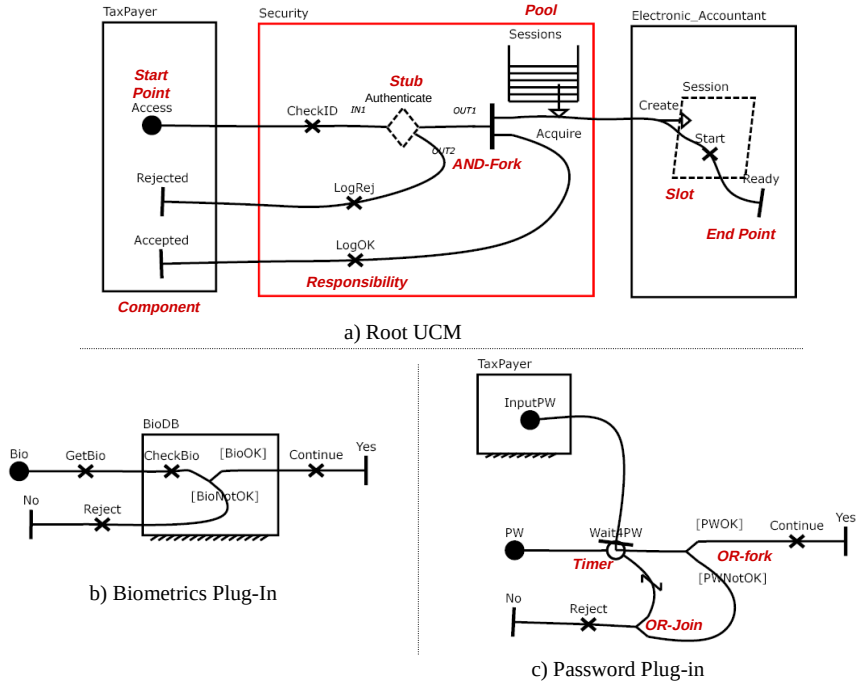


Fig. 4 Example of a root UCM and its two plug-ins

4.2 Dynamic Aspects

Dynamic responsibilities (on UCM paths) and dynamic components capture object and role dynamics in a static way. This capability is useful to describe dynamic systems (e.g. based on agents) while avoiding to have a series of snapshots exposing the system structure at different points in time. For example, Fig. 4 a) has two **dynamic responsibilities** shown as arrows. The first (Acquire) gets a **dynamic component** from a **pool** of sessions, and the second (Create) moves this session component into a **slot**, where it can start running. The Start responsibility is then executed by the session component obtained from Security by Electronic_Accountant. These notation features (as well as many other variants) are fairly unique to UCMs as they enable the static description of dynamic situations where components are created, deleted, stored, duplicated, or moved along a scenario path.

Dynamic stubs (dashed diamonds) are used to describe situations where the behaviour itself is dynamic. Unlike static stubs, which contain only one plug-in, dynamic stubs can have multiple plug-ins, one of which is chosen at run-time according to a **selection policy**. Our example could be extended to include a situation where a password is sufficient during the day but biometrics readings are required at night in the absence of a guard next to the access terminal. Dynamic stubs enable the simple integration and management of multiple scenarios and services.

4.3 Scenario Definitions and Path Traversal

Recently added to the UCM language, the concept of **scenario definition** offers the possibility to describe and extract individual scenarios from a complex UCM model. These individual 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, test cases, etc.).

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. These conditions are expressed using a relatively abstract path data model, different from the application data model which is not captured by UCMs. The path data model is comprised of global Boolean variables used in guarding conditions, timers, and selection policies. They can also be modified inside responsibilities. An initial condition is described by assigning the value true (T) or false (F) to each relevant global variable.

A UCM design with variables can be traversed by scenario definitions. The draft Z.152 document [21] describes over 30 generic requirements applicable to all path traversal mechanisms, but no specific algorithms. These requirements, which cover most of the elements of the UCM notation, essentially provide a dynamic semantics for UCMs. Many algorithms can satisfy these requirements and yet produce different traversals. In particular, if a UCM with AND-forks and AND-joins is not well-nested, then breadth-first and depth-first traversals may produce different but valid traversals.

In our example, the path data model contains five variables: *BioOK* (valid biometrics), *PWOK* (valid password), *Day* (day access), *Guard* (a guard is next to the terminal), and *WaitPW_timeout* (WaitPW is not triggered on time). These variables are used in Fig. 4 as well as in the selection policy of stub Authenticate, expressed with these two rules: <Day $\rightarrow$ PWOK; $\neg$(Day $\vee$ Guard) $\rightarrow$ Biometrics>.

Here are three possible scenario definitions, to be expanded in the next section.

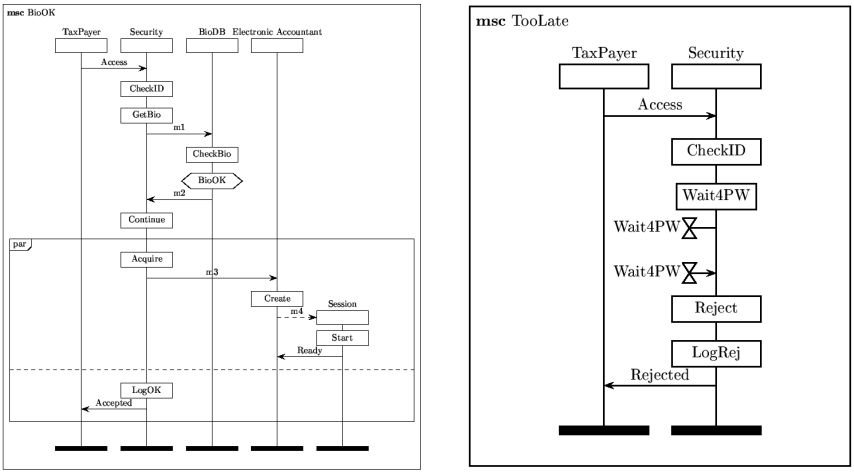
- Biometrics – Accepted: *BioOK* = T, *Day* = F, *Guard* = F. Start point: Access.
- Password – Too Late: *Day* = T, *WaitPW_timeout* = T. Start points: Access, InputPW.
- Night Guard: *Day* = F, *Guard* = T. Start point: Access.

4.4 Transformations and Validation

Use Case Maps provide an excellent source of information for guiding the generation of more detailed models. Various transformations towards LOTOS [2][4][5] and SDL [31] have been explored in order to validate requirements, generate designs, generate tests, detect undesirable scenario/feature interactions, and analyze performance.

Scenario definitions also enable transformations towards other languages. In particular, the generation of MSC scenarios help visualizing and analyzing long scenarios that traverse multiple maps (through stubs and plug-ins), and more generally to fill the gap between the two first development stages described in section 2 [26]. A path traversal algorithm combined to structure information (components) thus enables the automated generation of the MSCs found in Fig. 5 from the two first UCM scenario definitions described in section 4.3. Since UCMs do not contain any information relative to the message exchanges required to implement causal relationships across components, synthetic messages (*m1*, *m2*, *m3*…) are generated. Future transformations could take into consideration knowledge of desired standard protocols or mes-

sage patterns between pairs of components in order to generate more detailed and concrete MSCs.



a) Biometrics - Accepted

b) Password - TooLate

Fig. 5 Two MSC scenarios generated from the UCM example

Such MSCs help to quickly find whether undesirable interactions, which often result from the composition of multiple plug-ins or complex conditions, can happen in a given context. When conditions to be traversed are incomplete or ambiguous, the path traversal mechanism can return a warning. For instance, for the Night Guard scenario definition, the selection policy of stub Authenticate cannot determine which plug-in to select, because none of the preconditions can be satisfied.

4.5 Performance Modelling with UCM

Another usage of UCMs is for early performance modelling from requirements ans system-level scenarios. Various architectural configurations can be easily explored [7], and additional annotations provide the information necessary for the generation of performance models from UCM specifications.

GRL support performance attributes to the same extent it supports any other non-functional requirement, i.e. with textual attributes. This provides a good basis for traceability, but the annotations themselves have no particular semantics. UCMs however have placeholders for specific performance attributes (Fig. 6) used to generate performance models. For instance, an automated transformation to *Layered Queuing Networks* (LQN) is explored in [33]. Additionally, some of the annotations capture real performance requirements (e.g. response time between two timestamps, but not the resource model or the deployment information) which then become documented, analyzable up front, traceable to GRL models, and transferable to design models, e.g. in SDL. UCM performance annotations have also inspired many key concepts now found in the recent UML performance profile [30].

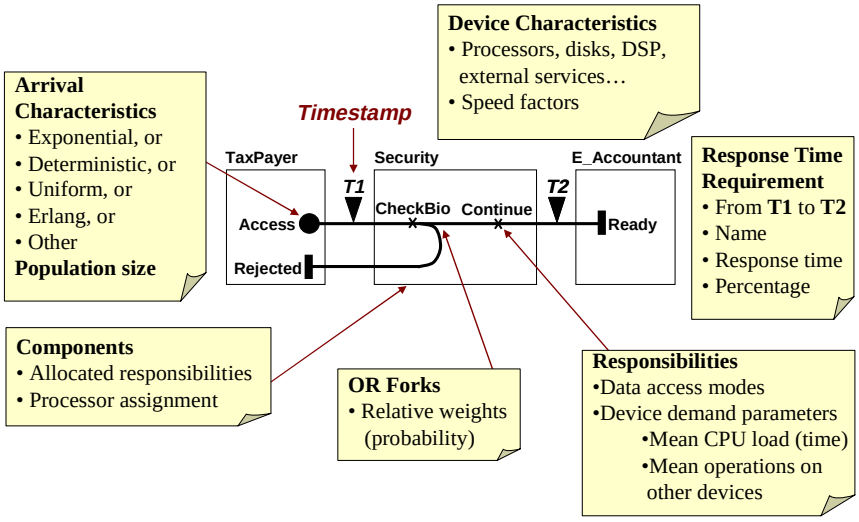


Fig. 6 UCM performance annotations

4.6 UCM Tool Support with UCMNAV

UCMNAV is a graphical tool for the edition and exploration of Use Case Maps [27][32]. 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 defined in [21] 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). The brand new UCMNAV 2.0 now supports scenario definitions, scenario highlights (traversed paths are colored), the generation of MSCs in textual form [17], and the generation of Layered Queuing Networks.

5 URN as a Missing Piece of the UML/SG17 Modelling Puzzle

Many interesting relationships exist between URN, UML, and other ITU-T languages. The main ones are summarized in Fig. 7 and will be further explored in the future Z.153 document.

GRL is a fairly unique piece of this puzzle and fills an urgent need in the modelling of goals, non-functional requirements, alternatives, and decision rationales. GRL focuses on the "why" aspects whereas scenarios describe the "what" of solutions. GRL goals and softgoals can be operationalized through tasks, and these tasks can be linked to UCM scenarios, hence describing the "how" of some requirements.

UCMs can express complex systems scenarios visually, even in the absence of components or messages. They offer capabilities more suitable for requirements-level

descriptions, analysis, and scenario integration than what is possible with UML use case diagrams and activity diagrams. The UCM notation also provides many necessary connections between scenarios and models for goals, structure, behaviour, performance analysis, and testing, hence improving its potential for traceability and transformations.

The modelling of goals and scenarios is an iterative process that may aid in identifying further goals and additional scenarios (and scenario fragments) important to system design, thus contributing to the completeness and accuracy of requirements, as well as to the quality of system design [24][25]. Combining the GRL and UCM notations makes it possible to evaluate technical solutions according to their contributions to the objectives of different stakeholders, therefore guiding the design towards viable solutions.

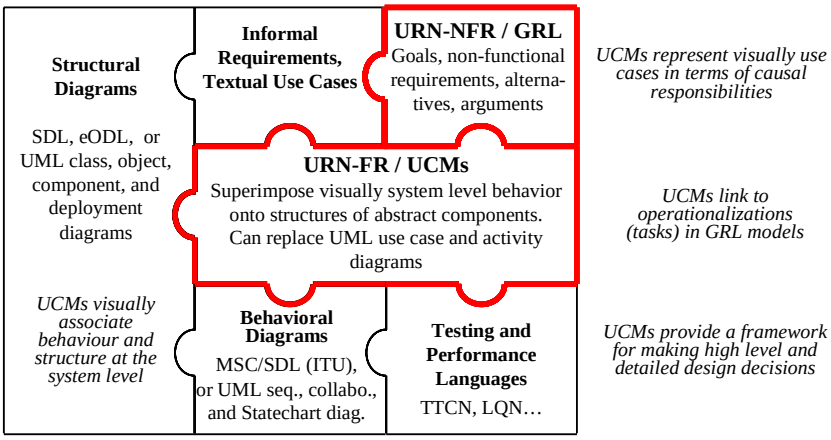


Fig. 7 URN as a missing piece of the modelling puzzle

GRL and UCMs have also demonstrated their usefulness in describing and evaluating design and requirements patterns [11], and this for various domains including wireless telephony [6] and service/scenario integration [28]. By taking advantage of a complementary usage of GRL and UCM, URN helps filling a notation and methodology void found in ITU-T languages and in UML, and also contributes to bridging the gap between models for requirements, analysis, and design.

6 Conclusions and Future Work

This paper presented the purpose, the objectives, and the structure of the future User Requirements Notation standard. The proposal currently studied combines GRL, for the description of business goals, non-functional requirements, alternatives, and rationales, with UCMs for the specification of functional requirements in the form of causal scenarios. The concepts and notation elements defining each of these complementary languages were introduced and illustrated with a short distributed application example (with a focus on security and access control). Multiple relations between these two languages as well as between URN and other related languages from ITU-T and OMG (including an overview of a transformation from UCM to MSC) have been

identified. Several benefits of the GRL/UCM marriage were also discussed along the way.

The URN framework in Z.150, still open for comments, is planned to be submitted for standardization in November 2002. Many challenges however remain for the other documents describing GRL (Z.151), UCMs (Z.152), and relationships to other languages (Z.153), all of which are planned to be submitted for standardization in September 2003.

The most important work item is to align the descriptions of the GRL and UCM languages according to the suggestions made so far in the SG 17 Language Coordination Project. Each of the SG 17 languages was developed in isolation over the years and has its unique way of formalizing abstract, concrete, and graphical grammars, meta-models, and interchange formats. Even data types are syntactically different, and often semantically incompatible. The UCM path data model, even if simple, would benefit from being compatible with data models of other languages, especially if it is extended to cover types other than Booleans (e.g. natural numbers and time). The usage of XML as an interchange format (that supports graphical layout) is currently only supported by the draft URN Recommendation but hopefully will spread to other ITU-T languages.

Several aspects of UCM path traversal requirements are still to be completed, including the definition of semantics for plug-in and component instantiations, and the definition of continuation criteria for a few remaining notation elements that are not covered yet. GRL propagation rules, used during the evaluation of models, need to be made explicit in the standard. Like for the UCM path traversal mechanism, no single algorithm will be provided, but general principles to which all evaluation algorithms should conform will be specified.

Another major work item is the content of Z.153, with a particular emphasis on the relationships between GRL and UCM, and on the conversion from UCMs to MSCs. Issues for connecting URN to ASN.1, TTCN, MSC, SDL, eODL, UML, X.911 (the ODP Enterprise Language) and non-standard performance modelling languages (e.g. LQN) need to be collected and addressed. There is also some overlap between the work on UCM performance annotations and that of time extensions for SDL (future Z.108), which may require additional coordination. There is a potential for transferring some requirements information (e.g. time response requirements) to SDL time constraints. A UML profile for URN could also be considered as a long-term goal, once UML 2.0 becomes more stable. Again, coordination with similar work on profiles for MSC and SDL will be required.

Acknowledgments

The authors are indebted towards the members of the URN Focus Group for all their contributions. A special thank goes to Don Cameron, the URN Rapporteur for the first two years, and to Andrew Miga for five years of UCMNAV development. We also acknowledge financial support from CITO, NSERG, Mitel Networks and Nortel Networks.

References

N.B. Many of these papers are available via http://www.UseCaseMaps.org/urn.

[1] Amyot, D. and Eberlein, A. (2002) "An Evaluation of Scenario Notations for Telecommunication Systems Development". To appear in: *Telecommunication Systems Journal*.

[2] Amyot, D. (2001) *Specification and Validation of Telecommunications Systems with Use Case Maps and LOTOS*. Ph.D. thesis, SITE, Univ. of Ottawa, Canada.

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

[4] Amyot, D., Buhr, R.J.A., Gray, T., and Logrippo, L. (1999) "Use Case Maps for the Capture and Validation of Distributed Systems Requirements". *RE'99, Fourth IEEE Int. Symp. on Requirements Eng.*, Limerick, Ireland, June 1999, 44-53.

[5] Andrade, R. (2000) "Applying Use Case Maps and Formal Methods to the Development of Wireless Mobile ATM Networks". *Lfm2000, The Fifth NASA Langley Formal Methods Workshop*, Williamsburg, VA, USA.

[6] Andrade, R. and Logrippo, L. (2000) "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, Computer Science and Engineering: Part I, Orlando, Florida, 11-16.

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

[8] Buhr, R.J.A. (1998) "Use Case Maps as Architectural Entities for Complex Systems". *IEEE Trans. on Software Eng.* Vol. 24, No. 12, Dec. 1998, 1131-1155.

[9] Chung, J., Nixon, B.A., and Yu, E. (1995) "Using non-functional requirements to systematically select among alternatives in architectural design". *Proc. of the First Int. Workshop on Architecture for Software Systems*. Seattle, USA.

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

[11] Gross, D. and Yu, E. (2001) "From Non-Functional Requirements to Design through Patterns". *Requirements Engineering*, 6:18-36.

[12] Hodges, J. and Visser, J. (1999) "Accelerating Wireless Intelligent Network Standards Through Formal Techniques". *IEEE 1999 Vehicular Technology Conference (VTC'99)*, Houston (TX), USA.

[13] Hurlbut, R. R. (1998) *Managing Domain Architecture Evolution Through Adaptive Use Case and Business Rule Models*. Ph.D. thesis, Illinois Institute of Technology, Chigago, Illinois, USA.

[14] ITU-T (1988) *Recommendation I.130, Method for the characterization of telecommunication services supported by an ISDN and network capabilities of ISDN*. Geneva.

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

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

[17] ITU-T (2001) *Recommendation Z.120 (11/99), Message Sequence Chart (MSC)*. Geneva.
[18] ITU-T (2001) *Recommendation Z.140: The Tree and Tabular Combined Notation version 3 (TTCN-3): Core language*. Geneva.
[19] ITU-T, URN Focus Group (2002), *Draft Rec. Z.150 – User Requirements Notation (URN)*. Geneva, February. http://www.UseCaseMaps.org/urn
[20] ITU-T, URN Focus Group (2002), *Draft Rec. Z.151 – Goal-oriented Requirements Language (GRL)*. Geneva.
[21] ITU-T, URN Focus Group (2002), *Draft Rec. Z.152 – UCM: Use Case Map Notation (UCM)*. Geneva.
[22] Lamsweerde, A. V. (2000), "Requirements Engineering in the Year 00: A Research Perspective". In the *Proceedings of 22nd International Conference on Software Engineering (ICSE)*. Limerick, Ireland, ACM press.
[23] Liu, L. et al. (2001) *GRL and OME*. http://www.cs.toronto.edu/km/GRL/
[24] Liu, L. and Yu, E. (2001) "From Requirements to Architectural Design — Using Goals and Scenarios". *From Software Requirements to Architectures Workshop (STRAW 2001)*, Toronto, Canada, May 2001.
[25] Liu, L. and Yu, E. (2002) "Designing Web-Based Systems in Social Context: A Goal and Scenario Based Approach". *CAiSE'02*, Toronto, Canada, May 2002.
[26] Miga, A., Amyot, D., Bordeleau, F., Cameron, C. and Woodside, M. (2001) "Deriving Message Sequence Charts from Use Case Maps Scenario Specifications". In: *Tenth SDL Forum (SDL'01)*, Copenhagen, Denmark.
[27] Miga, A. (1998) *Application of Use Case Maps to System Design with Tool Support*. M.Eng. thesis, Dept. of SCE, Carleton University, Ottawa, Canada.
[28] Mussbacher, G. and Amyot, D. (2001) "A Collection of Patterns for Use Case Maps". *First Latin American Conference on Pattern Languages of Programming (SugarLoafPLoP 2001)*, Rio de Janeiro, Brazil.
[29] OMG (2001), *Unified Modeling Language Specification*, Version 1.4, May 2001.
[30] OMG (2001), *UML Profile for Scheduling, Performance and Time*. OMG Document ad/2001-06-14, http://www.omg.org/cgi-bin/doc?ad/2001-06-14, June.
[31] Sales, I. and Probert, R. (2000) "From High-Level Behaviour to High-Level Design: Use Case Maps to Specification and Description Language". *SBRC'2000*, Belo Horizonte, Brazil.
[32] Use Case Maps Web Page and UCM User Group. http://www.UseCaseMaps.org
[33] Woodside, M. and Petriu, D. (2002) "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.
[34] Yu, E. (1997) "Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering". *Proc. of the 3rd IEEE Int. Symp. on Requirements Engineering (RE'97)*, Washington, D.C., USA, 226-235.
[35] Yu, E. and Mylopoulos, J. (1998) "Why Goal-Oriented Requirements Engineering". *Proceedings of the 4th International Workshop on Requirements Engineering: Foundations of Software Quality*, Pisa, Italy, 15-22.