From Requirements to Architectural Design –Using Goals and Scenarios

Lin Liu Eric Yu
Faculty of Information Studies, University of Toronto
{liu, yu}@fis.utoronto.ca

Abstract

To strengthen the connection between requirements and design during the early stages of architectural design, a designer would like to have notations to help visualize the incremental refinement of an architecture from initially abstract descriptions to increasingly concrete components and interactions, all the while maintaining a clear focus on the relevant requirements at each step. We propose the combined use of a goal-oriented language GRL and a scenarios-oriented architectural notation UCM. Goals are used in the refinement of functional and non-functional requirements, the exploration of alternatives, and their operationalization into architectural constructs. The scenario notation is used to depict the incremental elaboration and realization of requirements into architectural design. The approach is illustrated with an example from the telecom domain.

1. Introduction

In the context of Requirement Engineering and system architectural design, goal-driven and scenario-based approaches have proven useful. In order to overcome some of the deficiencies and limitations of these approaches when used in isolation, proposals have been made to combine goals, scenarios and agents in order to guide the RE to architectural design process. As there are both overlaps and gaps between these approaches, their interactions are complicated and highly dynamic.

In general, goals describe the objectives that the system should achieve through the cooperation of agents in the software-to-be and in the environment. It captures "why" the data and functions are there, and whether they are sufficient for achieving the high-level objectives that arise naturally in the requirements engineering process. The integration of explicit goal representations in requirements models provides a criterion for requirements completeness, i.e. the requirements can be judged as complete if they are sufficient to establish the goal they are refining.

Scenarios present possible ways to use a system to accomplish some desired functions or implicit purposes. Typically, it is a temporal sequence of interaction events between the intended software and its environment (composed of other systems or humans). A scenario could be expressed in forms such as narrative text, structured text, images, animation or simulations, charts, maps, etc. The content of a scenario could describe either system-environment interactions or events inside a system. The purpose and usage of scenarios also vary greatly. They can be used as a means to elicit or validate system requirements, as concretization of use-oriented system descriptions, or as a basis for test cases. Scenarios have also become popular in other fields, notably human-computer interaction and strategic planning.

In this paper, we explore the combined use of goal-oriented and scenario-based models during architectural design. The GRL language is used to support goal and agent oriented modelling and reasoning, and to guide the architectural design process. The UCM notation is used to express the architectural design at each stage of development. The scenario orientation of UCM allows the behavioral aspects of the architecture to be visualized at varying degrees of abstraction and levels of detail.

In the next section, basic concepts of GRL and UCM are introduced. In Section 3, we summarize our approach of using GRL and UCM together to model incrementally requirements and architectural design. In section 4, a case study in the wireless telecommunication domain is used to illustrate the proposed approach. In section 5, related works are discussed. Conclusions and future work are discussed in section 6.

2. GRL and UCM

2.1 GRL

Goal-oriented Requirement Language (GRL) is a language for supporting goal and agent oriented modelling and reasoning about requirements, especially for dealing with non-functional requirements (NFRs)[4][11]. It provides constructs for expressing various types of concepts that appear during the requirements and high-level architectural design process. There are three main categories of concepts: intentional elements, links, and actors. The intentional elements in GRL are goal, task, softgoal and resource. They are intentional because they are used for models that allow answers to questions such as why particular behaviors, informational and structural aspects were chosen to be included in the system requirements, what alternatives were considered, what criteria were used to deliberate among alternative options, and what the reasons were for choosing one alternative over the other.

A GRL model can either be composed of a global goal model, or a series of goal models distributed amongst several actors. If a goal model includes more than one actor, then the intentional dependency relationships between actors can also be represented and reasoned about. In this paper, the distributed goal models will not be discussed; [12] studies the roles of agent-orientation in requirements and architectural design.

A goal is a condition or state of affairs in the world that the stakeholders would like to achieve. In general, how the goal is to be achieved is not specified, allowing alternatives to be considered. A goal can be either a business goal or a system goal. Business goals are about the business or state of the business affairs the individual or organization wishes to achieve. System goals are about what the target system should achieve, which, generally, describe the functional requirements of the target information system. In GRL graphical representation, goals are represented as a rounded rectangle with goal name inside.

A task specifies a particular way of doing something. When a task is specified as a sub-component of a (higher-level) task, this restricts the higher-level task to that particular course of action. Tasks can also be seen as solutions in the target system, which will satisfice the softgoals (called operationalizations in NFR) or achieve goals. These solutions provide operations, processes, data representations, structuring, constraints and agents in the target system to meet the needs stated in the goals and softgoals. In GRL graphical representation, tasks are represented as a hexagon with the task name inside.

A softgoal is a condition or state of affairs in the world that the actor would like to achieve, but unlike a (hard) goal, there are no clear-cut criteria for whether the condition is achieved, and it is up to the developer to judge whether a particular state of affairs in fact sufficiently achieves the stated softgoal. Softgoals are used to represent NFRs in the future system. Non-functional requirements, such as performance, security, accuracy, reusability, interoperability, time to market and cost are often crucial for the success of a software system. They should be addressed as early as possible in the software lifecycle, and be properly reflected in software architecture before a commitment is made to a specific implementation. In GRL graphical representation, a softgoal, which is "soft" in nature, is shown as an irregular curvilinear shape with softgoal name inside.

A resource is an (physical or informational) entity, about which the main concern is whether it is available. Resources are shown as rectangles in GRL graphical representation.

Intentional links in GRL include means-ends, decomposition, contribution, correlation and dependency links. Means-ends links are used to describe how goals are in fact achieved. Each task connected to a goal by a means-ends link is an alternative means for achieving the goal. Decompositions define what other sub-elements needs to be achieved or available in order for a task to be performed. Contribution links describe how softgoals, tasks and links contribute to others. A contribution is an effect that is a primary desire during modelling. They can be either negative or positive, and orthogonally can be either sufficient or partial. Following are the graphical representations for links.

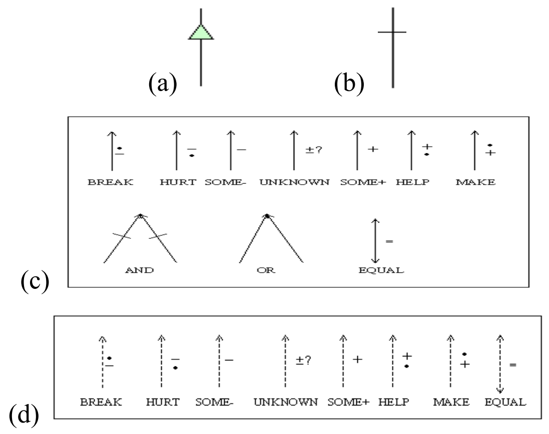

Figure 1 (a) Means-Ends; (b)Decomposition; (c) Contribution; (d) Correlation

2.2 UCM

Use Case Maps (UCM)[2][3] provide a visual notation for scenarios, which is proposed for describing and reasoning about large-grained behavior patterns in systems, as well as the coupling of these patterns. Something new that UCM offers is that it provides a behavioral framework for making architectural decisions at a high level of design, and characterizes behavior at the architectural level once the architecture has been determined.

Use Case Maps notation employ scenario paths to illustrate causal relationships among responsibilities. Furthermore, UCM provides an integrated view of behavior and structure by allowing the superimposition of scenario paths on a structure of abstract components. The combination of behavior and structure in UCMs enables reasoning about architecture. Scenarios in UCM can be structured and integrated incrementally. This enables reasoning about and detection of potentially undesirable interactions between scenarios and components. Furthermore, the dynamic (run-time) refinement capabilities of the UCM language allow for the specification of (run-time) policies and for the specification of loosely coupled systems where functionality is decided at runtime through negotiation between components.

The UCM notation is mainly composed of path elements, and also of components. The basic path notation has simple operators for causally linking responsibilities in sequences, as alternatives, and in parallel. More advanced operators can be used for structuring UCMs hierarchically and for representing exceptional scenarios and dynamic behavior. Components can be of different natures, allowing for a better and more appropriate description of some entities in a system.

Basic elements of UCMs are start points, responsibilities, end points and components. Starting points are filled circles representing pre-conditions or triggering causes. End points are bars representing post-conditions or resulting effects. Responsibilities are crosses representing actions, tasks or functions to be performed. Components are boxes representing entities or objects composing the system. Paths are wiggle lines that connect start points, responsibilities and end points. A responsibility is said to be bound to a component when the cross is inside the component. In this case, the component is responsible for performing the action, task, or function represented by the responsibility.

Alternatives and shared segments of routes are represented as overlapping paths. An OR-join merges two (or more) overlapping paths while an OR-fork splits a path into two (or more) alternatives. Alternatives may be guarded by conditions represented as labels between square brackets. Concurrent and synchronized segments of routes are represented through the use of a vertical bar. An AND-join synchronizes two paths while an AND-fork splits a path into two (or more) concurrent segments.

When maps become too complex to be represented as a single UCM, a mechanism for defining and structuring sub-maps become necessary. A top level UCM, referred to as a root map, can include containers (called stubs) for sub-maps (called plug-ins). Stubs are represented as diamonds. Stubs and plug-ins are used to solve the problems of layering and scaling or the dynamic selection and switching of implementation details.

Other notational elements include: timer, abort, failure point, and shared responsibilities. A detailed introduction to and examples of these concepts can be found in [2] [3].

Although UCM can represent the alternatives of system architectural design precisely in a high-level way, the tradeoffs between these alternatives, and the intentional features of making a design decision can not be explicitly shown in UCM models. And inevitably, as in other scenario-based approaches, UCM models are partial.

GRL provides support for reasoning about scenarios by establishing correspondences between intentional GRL elements and functional components and responsibilities in scenario models of UCM. The modelling of goals and scenarios is complementary and may aid in identifying further goals and additional scenarios (and scenario fragments) important to architectural design, thus contributing to the completeness and accuracy of requirements, as well as to the quality of architectural design.

3. Modelling Methodology with GRL+UCM

A complete requirements specification should clarify the objectives of a system, the concrete behaviors and constraints of the system-to-be, and the entities that will be responsible for certain functions in that system.

The goal-based approach focuses on answering the "why" questions of requirements (such as "why does the system need to be redesigned?" or "why is a new architecture for TSMA necessary?"). The strength of this approach is that it covers not only functional requirements but also non-functional requirements (in other words, the quality requirements). Although goal-orientation is highly appropriate for requirements engineering, goals are sometimes too abstract to capture at once. Whereas goals can often be made explicit only after a deeper understanding of the system has been achieved, it is often possible to create operational scenarios about using the hypothetical system relatively easily.

In our approach, first GRL models are created, the original business goals and non-functional requirements are refined and operationalized, until some concrete design decisions are obtained. These design decisions are then further elaborated into UCM scenarios. In this step, "how" questions are asked instead of "what".

At the same time, UCM scenarios are used to describe the behavioral features and architectures of the intended system in the restricted context of achieving some implicit purposes, which basically answers the "what" questions such as "what should the system do as providing a in-coming call service?" or "what is the process of wireless call transmission?". Then, by issuing "why" questions referring to these scenarios (e.g. "why reside a function entity in this network entity instead of another network entity?") some implicit system goals are made explicit.

The GRL-UCM combination aims to elicit, refine and operationalize requirements incrementally until a satisfying architectural design is launched. The general steps of the process are illustrated in Figure 2.

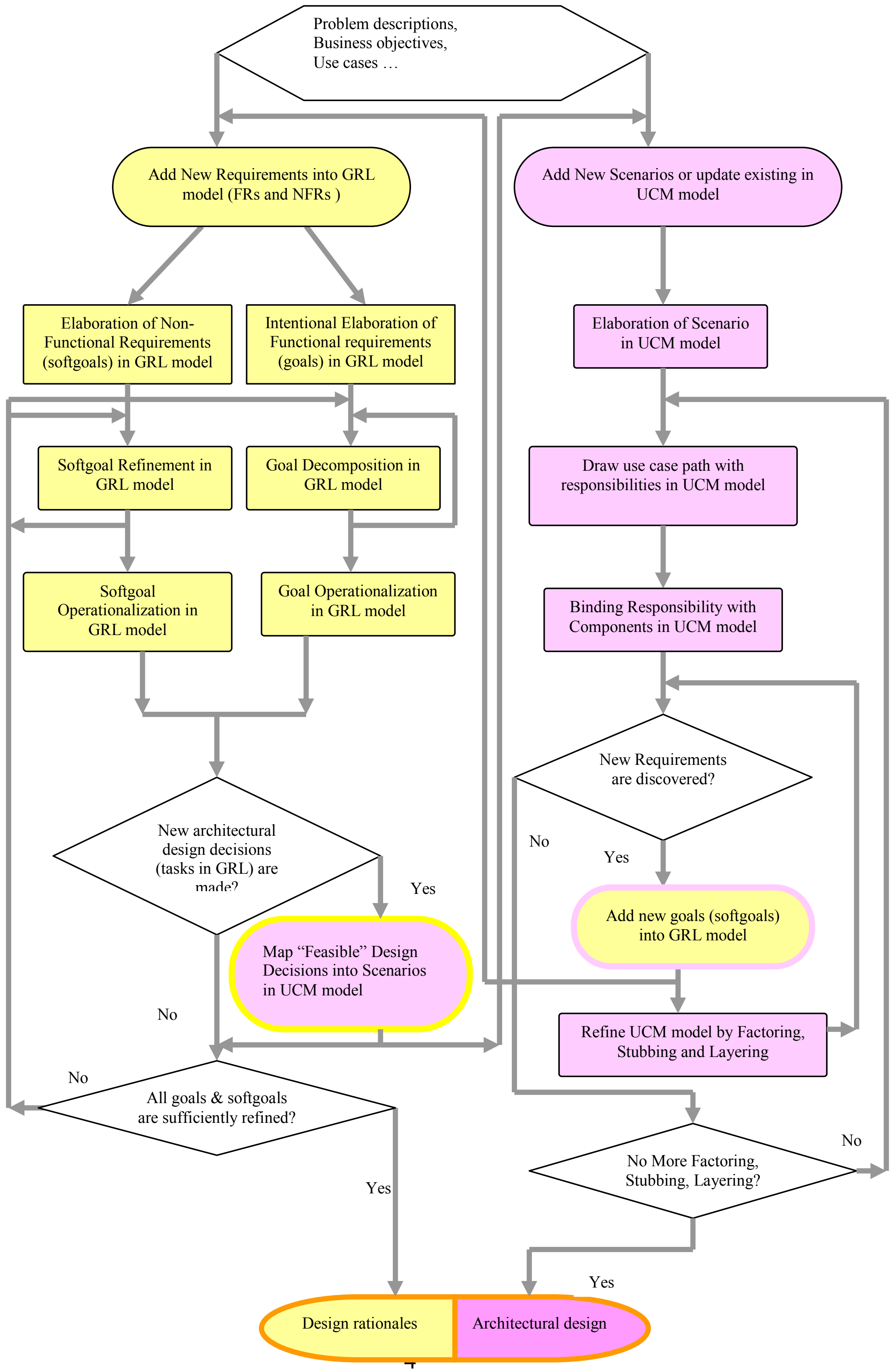
Problem descriptions,
Business objectives,
Use cases …
Add New Requirements into GRL model (FRs and NFRs)
Add New Scenarios or update existing in UCM model
Elaboration of Non-Functional Requirements (softgoals) in GRL model
Intentional Elaboration of Functional requirements (goals) in GRL model
Elaboration of Scenario in UCM model
Softgoal Refinement in GRL model
Goal Decomposition in GRL model
Draw use case path with responsibilities in UCM model
Softgoal Operationalization in GRL model
Goal Operationalization in GRL model
Binding Responsibility with Components in UCM model
New Requirements are discovered?
No
Yes
New architectural design decisions (tasks in GRL) are made?
Yes
Add new goals (softgoals) into GRL model
Map "Feasible" Design Decisions into Scenarios in UCM model
No
Refine UCM model by Factoring, Stubbing and Layering
No
All goals & softgoals are sufficiently refined?
No
No More Factoring, Stubbing, Layering?
Yes
Yes
Design rationales
Architectural design

Figure 2. Integration of Goal-Oriented and Scenario-based Modelling

4. Case Study

To illustrate the interleaved application of GRL and UCM, we use an example from the mobile telecommunication systems domain [9]. A mobile switching center (MSC) is required to support narrowband and wideband voice, data and imaging services and so on. We use GRL and UCM together to trace the process from capturing the original business objective, to refining and operationalizing this objective, and tradeoffs between architectures.

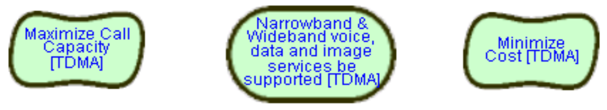
Maximize Call Capacity [TDMA]
Narrowband & Wideband voice, data and image services be supported [TDMA]
Minimize Cost [TDMA]

Figure 3: Original Goal Model with one functional goal and two non-functional goals

Step 1: GRL Model- The original functional and non-functional requirements are represented as three nodes in Figure 3. The goal node in the middle represents the functional requirement on the TDMA that it must support narrowband and wideband voice, data and image services. There are two quality requirements identified at the very beginning, one is to maximize the call capacity in the new TDMA architecture, the other is to minimize the cost of the infrastructure.

Step 2: UCM Model- The essential scenario that implements the functional goal in the GRL model is given in Figure 4. The scenario path (denoted by the wiggly line) represents a causal sequence of responsibilities (denoted by a cross) that is triggered by an initial event (denoted by a filled circle), resulting in a terminating event (denoted by a bar). The responsibilities are not bound to any components.

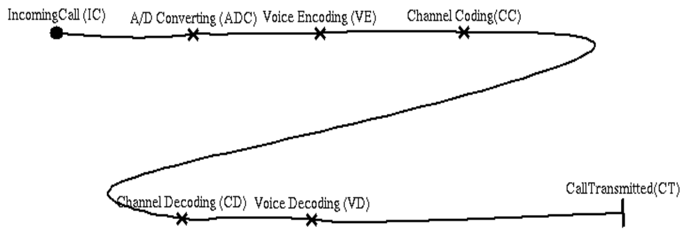

Figure 4: Unbound use case path with responsibilities

Step 3: UCM Model – Binding Responsibilities to components of the future system.

The following UCM diagram (Figure 5) shows the existing TDMA architecture. In this architecture, the decoder of the voice coder is located in the base station. This implies that the 64-kb/s PCM of decoded voice will be transmitted out of the cell site to the switch for each call, requiring an entire Digital Signal level 0 channel (DS0) to support the 64-kb/s signal.

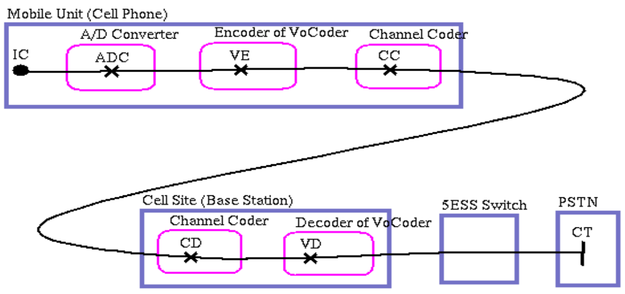

Figure 5: Bound use case path with functional objects and physical entities

Step 4: GRL Model – Goal Refinement and Operationalization. In the goal model in Figure 6, the original functional goal is connected to the task node representing the current solution for TDMA. It can be seen that the current solution can cause some delay per call, which may negatively influence the voice quality of the call, and the call capacity of the system. This solution does not use packet switching protocol enough, so cost savings are lost. Traffic performance between base station and switch is also low.

With current infrastructure, the efficiency of TDMA is barely equivalent to that of an analog system, which means the requirements on improving the capacity, quality, cost and performance are all weakly denied.

Step 5: UCM Model – Change the Binding of Responsibilities.

As the above design could not satisfy the non-functional requirements, other options should be explored. The UCM model (in Figure 7) describes a new

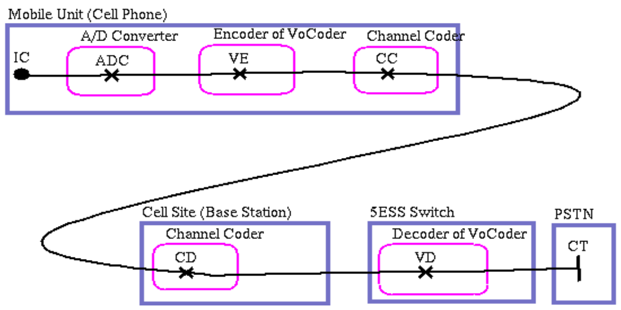

Figure 7: UCM model of another way of binding

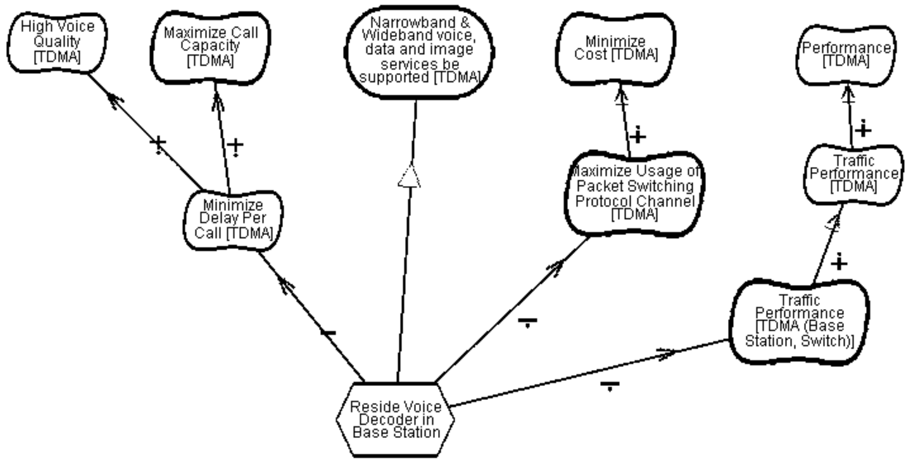
High Voice Quality [TDMA]
Maximize Call Capacity [TDMA]
Narrowband & Wideband voice, data and image services be supported [TDMA]
Minimize Cost [TDMA]
Performance [TDMA]
Minimize Delay Per Call [TDMA]
Maximize Usage of Packet Switching Protocol Channel [TDMA]
Traffic Performance [TDMA]
Traffic Performance [TDMA (Base Station, Switch)]
Reside Voice Decoder in Base Station

Figure 6: Refined GRL model with one design solution and more non-functional

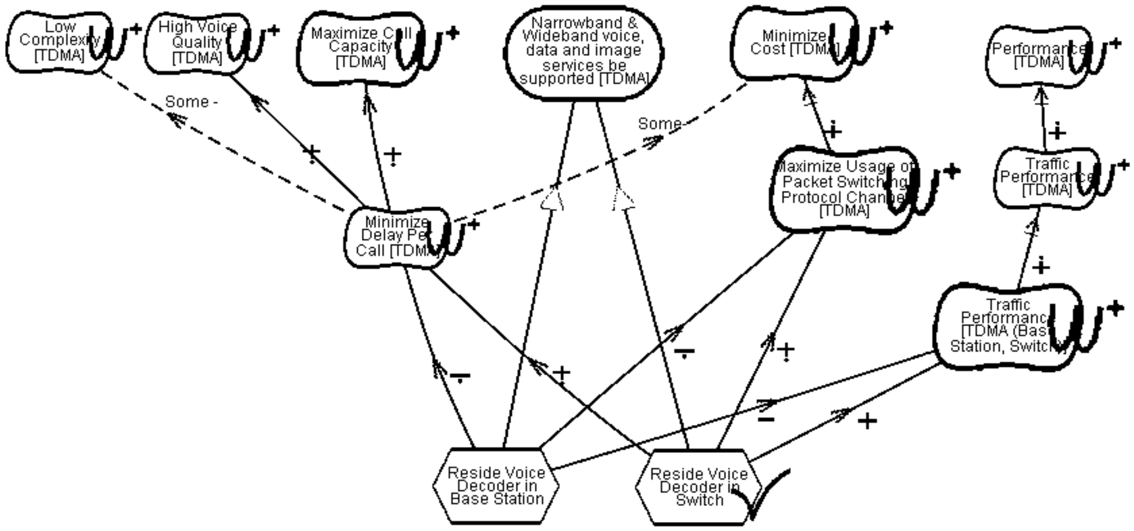
Low Complexity [TDMA]
High Voice Quality [TDMA]
Maximize Call Capacity [TDMA]
Narrowband & Wideband voice, data and image services be supported [TDMA]
Minimize Cost [TDMA]
Performance [TDMA]
Some -
Some-
Maximize Usage of Packet Switching Protocol Channel [TDMA]
Traffic Performance [TDMA]
Minimize Delay Per Call [TDMA]
Traffic Performance [TDMA (Base Station, Switch)]
Reside Voice Decoder in Base Station
Reside Voice Decoder in Switch

Figure 8: GRL model evaluating the contribution of the new architecture to NFRs

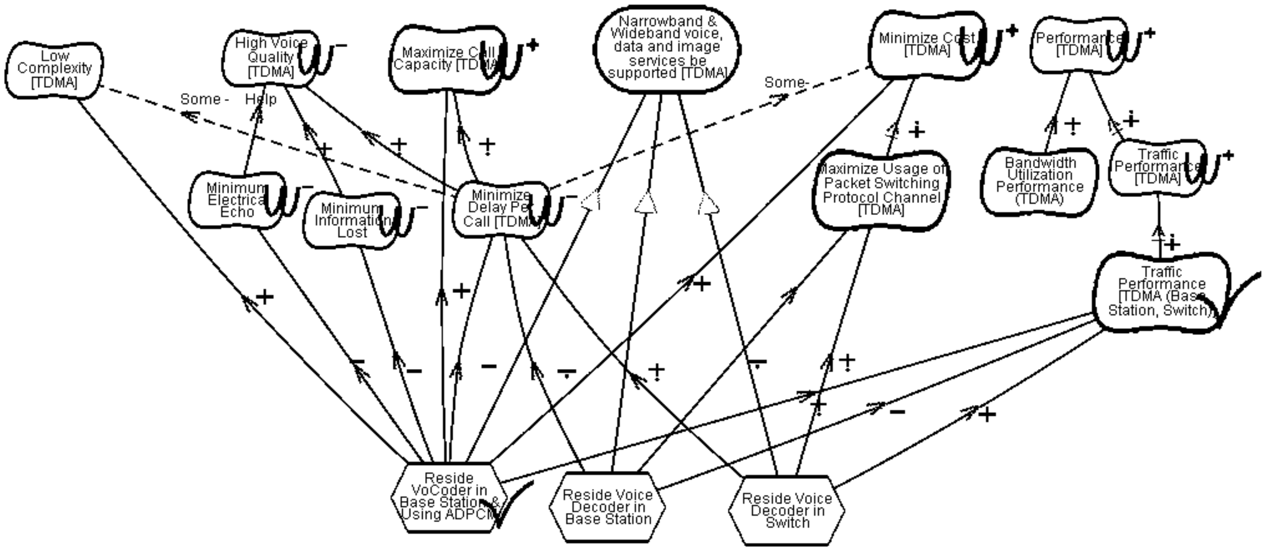
Low Complexity [TDMA]
High Voice Quality [TDMA]
Maximize Call Capacity [TDMA]
Narrowband & Wideband voice, data and image services be supported [TDMA]
Minimize Cost [TDMA]
Performance [TDMA]
Some -
Help
Some-
Minimum Electrical Echo
Minimum Information Lost
Minimize Delay Per Call [TDMA]
Maximize Usage of Packet Switching Protocol Channel [TDMA]
Bandwidth Utilization Performance (TDMA)
Traffic Performance [TDMA]
Traffic Performance [TDMA (Base Station, Switch)]
Reside VoCoder in Base Station & Using ADPCM
Reside Voice Decoder in Base Station
Reside Voice Decoder in Switch

Figure 10: Goal model evaluating the viability of solution 3

architecture to improve the capacity of the TDMA cellular telecommunications system. In this design, the decoder of voice coder is relocated into the switch instead of the base station, so for each call the base station transmits an 8-kb/s signal – rather than a 64-kb/s signal to the switch. In such a system, a theoretical maximum of 8× capacity improvement is possible.

Step 6: GRL Model – Contributions of the new architecture to the non-functional requirements. The GRL model (in Figure 8) shows that the new TDMA architecture with voice coder relocated in the switch, thus weakly satisficing the requirements on improving the capacity, quality and performance, though at the same time the cost and complexity are negatively influenced. To minimize call delay somehow increased the complexity and cost of the architecture (represented in Figure 7 with correlation links). Comparing the two architectures, if a cell site supported x calls, the previous architecture would need x DS0s to support those calls. But the voice coder relocation architecture would requirement only x/3 DS0s. Given the evaluation result, we conclude that the new architecture is an acceptable design.

GRL evaluates the satisfication of softgoal via a qualitative labeling procedure. The label of a high level node is propagated from the label of low level nodes, and the contribution from these nodes.

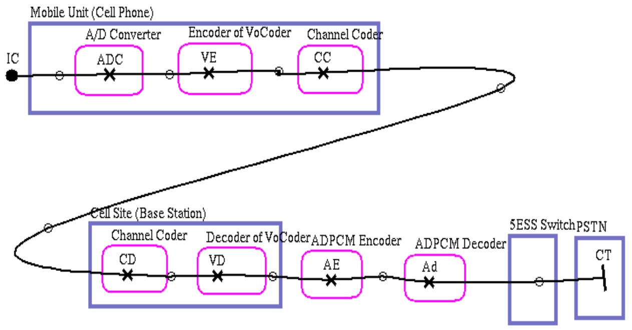

Figure 9: UCM model of solution 3: new responsibilities and functional units added

However, before putting this relocating solution into practice, other possible solutions should also be considered. The following is one possible solution without relocating the decoder of the voice coder.

Step 7: UCM Model – In Figure 9, by adding new functional units without changing the location of the decoder of voice coder, a simpler solution is described. For increasing call capacity, 32-kb/s adaptive differential pulse code modulation (ADPCM) equipment is used with the voice decoder still in the base station.

Step 8: GRL Model – Evaluation of the new architecture according to the non-functional requirements, and comparison to other options. The GRL model in Figure 10 shows that this simplest solution weakly satisficed the requirements on improving the capacity, performance, low cost and low complexity. However, voice quality is seriously eroded by the electrical echo, the delay for the extra cycle of speech coding, and the information lost produced in this kind of architecture. If voice quality is a lower priority for a user, this architecture could also be an acceptable choice.

Having analyzed the benefits and tradeoffs of these architectures, we could see that UCM is a natural counterpart to GRL in the process from requirements to high-level design, because it provides a concrete model of each design alternative. Based on the architectural features in such a model, new non-functional requirements may be detected and added to the GRL model. At the same time, in the GRL model, new means to achieve the functional requirements could always be explored and be embodied in a UCM model.

In the case study above, the UCM model are rather simplistic because we have only tackled the highest level of architectural design in the wireless telecommunication protocol. As we go down to a sufficiently detailed design, a UCM model may be fairly complex, and more modelling constructs could be used. Figure 11 (From [1]) is a root map of a mobile system. It illustrates the “big picture” of a simplified mobile wireless communication system. As shown in this graph, stubs are used to hide details of certain sections of a scenario, e.g., the mobility management functions (MM stub), the communication management functions (CM stub), the handoff procedures (HP stub) and handoff failure actions (HFA stub).

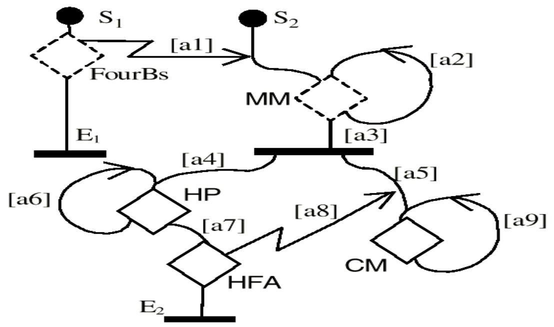

Figure 11: The Mobile system Root Map[1];

A plug-in gives more detail for the stubs. Due to space limitations, we won’t present all of the plug-ins or explain the details of each responsibility. However, one thing need to note is, for each stub (especially a static stub), there may be more than one ways to refine the plug-ins. This is a powerful construct to form new design alternatives by integrating different possible designs of various parts of the system.

Figure 12 depicts an integrated scenario of establishing a call between the originating and the terminating parties. There may be other possible designs, but we won't investigate because of space limitation. Components in

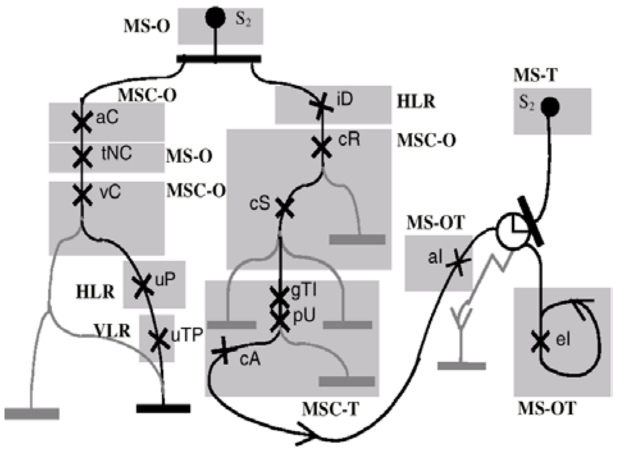

Figure 12: Integration of Scenario Fragments [1]

Figure 12 include: Originating Mobile Station (MS-O), Originating Mobile switching Center (MSC-O), Home Location Register (HLR), Visitor Location Register (VLR), Terminating Mobile Station (MS-T), Terminating Mobile switching Center (MSC-T), Originating and Terminating Mobile Stations (MS-OT).

Although we used a telecommunication system architecture example, the approach is applicable to allocation of responsibility in software systems in general, where there are usually conflicting goals and tradeoffs.

5. Related work

As existing scenario-based approaches serve different purposes, use different representational features, and have different analysis capabilities, the concept of scenario needs to be differentiated along the dimensions.

In Krutchen's 4+1 model of software architecture [7], scenarios are used to show connections across other views such as logical views, process views, physical views and development views. The use of a multiple view model of architecture makes it possible to separately address the concerns of the various stakeholders. However, with an model composed of several separate views it is not easy to keep a coherent track of the incremental design process. As UCM shows the behavioral and structural aspects together as one view, it is good for showing the incremental elaboration of the design.

The Software Architecture Analysis Method (SAAM) [5, 6] is a scenario-based method for evaluating architectures. It provides a means to characterize how well a particular architectural design responds to the demands placed on it by a particular set of scenarios. Based on the notion of context-based evaluation of quality attributes, scenarios are used as a descriptive means of specifying and evaluating quality attributes. For example, to evaluate the modifiability of a user interface architecture Serpent, two scenarios are considered, one is "changing the windows system/toolkit", and the other is "adding a single option to a menu". The similarities between this paper and SAAM include: both works are concerned with the quality of architecture, and both use scenarios to describe architectures. However, there are obvious differences: SAAM scenarios are use-oriented scenarios, which are designed specifically to evaluate certain quality attributes of architecture. In GRL+UCM, scenarios are more design-oriented, being concerned with refinement of system requirements. The quality of the architectures corresponding to these scenario are judged based on expert knowledge rather than simulations or tests as in SAAM.

The combined use of goals and scenarios have been explored within RE, primarily for eliciting, validating and documenting software requirements. Van Lamsweerde and Willement studied the use of scenarios for requirement elicitation and explored the process of inferring formal specifications of goals and requirements from scenario descriptions in [8]. Though they thought goal elaboration and scenario elaboration are intertwined processes, their work regarding scenarios in [8] mainly focuses on goal elicitation. Our emphasis is the other way around, i.e., how to use goal model (especially NFRs) to direct scenario – based architectural design. The fundamental point is that both the goal-oriented modelling in GRL and the scenario-based modelling in UCM run through requirement process to architectural design, and also their interactions.

In the CREWS project, Collete Rolland et al. have looked into the coupling of goal and scenario in RE with CREWS-L'Ecritoire [10]. In CREWS-L'Ecritoire, Scenarios are used as a means to elicit requirements/goals of the system-to-be. Their method is semi-formal. Both goals and scenarios are represented with structured textual prose. The coupling of goal and scenario could be considered as a "tight" coupling, as goals and scenarios are structured into <Goal, Scenario> pairs, which are called "requirement chunks". Their work focuses mainly on the elicitation of functional requirements/goals.

In GRL+UCM, both graphical representations and textual descriptions (in natural language and XML format) for goal models and scenario models are provided. The semi-formal graphical notations are intended to be used during the early stages of architectural design, to help explore and prune the space of design alternatives. The current coupling of goal and scenario is loose, as goal models and scenario maintain their local completeness, and one scenario may refer to more than one goal, and vice versa. There are no rigid constraints on the requirements process. That is, the goal model and scenario model can be developed in parallel simultaneously, they interact whenever there are design decisions that need to be traded off, or new design alternatives need to be sought, or new business goals or non-functional requirements are discovered. Both functional and non-functional requirements are considered, with perhaps more attention being devoted to non-functional requirements. The modelling process involves both requirements engineering activities and high-level architecture design.

6. Conclusion and future works

In summary, goal-orientation and scenario-orientation complement each other not only in requirement engineering but also during the incremental architectural design process. The combined use of GRL and UCM enables the description of functional and non-functional requirements, abstract requirements and concrete system architectural models, intentional strategic design rationales and non-intentional details of concurrent, temporal aspects of the future system.

In the future, we hope to look into visualizing the connections between GRL and UCM to support a more formal combination of the two notations. This would allow the mapping and interaction between the two kinds of models to be less dependent on expert users.

GRL and UCM are knowledge containers. To make good use of them, we need to acquire both software design knowledge and more knowledge of various domains, and represent this knowledge in GRL and UCM. The development of such a repository would enable the reuse of knowledge and aid in guiding the design process.

7. Acknowledgements

The work of this paper is motivated by an original submission to ITU-T study group 10 on the topic of User Requirements Notation (URN). The kind cooperation of people from Mitel Networks, Nortel Networks and other institutions is gratefully acknowledged.

8. References

[1] Andrade, R. and Logripo, L. Reusability at the Early Development Stages of Mobile Wireless Communication Systems. In *Proceedings of the 4th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2000)*, 12, Computer Science and Engineering: Part I, July 2000. Orlando, Florida, 11-16.

[2] Amyot, D. Use Case Maps Quick Tutorial Version 1.0. On-line at: http://www.usecasemaps.org/pub/UCMtutorial/UCMtutorial.pdf.

[3] Buhr, R.J.A. and Casselman, R.S. Use Case Maps for Object Oriented Systems, Prentice-Hall, USA, 1995.

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

[5] Kazman, R. Using Scenarios in Architecture Evaluations. *SEI Interactive,* June 1999. On-line at http://interactive.sei.cmu.edu/Columns/The_Architect/1999/June/Architect.jun99.htm

[6] Kazman, R., Bass, L., Abowd, G. and Webb, M. SAAM: A Method for Analyzing the Properties of Software Architectures. In *Proceedings of the 16th International Conference on Software Engineering*. May 1994. Sorrento, Italy. 81-90.

[7] Kruchten, P. The 4+1 view Model of Software Architecture. *IEEE Software*, 12, 6 (November 1995). 42-50.

[8] Lamsweerde, A.V., Willemet, L. Inferring Declarative Requirements Specifications from Operational Scenarios. *IEEE Transactions on Software Engineering*, Special Issue on Scenario Management, December 1998.

[9] Lee, A.Y. and Bodnar, B.L. Architecture and Performance Analysis of Packet-Based Mobile Switching Center-to-Base Station Traffic Communications for TDMA. *Bell Labs Journal*. Summer 1997. 46-56.

[10] Rolland, C. , Grosz, G. and Kla, R. Experience With Goal-Scenario Coupling In Requirements Engineering. In *Proceedings of the IEEE International Symposium on Requirements Engineering* 1998. June 1999. Limerick, Ireland.

[11] Yu, E. and Mylopoulos, J. Why Goal-Oriented Requirements Engineering. In *Proceedings of the 4th International Workshop on Requirements Engineering: Foundations of Software Quality*. June 1998, Pisa, Italy. E. Dubois, A.L. Opdahl, K. Pohl, eds. Presses Universitaires de Namur, 1998. pp. 15-22.