CAPTURING NON-FUNCTIONAL SOFTWARE REQUIREMENTS USING THE USER REQUIREMENTS NOTATION

K. Saleh and A. Al-Zarouni
Department of Computer Science, American University of Sharjah
Box 26666 Sharjah United Arab Emirates
Email: ksaleh@aus.ac.ae

ABSTRACT

Non-functional requirements (NFRs) are crucial software requirements that have to be specified early in the software development process while eliciting the functional software requirements (FRs) [1,2]. NFRs can be viewed as additional constraints that have to be satisfied by the functional requirements. The non-satisfaction of NFRs is one of the main reasons for the failure of software projects. Formal specification techniques for describing FRs has been considered excessively in the software literature. Artifacts for developing a Use Case Model using the Unified Modeling Language (UML) are very useful for eliciting FRs [3]. However, the formal specification of NFRs is a current topic of interest in the software community. Formalizing the capture of NFRs would facilitate a formal documentation, validation and testability of NFRs in software systems. The User Requirements Notation (URN) is a standard proposed by the International Telecommunication Union (ITU) [4,5] for the representation of requirements in future telecommunication systems and services and in software systems in general. This paper focuses on NFRs and how to model them using the Goal-Oriented Requirements Language (GRL), a component of URN [6]. We illustrate the use of GRL for the capture of NFRs for mobile agent systems [7, 9, 11]. The GRL diagrams for the NFRs of this system were developed using the Organizational Modeling Environment (OME) tool [8].

Keywords: Mobile Agent System, Non-Functional Requirements, User Requirements Notation.

1. INTRODUCTION

The level of success of a software system can be assessed based on its satisfaction of two complementary types of requirements. First, the user-centered requirements that concentrate on the functionalities provided to the system’s user. Second, the context-centered requirements which can be system, process and human requirements. The latter type of requirements are referred to as Non-Functional Requirements (NFRs). NFRs are mainly system’s constraints that may affect greatly the operational environment and design choices a developer may pursue during the development of the software. NFRs include among other things, operational environment: hardware and software interfaces, accuracy, performance: timeliness and storage requirements, security, reliability, maintainability, portability, robustness, and usability. Functional Requirements describe mainly the visible and external input and output interactions with the system under consideration, whereas Non-Functional Requirements are those that impose special conditions and qualities on the system to develop. Consequently, system acceptance testing is based on both functional and non-functional system’s requirements.

International standards were developed to meet the needs for the modeling and specification of software system’s functional behaviors. The Unified Modeling Language (UML) is a standard developed by the Object Management Group, a consortium of software companies, to deal specifically with software systems, and systems in general [3]. Recently, because of the failures of software systems to satisfy their NFRs, the case and need for developing standards for formalizing the description and capture of these requirements became more evident. The International Telecommunication Union (ITU) Group 17 has started to work towards the standardization of a User Requirements Notation (URN) targeting the description of requirements for future telecom systems and services. URN provides a notation for the representation of non-functional requirements (NFRs) such as performance, cost, security, and usability, in addition to a complementary scenario notation for functional requirements. The proposed NFR notation is the Goal-oriented Requirement Language (GRL) is
the first attempt for the explicit capture and representation of NFRs. In addition, Use Case Maps (UCMs) are used as scenarios that help describing and understanding complex functional behavior of software systems [4].

The rest of the paper is organized as follows. Section 2 briefly introduces non-functional requirements. Section 3 briefly describes the ITU-T standards for capturing NFRs, namely, the Goal-oriented Requirements Language of the User Requirements Notation. Section 4 illustrates the use of GRL for the elicitation and capture of the NFRs of mobile agent systems developed in [7, 9, 11]. Section 5 concludes the paper and provides ideas for future research work.

2. NON-FUNCTIONAL REQUIREMENTS

Software is becoming critical in driving the information-based economy. Time-to-market, robustness, and quality are important factors for measuring the success of systems in competitive economies and environments. There is also a need to evolve and adapt rapidly and smoothly in an environment of continuous changes of business requirements. Therefore, there are stronger needs for standards and formalizations of the processes by which we specify and capture these requirements. Requirements capture is one of the initial phases to undertake in the system development process.

Non-functional requirements are properties and qualities the software system must possess while providing its intended functional requirements or services. These types of requirements have to be considered while developing the functional counterparts. They greatly affect the design and implementation choices a developer may make. They also affect the acceptability of the developed software by its intended users.

In the following, we briefly describe the three categories of non-functional requirements that may be imposed on a software system.

2.1 System-related NFRs

These types of requirements impose some criteria related to the internal qualities of the system under development and the hardware/software context in which this system will operate.

- **Operational** requirements: these requirements specify the environment in which the software will be running, including, hardware platforms, external interfaces, and operating systems.
- **Performance** requirements: these requirements specify possibly lower and upper bounds on speed, response time and storage characteristics of the software.
- **Maintainability** requirements: these requirements specify the expected response time for dealing with the various maintenance activities, such as future release dates.
- **Portability** requirements: these requirements specify future plans for porting the software to different operating environments. These requirements are linked to both operating and maintainability requirements and may impose certain design decisions and implementation choices such as the choice of a programming language.
- **Security** requirements: these requirements specify the levels and types of security mechanisms that need to be satisfied during the operations of the system. These may include adherence to specific security standards and plans, and the implementation of specific techniques.

2.2 Process and Project-related NFRs

These types of NFRs impose criteria to be followed while developing the project.

- **Conformance** to standards requirements: these requirements specify the internal, international and industry standards that have to be adhered to when developing the project.
- **Development time and cost** requirements: these requirements are set by the various stakeholders and can be refined in the project plan and modified as the project progresses.
- **Development process** requirements: these requirements impose specific software development model or framework, in addition, to workproducts, milestones and deliverables expected. These requirements will affect the project plan and the chosen development processes.
- **Testing** requirements: these requirements impose specific testing strategies or guidelines to be used in the development process. The types of test plans needed and the testing quality may be affected by these requirements. It can also specify the accepted rate of passed test cases.
**Installation** and **deployment** requirements: these requirements impose specific installation restrictions and guidelines. These restrictions can be related to the platform and location for testing, and the installation plans.

### 2.3 Human-related NFRs

These types of requirements deal with constraints related to the stakeholders and the social and societal context in which the system is deployed.

**Usability** requirements: these requirements specify criteria for judging the degree of usability and user friendliness of the system. It may also specify guidelines or prescribe standards for developing the graphical user interfaces for the system.

**Look and Feel** requirements: these requirements specify criteria and standards the Look and Feel of the system may have to adhere to. These standards may be industry or internal standards.

**Legal** requirements: these requirements specify the laws and regulations that the system has to satisfy to avoid any legal implications. These laws and regulations may be local, regional or international ones.

**Cultural and political** requirements: these requirements specify the social and political constraints defined by the context and environment in which the system stakeholders live. These requirements may have effects on the usability, conformance to standards and legal requirements.

### 3. STANDARDS FOR CAPTURING NFRS

Capturing and specifying NFRs is gaining attention from the international standardization bodies. The International Telecommunication Union, Telecommunication Standardization Sector (ITU-T) Study Group 10 (SG10) is focusing on languages and software aspects for telecommunication systems. SG10 addresses issues related to modelling languages and mechanisms including the Unified Modeling Language (UML), System Description Language (SDL), and Message Sequence Charts (MSC). Study Group 17 deals with the User Requirement Notation (URN) and is finalizing a standard for the URN [4]. In this section, we provide some background on the URN.

#### 3.1 The User Requirement Notation (URN)

URN is designed to deal with both functional and non-functional requirements [6]. The main highlights of the URN design objectives are to:

- **a)** Describe scenarios for functional requirements as high level reusable entities with no design details or sub-structuring provided for these entities.
- **b)** Facilitate the transition and mapping of requirements specification to a high level design and provide alternative architectural designs.
- **c)** Facilitate the elicitation and synthesis of additional requirements.
- **d)** Facilitate the early detection and avoidance of undesirable feature interactions and side-effects among them.
- **e)** Provide facilities and artifacts to describe, analyze and deal with non-functional requirements.
- **f)** Provide facilities to express the relationship between business objectives and goals to system requirements, expressed as scenarios for functional requirements, and global constraints over the system, its development, deployment, maintenance and evolution, and operational processes for non-functional requirements.
- **g)** Provide facilities to capture reusable analysis and design knowledge related to previous experiences dealing with non-functional requirements.
- **h)** Support traceability and mappings to other languages including UML.

All of the above objectives are refined and instituted in the following URN standard documents.

- **Z.150 URN** [4] - This standard includes the motivations, scope, and objectives for the URN, in addition to some basic terminology and requirements engineering concepts. Many specific requirements also refine the list of URN objectives into URN-NFR (URN - Non-Functional Requirements), and URN-FR (URN - Functional Requirements). FRs describe the external and observable input/output interactions with the
software under development. Notations and methodologies for eliciting functional requirements and using them in the software development process are being used extensively in the software industry. However,URN-NFR is used to help standardize the modeling of business goals and product quality features.

- **Z.151 GRL [6]** - This standard presents the Goal-oriented Requirement Language (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 tutorial and references are included.

- **Z.152 UCM [5]** - This standard presents the Use Case Map (UCM) as a notation for URN-FR. Again, visual and concrete XML representations are defined, together with a conformance table, a tutorial, and references.

- **Z.153** - This proposed standard document will focus on the relationships between GRL and UCM, and between URN and other languages such as SDL and UML.

The rest of this section focuses on the non-functional part of URN and specifically on GRL.

### 3.2 The Goal-oriented Requirement Language (GRL)

GRL is a graphical language for supporting goal-oriented modelling and reasoning about non-functional requirements. GRL provides constructs for expressing various types of concepts that appear during the requirement elicitation and capture processes of the software development project, and connects requirements to the business objectives.

There are three main categories of syntactical constructs in GRL: intentional elements, actors, and links or relationships. In the following, we briefly describe each of these constructs and their graphical representations.

#### 3.2.1 Intentional Elements

An intentional element in GRL can be a goal, a task, a softgoal, a belief or a resource. Intentional elements are used to present the different alternative behavioural (dynamic) and structural (static) aspects of the system requirements and concentrate on the rational for choosing a particular alternative over the others. These elements complement the non-intentional elements of the requirements described by scenarios or message sequences. Both intentional and non-intentional elements of the requirements must be elicited to capture the complete user requirement specifications.

**Goal:** A goal is a real-world condition that the system stakeholders would like to achieve, satisfy or meet. A goal can be either a business goal to be achieved by the organization or a system goal (normally functional and quantifiable) to be achieved by the system. A goal is graphically represented by a rounded rectangle as shown in the figure below.

![Figure 1. Goal symbol.](image)

**Task:** A task specifies a process or a set of activities. A large task can include or refer to smaller subtasks. Tasks are used to address goals and softgoals. Tasks provide operations, processes, data representations, structuring, and constraints needed to meet the system’s goals and softgoals. A task is graphically represented by a hexagon.

![Figure 2. Task symbol.](image)
**Resource:** A resource is a physical or data entity. The requirement model should specify whether the resource is available or not. A resource is graphically represented by a rectangle.

![Resource symbol.](image)

**Softgoal:** A softgoal is similar to a (hard) goal, but is normally hard to verify whether it is achieved in an implemented system. In other words, a softgoal is not easily testable. A softgoal is graphically represented by an irregular curvilinear shape.

![Softgoal symbol.](image)

**Belief:** A belief is used to represent a design rationale. Beliefs help tracking and tracing actions and decisions taken at the design and implementation levels. A belief is graphically represented by an ellipse.

![Belief symbol.](image)

### 3.2.2. Actor

An actor is an active entity that carries out actions to achieve certain goals. Graphically, an actor is represented by a circle. An actor can have a boundary that includes some intentional elements. The boundary of an actor is shown as a grey shadow. A model can be developed first by identifying actors and their interrelationships. Then, intentional elements attached to actors are added.

![Actor symbol.](image)

![Actor boundary symbol.](image)

![Actor structure.](image)

### 3.2.3. Intentional Relationships (links):

A link connecting two elements represents the intentional relationship between these two elements. Links and elements constitute the overall goal model. There are four different types of relationships that may exist among model elements: Means-End, Decomposition, Contribution and Correlation relationships.

**Means-End relationship:** This relationship is used to describe how goals are achieved. Each task provided is an alternative means for achieving a given goal. A task may have different impacts on softgoals, which would serve as criteria for evaluating and choosing among task alternatives. Graphically, a means-end link connects an End node with the Means node achieving it. A means-ends link connects a goal with a task achieving it, and a resource with a task making it available.
Decomposition relationship: This relationship allows to show the elements needed to perform a task. Graphically, this link connects a task (component) to its sub-components. In GRL, tasks and goals can be connected by decomposition links. The sub-components of a task can be a goal, task, resource and softgoal.

Contribution relationship: This relationship allows to show how softgoals, task, believes, or links contribute to other intentional elements in the model. Graphically, a contribution link describes how an element contributes to the satisfaction of another element. The following types of contribution relationships exist:

- AND contribution is used to model a positive and necessary contribution of an element.
- OR contribution is used to model a positive and sufficient contribution of an element.
- MAKE contribution is used to model a positive and sufficient contribution of an element.
- BREAK contribution is used to model a negative and sufficient contribution of an element.
- HELP contribution is used to model a positive but not sufficient contribution of an element.
- HURT contribution is used to model a negative but not sufficient contribution of an element.
- SOME+ contribution is used to model a positive contribution of an element with an unknown extent.
- SOME- contribution is used to model a negative contribution of an element with an unknown extent.
- EQUAL contribution is used to model a neutral contribution (equal in both directions).
- UNKNOWN contribution is used to model a contribution the extent and direction of which is unknown at the modeling phase.

The graphical representation of each type of contribution is shown in Figure 11.

Correlation Relationship: This relationship allows for expressing knowledge about interactions between intentional elements. The graphical representation of a correlation link is very similar to a contribution link but the arrows are dashed.
4. NFRS FOR MOBILE AGENT SYSTEMS

In this section, we illustrate the use of GRL for the elicitation of the NFRs of Mobile Agent Systems. We have used the mechanisms developed in the mobile agent systems described in [7, 9, 11]. The Organization Modelling Environment (OME) tool was used to develop the graphical model of these NFRs. OME is a goal-oriented modeling and analysis tool. It provides users with a user-friendly graphical interface to develop models. It also supports access to a powerful knowledge base that allows for sophisticated computer-aided analysis. This tool is intended to provide software developers with a traceable link between the requirements, specification and architectural design phases of the software development process.

Mobile agents are autonomously running programs that are able to take decisions and initiate actions and move from machine to machine. They work on behalf of some users (or other systems). Several agents can cooperate to solve some task or may compete to acquire some resources. In the rest of this section, we will analyse a mobile agent system and describe some of the non-functional requirements, especially security and fault-tolerance. In addition, we describe the different alternatives to reach each of the captured non-functional requirements. The alternatives we list in this work consider three different mobile agent systems described in [7,9,11]. These alternatives will be evaluated using GRL diagrams.

Interoperability requirement:
Interoperability is the ability of the mobile agent to interact with specific systems. In our case, the mobile agent shall have the ability to interface and act in different platforms. In addition, it should be able to recognize agents from other platforms. Furthermore, it is desirable for mobile agents to be managed by different agent systems or platforms to multicast.

Security requirement:
Security is the most important concern hindering the growth of the mobile agent computing paradigm. Mobile agents need to be protected against malicious manipulations at a host computer, in addition, local resources must be adequately protected by a visiting agent. Visiting mobile agents need to be authenticated to ensure proper executions. In the following, we discuss some alternatives that were introduced in [9,11] to deal with safeguarding both agents and visited machines.

a. Protection of host machines: This can be achieved by authenticating the agent’s launcher and to impose certain local restrictions and use access control mechanisms to limit the accessibility to local resources. The RSA public key based protocol can be used to authenticate a legal agent (ie, coming from a recognized platform). Once an agent is authenticated, it is allocated certain limited access to resources.

b. Protection of mobile agents: This can be achieved by mechanisms such as self-authentication, encryption of components, and replication and voting. This requirement intersects with the fault tolerance requirement,
since the system must protect against agent loss. In this case, replication and voting (as described in the fault tolerance requirement) can be used to recover a mobile agent. In the case of a malicious change to the agent’s content (data or state), self-authentication can be used and any discovered inconsistencies can lead to the recovery of a previous valid state or checkpoint of the attacked agent. Also, the content of an agent can be divided into components each of which can be encrypted to ensure that proper agents or machines can only have access to the component’s content. For example, constant data and states can be compartmentalized in one component.

Figure 13 shows the mobile agent security model drawn using GRL and the OME tool.

**Fault Tolerance Requirement:**
Mobile agents and the platforms on which they run must satisfy certain level of fault tolerance that will increase the trustworthiness of the application when abnormal situations occur. Agents travel across uncertain networks. During the life cycle of a mobile agent, a lot of unpredicted situations may occur. Errors may happen on the computing or on the network facilities. It may be necessary to reincarnate an agent if for example it was lost or maliciously destroyed or modified at one of the system computers, or if a reached computer breaks down. In the following sections, we discuss several ideas proposed to implement fault-tolerance in mobile agent systems [10]: primary-backup model, exactly-once mobile agents, sliding window, and replicated agents with voting.

a. **Primary-backup model:** Primary and backup servers are dedicated, and a protocol for checkpointing is used. Once a failure occurs, one of the backups becomes the new primary server.
b. **Exactly-once model:** In some applications, like voting, it is important to ensure that a transaction be executed only once. Protocols for ensuring this requirement must be implemented [10].
c. **Sliding window model:** A backup of an agent is kept and once it is guaranteed that a loss did not occur, the current backup is discarded and a new backup is saved at a different machine [10].
d. **Replicated agents with voting model:** In this model replication and voting are used to deal with the execution of an agent on a faulty processor. Votes have to be authenticated to ensure that a faulty processor does not cast false votes [10].

Figure 14 shows a GRL model of fault tolerance and its alternative methods.

**Portability requirement:**
Portability is the degree to which software running on one platform can easily be converted to run on another. In mobile agent systems, the platform must have full portability. That is why most of the platforms are Java-Based since Java is known for its portability.

**Scalability requirement:**

The scalability of a system tells how many users or agents can be concurrently active at one time. This requirement affects the design and the implementation significantly. To make our platform highly scalable, we need a robust architecture that may involve multiple processes to support concurrent users. Also, designers need to consider the distribution of multiple processes. The architecture of the platform in this case will be more complicated than an architecture supporting a small number of users.

**CONCLUSIONS AND FUTURE WORK**

Unlike functional requirements, non-functional requirements are very hard to elicit by non-expert users. To be elicited properly and completely, several iterations and elicitation meetings with the appropriate stakeholders are needed. Analyst expertise in NFRs is also needed. The graphical modeling of this type of software requirements would be very useful for visually capturing and validating them. The completeness of these captured NFRs is essential for the success of the software system under consideration. In this paper, we have introduced and illustrated the use of the User Requirement Notation (URN) for the formal description of captured NFRs. We have experimented with the OME tool to illustrate the
NFRs of mobile agent-based systems. In the future, we are planning to study: (1) the possible integration of URN with the UML, (2) the use of the GRL description of NFRs as a reference to produce acceptance test plans and to guide the acceptance testing and product validation processes, and (3) the development of easily testable NFRs.

ACKNOWLEDGMENT
We would like to acknowledge the support of the AUS research unit.

REFERENCES