INTERNATIONAL TELECOMMUNICATION UNION

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

Z.150

(02/2003)

SERIES Z: LANGUAGES AND GENERAL SOFTWARE ASPECTS FOR TELECOMMUNICATION SYSTEMS

Formal description techniques (FDT) – User Requirements Notation (URN)

User Requirements Notation (URN) – Language requirements and framework

ITU-T Recommendation Z.150

ITU-T Z-SERIES RECOMMENDATIONS

LANGUAGES AND GENERAL SOFTWARE ASPECTS FOR TELECOMMUNICATION SYSTEMS

FORMAL DESCRIPTION TECHNIQUES (FDT)	
Specification and Description Language (SDL)	Z.100–Z.109
Application of formal description techniques	Z.110–Z.119
Message Sequence Chart (MSC)	Z.120–Z.129
Extended Object Definition Language (eODL)	Z.130–Z.139
Tree and Tabular Combined Notation (TTCN)	Z.140–Z.149
User Requirements Notation (URN)	**Z.150–Z.159**
PROGRAMMING LANGUAGES	
CHILL: The ITU-T high level language	Z.200–Z.209
MAN-MACHINE LANGUAGE	
General principles	Z.300–Z.309
Basic syntax and dialogue procedures	Z.310–Z.319
Extended MML for visual display terminals	Z.320–Z.329
Specification of the man-machine interface	Z.330–Z.349
Data-oriented human-machine interfaces	Z.350–Z.359
Human-computer interfaces for the management of telecommunications networks	Z.360–Z.369
QUALITY	
Quality of telecommunication software	Z.400–Z.409
Quality aspects of protocol-related Recommendations	Z.450–Z.459
METHODS	
Methods for validation and testing	Z.500–Z.519
MIDDLEWARE	
Distributed processing environment	Z.600–Z.609

For further details, please refer to the list of ITU-T Recommendations.

ITU-T Recommendation Z.150

User Requirements Notation (URN) – Language requirements and framework

Summary

Scope-objective

This Recommendation with other Recommendations in the Z.150 series defines URN (User Requirements Notation) for describing user requirement as goals and scenarios in a formal way without any reference to implementation mechanisms and with optional dependency on component specification. Such a notation is needed to capture user requirements prior to any design.

Coverage

URN has concepts for the specification of behaviour, structuring, goals, and non-functional requirements. This Recommendation focuses on language requirements for URN and on providing the context for a requirements engineering framework. Other Recommendations in the Z.150 series define the notation for URN.

Applications

URN is applicable within standards bodies and industry. URN helps to describe and communicate requirements, and to develop reasoning about them. The main applications areas include telecommunications systems and services, but URN is generally suitable for describing most types of reactive systems. The range of applications is from goal modelling and requirements description to high-level design.

Status/Stability

This Recommendation provides the scope and requirements for URN.

The main body of the Recommendation has following attachments:

- Annex A Compliance to this Recommendation;
- Appendix I Requirements engineering activities;
- Appendix II Guidelines for the maintenance of URN;
- Bibliography.

Associated work

This work is associated with languages, notations, and methodological aspects related to other ITU-T Study Group 17 languages.

Source

ITU-T Recommendation Z.150 (2003) was prepared by ITU-T Study Group 17 (2001-2004) and approved under the WTSA Resolution 1 procedure on 13 February 2003.

Keywords

Evaluation, formal specification, functional requirements, goal, graphical notation, hierarchical decomposition, non-functional requirements, requirements engineering activities, scenario, transformation.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications. The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had not received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementors are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database.

CONTENTS

			Page
1	Scope		1
	1.1	Motivation	1
	1.2	Document organization	2
2	References		3
3	Definitions		3
4	Abbreviations and acronyms		5
5	Scope of URN		5
	5.1	What is URN?	5
	5.2	What is URN-NFR?	6
	5.3	Why goal-oriented requirements engineering?	7
	5.4	What is URN-FR?	8
	5.5	Intended usage	8
6	Language requirements for URN-NFR		9
	6.1	Expressing tentative, ill-defined and ambiguous requirements	9
	6.2	Clarifying, exploring, and satisficeing goals and requirements	9
	6.3	Expressing and evaluating measurable goals and NFRs	10
	6.4	Argumentation	10
	6.5	Linking high-level business goals to system requirements	10
	6.6	Multiple stakeholders, conflict resolution and negotiation support	10
	6.7	Requirements prioritization	10
	6.8	Requirements creep and churn and other evolutionary forces	10
	6.9	Integrated treatment of functional and non-functional requirements	10
	6.10	Multiple rounds of commitment	11
	6.11	Life-cycle support	11
	6.12	Traceability	11
	6.13	Ease of use and precision	11
	6.14	Modularity	11
	6.15	Reusable requirements	12
7	Language requirements for URN-FR		12
	7.1	System trigger and termination conditions	12
	7.2	System operations and responses	12
	7.3	Complex and lengthy behaviour	13
	7.4	Relationships among scenarios	14
	7.5	Component definition	14
	7.6	Environment specification	14

Page

8 Other language requirements for URN 15
8.1 Requirements traceability 15
8.2 Requirements test case specification 16
8.3 Performance analysis of requirements 17
8.4 Change management 17
8.5 Concrete representations 17
8.6 Usability 18

9 Language requirements summary 18
9.1 Requirements table format 18
9.2 URN requirements table 18

Annex A – Compliance to this Recommendation 21

Appendix I – Requirements engineering activities 22

Appendix II – Guidelines for the maintenance of URN 24
II.1 Maintenance of URN 24
II.2 Rules for maintenance 25
II.3 Change request procedure 25

Bibliography 27

ITU-T Recommendation Z.150

User Requirements Notation (URN) – Language requirements and framework

1 Scope

This Recommendation provides motivation, scope and language requirements for the ITU-T User Requirements Notation. The specification of compliant notations belongs to other Recommendations.

The text of this clause is not normative.

1.1 Motivation

A notation is needed that can describe user requirements, goals and scenarios without any reference to specific inter-component communication facilities or system components and their states, but at the same time can capture the user requirements prior to design. The focus during the requirements specification stage is on behaviour and on quality attributes. The notation can also be used during the high-level design phase when activities or *responsibilities* specified in the scenarios are allocated to components. Scenario specification without sub-system component reference would facilitate reusability of scenarios across a wide range of architectures. The ability of the notation to straddle requirements specification and high-level design will facilitate negotiations between stakeholders and implementers.

Before URN was recommended, there was an increasing demand for non-static protocols with policy-driven negotiation using dynamic entities. Agent-based systems are examples of systems that require such policy-driven mechanisms. When specifying this kind of protocol, it is not possible to make an early commitment to messages and components at the requirements capture phase.

There is also the need for detection and avoidance of undesirable interactions between features or services. Older techniques require large investment in terms of messages and components that need to be checked for interactions. Using the notation specified in this Recommendation can provide insights at the requirements level and enable designers to reason about feature interactions early in the design process.

It is also important to deal with business objectives, goals, and non-functional requirements (NFRs) in a more systematic manner during requirements analysis and design. NFRs are requirements such as stringent performance constraints, systems operational costs, reliability, maintainability, portability, interoperability, robustness, and the like. In software development practice, many NFRs are stated only informally, making them difficult to analyse, specify and enforce during software development and to be validated by the user once the final system has been built. Goals and NFRs, however, do play a crucial role during system development, serving as selection criteria for choosing among alternatives during requirements analysis, for example, determining where the system boundaries should be and what functional requirements to include in the system.

Many of the alternative approaches to deal with NFRs originated from the technical work related to quality metrics. Such approaches attempt to quantify NFRs and then measure to what extent an existing system or parts of it meet the desired non-functional requirements. Useful metrics exist for NFRs such as performance, reliability, software complexity, and development process maturity. Other approaches, which recognise that many NFRs are often difficult, if not impossible, to quantify, use qualitative-oriented methods such as architectural change scenarios or combinations of both qualitative and quantitative methods to evaluate systems. These approaches, however, assume an already existing software system (or parts thereof) that is evaluated for its NFR properties. They do not assist in the specification of NFRs prior to building the system, nor do they provide support during the analysis and design of systems. The notation proposed herein deals with NFRs and goals

during the process of requirements analysis and system design; it allows for the expression of conflict between goals, of decisions that resolve conflicts and of the rationale for the trade-off decisions.

The URN is defined to have the following capabilities:

a) describe scenarios as first class entities without requiring reference to system subcomponents, specific inter-component communication facilities, or subcomponent 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 detection and avoidance of undesirable interactions between features;

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

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

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

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

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

l) provide facilities to connect URN elements to external requirements objects;

m) provide facilities to manage evolving requirements.

The previous methods that used informal natural language for capturing requirements can leave too much open for interpretation and can contain invalid logic. Manual methods are used to validate these specifications with the result that defects are sometimes not caught until the implementation phase. Studies of software development have clearly shown that the earlier defects are detected, the less costly they are to repair. Standardization of a notation supporting requirements engineering activities aims to make it easier to detect more defects at the requirements definition stage.

These same informal methods have also proven less than satisfactory for negotiating relative priorities of different business objectives and, in general, for managing trade-offs in the domain of non-functional requirements. The end result can be the delivery of a product to market that does not satisfy customers and does not meet business objectives. Standardization of a notation for requirements engineering activities aims to make it easier to define a product that balances stakeholder objectives and satisfies customer expectations.

Standardization of a formally defined notation used for capturing user requirements is a move to make the practice of this activity more rigorous and predictable and the results yielded by this activity clearer, more consistent, correct, and complete. These results should lead to a reduction of development costs, earlier delivery of product to market, and increased customer satisfaction.

1.2 Document organization

This Recommendation defines the language requirements for URN regarding both functional and non-functional aspects.

Clauses 2 to 4 cover background information on references, definitions, and abbreviations and acronyms.

Clause 5 details the scope of this Recommendation by providing an overview of the requirements engineering activity that uses the URN, both from functional and non-functional perspectives.

Clauses 6, 7, and 8 provide detailed language requirements for URN-NFR, URN-FR, and other areas.

Clause 9 summarizes the list of language requirements that the URN must fulfil.

Annex A presents a conformance statement for notations and tools in relation to this Recommendation.

Two appendices are defined:

- Appendix I introduces general activities in requirements engineering.
- Appendix II describes guidelines for the maintenance of URN.

A bibliography can be found at the end of this Recommendation.

2 References

The following ITU-T Recommendations and other references contain provisions, which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

3 Definitions

This Recommendation defines the following terms:

3.1 additional information: The information stakeholders believe is important for designers and implementers to know but is not considered when validating a requirements specification.

3.2 behaviour: The sequence of actions with stimulus and responses aspects performed by a system that may change its state. 2.10/Sup. 1 to Z.100.

3.3 component: A generic and abstract entity that can represent software entities (e.g. objects, processes, databases, or servers) as well as non-software entities (e.g. actors or hardware).

3.4 dynamic refinement: A modelling mechanism that addresses issues related to structure and behaviour evolving at run time.

3.5 evaluation: The process used to determine the *satisfiability* (see 3.19) or *satisficeability* (see 3.20) of a solution with respect to its goal(s).

3.6 executability: An attribute of a model that can be interpreted or compiled and run. Executability applies mainly to functional models such as the one defined using URN-FR.

3.7 feature interaction: A desirable or undesirable interaction between two or more features, functionalities, services, policies, or scenarios. With respect to scenarios, a feature interaction is the set of conditions under which the execution of one scenario is affected by the execution of another. Undesirable feature interactions are also called *conflicts*.

3.8 functional requirement: A requirement (see 3.16) defining functions of the system under development.

3.9 goal: An objective or concern used to discover and evaluate functional and non-functional requirements.

3.10 high-level design: A design document describing system functionalities, the system architecture, and scenarios.

3.11 language requirement: A language requirement, as distinct from requirement (see 3.16), is a required characteristic of a language.

3.12 non-functional requirement: A requirement (see 3.16) characterizing a system property such as expected performance, robustness, usability, maintainability, etc. Non-functional requirements capture business goals/objectives and product quality attributes.

3.13 postcondition: A postcondition expresses the condition following (successful) execution of a given operation or scenario. A postcondition normally expresses a relationship between the output variables in terms of the input variables. Where input variables may also be output variables, the relationship is defined in terms of initial system state and final system state.

3.14 precondition: A precondition expresses the conditions for which an operation or scenario is defined (i.e. if the conditions are not satisfied, then the result of the operation is not defined). A precondition normally expresses a relationship between input variables, or the system state prior to execution.

3.15 quality attribute: A non-functional requirement that relates to systems or products rather than to a business objective/goal.

3.16 requirement: A requirement, as distinct from a language requirement (see 3.11) and a user requirement (see 3.25), is an expression of ideas to be embodied in the system or application under development.

3.17 requirements engineering: The activity of development, elicitation, specification, and analysis of the stakeholder requirements, which are to be met by systems.

3.18 responsibility: A scenario activity representing something to be performed (operation, action, task, function, etc.). A responsibility can potentially be associated or allocated to a component.

3.19 satisfiability: The ability to determine whether a goal can be satisfied according to some strict criteria. For example, a performance goal may state that a system must generate a response to a certain event within a specified time, and it is possible to measure the time lapse between the occurrence of the event and the occurrence of the response.

3.20 satisficeability: The ability to determine whether a goal can be satisfied within acceptable limits. For "satisfied within acceptable limits" substitute "satisficed". For example, a security goal may state that a system cannot be accessed by unauthorized persons. Simple measure may ensure that most people will be unable to access the system. Ever more complex measures may ensure that a wider range of people will be unable to access the system. The law of diminishing returns applies here. It is impossible to guarantee that no unauthorized person can access the system.

3.21 scenario: A partial description of system usage defined as a set of partially-ordered responsibilities a system performs to transform inputs to outputs while satisfying preconditions and postconditions.

3.22 stakeholder: An individual or organization interested in the success of a product or system. Stakeholders include customers, users, developers, engineers, managers, manufacturers, testers, and so on.

3.23 system: A generic term describing a combination of components collaborating among themselves and with the external environment. A system can also be a new system or a system extension.

3.24 specification: A clear and accurate description of characteristics of a product or procedures. A specification is formal when it is written using a formal language. Requirements specifications focus on the problem domain (the "what"), whereas design specifications focus on the description of the design in conformance with the requirements specification (the "how").

3.25 user requirement: A desired goal or function that a user or other stakeholders expect the system to achieve. A user requirement may or may not be a requirement (see 3.16).

4 Abbreviations and acronyms

This Recommendation uses the following abbreviations:

ASN.1	Abstract Syntax Notation One
COTS	Commercial-Off-The-Shelf
FR	Functional Requirements
GRL	Goal-oriented Requirement Language
ISO	International Organization for Standardization
ITU	International Telecommunication Union
MOF	Meta-Object Facility
MSC	Message Sequence Chart
NFR	Non-Functional Requirements
OMG	Object Management Group
RE	Requirements Engineering
SDL	Specification and Description Language
TTCN	Tree and Tabular Combined Notation
UCM	Use Case Map
UML	Unified Modelling Language
URN	User Requirements Notation
URN-FR	User Requirements Notation – Functional Requirements
URN-NFR	User Requirements Notation – Non-Functional Requirements
W3C	World Wide Web Consortium
XML	eXtensible Markup Language

5 Scope of URN

5.1 What is URN?

The User Requirements Notation (URN) shall allow software engineers to:

- specify or discover requirements for a proposed system or an evolving system, and
- review such requirements for correctness and completeness.

The URN is intended for use in requirements descriptions in specifications developed by national and international standards organizations. In the ITU-T, requirements descriptions are often called Stage 1 descriptions (e.g. in I.130 and Q.65). The URN is also intended for use by commercial organizations developing requirements specifications for new products and product extensions; these specifications are not necessarily governed by standards.

The URN is used to construct functional and non-functional requirements models. Like most notations in ITU-T's family of languages, the primary notations of URN shall be graphical, because graphical presentations are often compact and easy to understand. The Recommendations for URN specify a non-functional requirements URN (URN-NFR) and a functional requirements URN (URN-FR) as well as a set of relationships between the URN-NFR and the URN-FR.

The URN is viewed as complementary to notations such as Message Sequence Charts (MSC), the Specification and Description Language (SDL), TTCN-3, and the Unified Modelling Language (UML). Information contained in URN models could possibly be linked to these other languages.

5.2 What is URN-NFR?

Business objectives and product quality attributes are modelled using the URN-NFR. Software engineers use the URN-NFR model to identify and negotiate trade-offs among competing objectives and quality attributes. An outcome of this exercise is a set of technology and implementation choices that reflect these trade-offs. The outcome of the URN-NFR modelling exercise sets the context for the URN-FR modelling exercise.

Non-functional requirements (NFRs, also called quality requirements) are global requirements on the software system, its development, deployment, maintenance and evolution and operational processes, such as the systems operational costs, performance, reliability, maintainability, portability, robustness, and the like. NFRs may originate from objectives related to the business organization (sometimes called consumer-oriented quality requirements) but also from requirements on the software system, its development environment and process (sometimes called technically-oriented quality requirements). Errors of omission or commission in laying down and properly taking into account such requirements are generally acknowledged to be among the most expensive and difficult to correct once a software system has been implemented, and have direct impact on the success of the software system. NFRs are difficult to specify and deal with since they often do not have precise definitions, and do not have clear-cut criteria of when they have been satisfied. For example, it is difficult to know how to specify the extensibility requirements of a system, and when a system has met that requirement. In addition NFRs often conflict with each other; for example, providing for extensibility by layered system architecture that may adversely affect the performance of the system.

A URN that addresses NFRs should therefore address such requirements up-front during analysis, and allow such NFRs to be expressed, even if they are ill-defined and tentative. The URN should then support further refinement and clarification of these ill-defined and tentative NFRs and, if possible, allow their quantification to be expressed. It should allow exposing and modelling conflicts among NFRs during analysis, and provide abilities to evaluate trade-offs among conflicting requirements, and expose and facilitate negotiation between the involved stakeholders. A URN should allow relating NFRs to potential alternative elements within the functional requirements specification such that NFRs can be used as selection criteria among them.

Sometimes the precise meaning and the degree of achievement of NFRs may become clear only during the design stage, or even during implementation of the software system. A URN should therefore also support and guide the process of refining and clarifying NFRs to serve as selection criteria during these later phases of the software lifecycle. This objective for URN emphasises the need for linking NFRs to all phases of the software development life cycle, since the degree of achievement of NFRs may be affected by decisions during all phases. A URN should provide support for managing, tracing, validating and evolving NFRs, as well as functional requirements (FRs), during the whole development life cycle.

In order to provide for such a comprehensive life-cycle support for NFRs, two objectives for a URN are introduced:

- To explicitly support goal-based modelling and reasoning for both functional and non-functional requirements as a means for relating higher-level business and organizational objectives to the functional and non-functional aspects of the intended system.
- Since goals that express NFRs are initially often ill-defined, tentative and ambiguous, the URN should provide support for the process of refining and clarifying such goals to be more precise. This process support should be part of an engineering process for requirements, which acknowledges that establishing requirements is a decision making process with many interrelated activities, and which has relevance during the whole life cycle of a software system.

5.3 Why goal-oriented requirements engineering?

Broadly speaking, many user requirements are often first stated as desired goals that stakeholders wish to achieve. Providing for an immediate way of expressing such goals, rather than activities and entities that support achieving such goals, allows reasoning about alternative ways to achieve stakeholder goals. Goal-based modelling can therefore be used for the discovery and treatment of NFRs as well as the development of functional requirements.

Goals that relate to functional requirements enable expressing and reasoning about functional alternatives for which clear criteria exist and allows an evaluation of the system as to whether or not it provides for the desired functionality. Goals that relate to NFRs enable expressing a softer notion of achievement. It is said that an NFR-related goal is satisficed[1] when there is sufficient positive and little negative evidence for their achievement, and that they are unsatisficeable when there is sufficient negative evidence and little positive support for their satisfiability.

Unlike functional goals where automatic reasoning can (to some extent) establish whether they were fully achieved or not, NFR-related goals may need humans to intervene in cases when only weak or conflicting evidence is provided. This calls for evaluation and decision-making that needs to be made interactively by engineers not only during requirements analysis, but also during the design and implementation process. During this analysis process, both goals denoting functional and non-functional requirements are stated up-front, and refined to produce goal graphs. During this process, the NFR-related goals are used as selection criteria among alternative functional requirements that achieve the functional goals of the stakeholders in general, and of the system in particular. During design and implementation, goal graphs are further refined and linked to architectural, detailed design and implementation functions and structures. Functional requirements-related goals provide for focal points of alternative design and implementation choices, while NFR-related goals provide for the selection criteria that are considered for each potential functional alternative. Selecting one branch of a goal graph for further refinement implies a design choice.

A URN that provides for goal modelling allows the elements of the software system requirement specification to be linked to their rationales which are to be found in the system's environment. This allows the capture of "why" elements of the intended specifications "where" proposed, and reasoning about whether the proposed specification is sufficient for achieving the higher-level objectives of the system and the organisation. Using goals also allows the requirement process to be guided in exploring and evaluating alternative system specification, exposing conflicting interests among stakeholders, and aiding the management and the evolution of requirements, when objectives change over time. Including support for the process of requirements engineering in the URN, allows reasoning about high-level objectives while they are still informal and in need of clarification, and provides support for refining those objectives towards a more precise

---

[1] We use here the notion that a goal can rarely be said to be satisfied. Goal satisficing suggests that the used solution is expected to satisfy the requirement within acceptable limits.

specification. During this refinement process, alternatives may be expressed, evaluated, justified or rejected both in terms of NFRs, and in terms of pertinent domain knowledge, until the requirements engineers arrive at a satisfactory specification.

5.4 What is URN-FR?

A URN-FR model shall be an abstract representation of the behaviour of a proposed system and its environment. The stakeholder can use the URN-FR to specify scenarios; that is, sequences of responsibilities that must be executed to transform inputs to outputs while satisfying relevant preconditions and postconditions. Scenarios are also called maps because they are thought of as roadmaps connecting inputs to outputs. The notation shall allow the user to specify relationships between scenarios. It is possible to use the notation to specify abstract architectural components and allocate responsibilities to them but it is not necessary to do so. Responsibilities are connected by causality flows. A causality flow is the assertion of a causal connection between responsibilities. That is, the execution of this sequence of responsibilities in some fashion causes or enables the execution of a subsequent responsibility. The intent behind the notation is to leave the specification of detailed interactions between responsibilities to more concrete notations such as Message Sequence Charts. The objective is to allow software engineers to express their domain knowledge in an intelligible way without letting detailed design considerations get in the way.

A URN-FR shall facilitate negotiations between software engineers and implementors. The aim is to use the URN-FR model to discover as many questions of policy as possible so that the stakeholders can rule on these questions prior to implementation. The URN-FR notation can be used for high-level design as well as for functional requirement specification. Developers can begin the high-level design phase by iterating the model constructed by the requirements engineers and by considering additional architectural concerns. If, by doing so, the developers discover new requirements, they can discuss the matter with the requirements engineers by referring to the model.

When URN-FR is used, a model is evaluated for clarity, consistency, correctness, and completeness first by visual inspection, but also by formal and algorithmic evaluations.

5.5 Intended usage

A primary purpose of URN is to facilitate the communicating of requirements among pertinent stakeholders prior and during the system development life cycle in general, and during requirements analysis in particular. This includes communicating among stakeholders such as clients, standards bodies, business analysts, intended users of the system, architects, designers, testers, implementors and the like. If we take standards bodies as example, it will show the significance of utilizing URN for these bodies. Industry standards are dynamic in nature, continuously evolving to meet stakeholders' requirements with ever-shorter intervals for standards development. The current timelines at which a new version of the specification is to be completed to the needed level of precision, quality and completeness cannot be accommodated using existing specification techniques. A key assumption is that future standards work must apply techniques that can be automated or semi-automated. The use of formal documentation techniques using tools will shorten the standards development cycle, introduce a formal test methodology, and assist in rapid validation and verification, harmonization, and evolution of the standards.

In addition, during the early phases of requirements elicitation (and high-level design), URN should support an exploratory mode of work, in which principal alternatives are considered and where minute details are omitted and left for future elaboration. Such exploratory work is often done in conjunction with non-technical stakeholders in order to explore feasible directions towards system specifications (and design). Having achieved agreement on principal directions the URN should support detailed specifications that may then be undertaken in conjunction with more technically oriented stakeholders, and that allow for validating of requirements in a formal manner.

These considerations give rise to three objectives on URN:

- The ability to provide for informal (or semi-formal) requirements descriptions that focus on coarse-grained or abstract goals, behaviours, and structures of the intended system. Such a description would facilitate the exploring of alternatives and would omit details not pertinent to such reasoning. This would facilitate the communicating of requirements among non-technically oriented stakeholders.
- The ability to provide for formal requirements descriptions that do focus on detailed goals, behaviours, and structures. This would facilitate the communicating of requirements among technically oriented stakeholders.
- The ability to support the transition from informal (or semi-formal) descriptions to formal ones, together with the ability to reason about and explore alternative "formalizations" during that transition. This would provide the basis for communicating among both non-technically and technically oriented stakeholders.

The URN definition acknowledges the abilities of existing notations (such as SDL, MSC, and UML) but positions their abilities as belonging to the more formal and detailed requirements description approaches. The URN definition addresses earlier phases during requirement analysis when a more exploratory, coarse-grained and informal approach is more suitable, that does not overwhelm the stakeholders with irrelevant details and provides support for exploring alternatives.

6 Language requirements for URN-NFR

This clause describes the language requirements for a URN that deals mainly with goals and NFRs.

6.1 Expressing tentative, ill-defined and ambiguous requirements

A URN that deals with goals and NFRs shall provide the ability to express tentative and ill-defined requirements that are difficult, if not impossible, to formalise, and where no clear criteria exist for their achievement during requirements analysis, or during design and implementation. Expressing such requirements is of particular importance during the early phase of requirements elicitation when the understanding that stakeholders have of their objectives is still vague, tentative, ill-defined, ambiguous, and in need of clarification.

6.2 Clarifying, exploring, and satisficeing goals and requirements

When clarifying and "satisficeing" tentative, ill-defined and ambiguous requirements, and when exploring alternatives (here called "loose" requirements), a URN-NFR shall provide support for disambiguating such requirements in a systematic manner. This support shall be provided throughout iterative refinements, during requirement elicitation and during requirement analysis. It shall provide support for exploring alternative meanings for loose requirements. In addition, since no clear-cut criteria exist for when such loose requirements are achieved, there is a need to provide a more flexible, and fine-grained notion of achievement, such as sufficiently achieved, some contribution towards achievement, some negative evidence against achievement, and insufficiently achieved. Alternative solutions would achieve such loose requirements with different degrees of satisfaction. Accordingly, URN-NFR shall provide support for expressing different degrees of achievement. Interactive, semi-automatic (that is, not completely automated) analysis facilities, which "know" when to refer back to the analyst for a subjective opinion during evaluation, can provide support for evaluating how well solutions satisfy such requirements.

6.3 Expressing and evaluating measurable goals and NFRs

The URN-NFR shall support expressing goals and NFRs that do have clear metrics and measurements for their achievement, and incorporate such goals and NFRs in the reasoning and evaluation process. A particular benefit of providing support for both qualitative and quantitative goals and NFRs is the ability to show how one is traded off for the other. Key issues in many systems include performance requirements that need support for their evaluation, together with the ability to document how and why they are traded off for other desired quality requirements of the system.

6.4 Argumentation

A URN-NFR shall support the recording of arguments for or against each iterative refinement. Such arguments would then be taken into account when evaluating solutions for their degree of how well they achieve requirements.

6.5 Linking high-level business goals to system requirements

Since the tentative and ill-defined NFRs are often high-level organizational and system objectives, a URN-NFR shall support linking such high-level concepts to the more concrete elements of the requirements specification. Such links are able to provide an understanding of how intended software systems in fact contribute to the high-level, strategic directions, an organization wishes to take.

6.6 Multiple stakeholders, conflict resolution and negotiation support

Since requirements may originate from multiple stakeholders, a URN-NFR shall be able to express the origin of each requirement, and whether the different interests of stakeholders bring synergy or conflict with each other.

6.7 Requirements prioritization

A URN shall support the prioritization of requirements in general and for stakeholders in particular. This supports the negotiation process when conflicting requirements arise. Prioritization also allows expression of the importance of requirements, how they might change over time, and in what way this may change the focal point of the system development effort.

6.8 Requirements creep and churn and other evolutionary forces

A URN shall support the ability to detect evolution in requirements between the time they are formulated and the time the product is delivered, in particular when requirements are added or changed (requirement creep). It shall also support frequent modification of the same requirements or their priorities (requirement churn). Both have an impact on the requirements specification, and how changes in the requirements specification affect the rest of the development process.

6.9 Integrated treatment of functional and non-functional requirements

A URN shall enable dealing with both functional and non-functional requirements concurrently. In particular a URN-NFR needs to express in what way NFRs may serve as selection criteria when choosing among alternative functional requirements, and for expressing constraints when wishing to achieve functional requirements during design.

6.10 Multiple rounds of commitment

Moving from high-level objectives to system requirements may need multiple rounds of decision-making and commitment by stakeholders. Each new round of decision-making is based on previously adopted decisions that structure the decision space by focusing on certain alternatives and excluding others. During the course of requirements elicitation and analysis, new requirements may be introduced that impact existing requirements and commitments. A URN-NFR shall, therefore, provide support for multiple rounds or layers of decision-making, where each layer proceeds from commitment points of previous layers.

6.11 Life-cycle support

Requirements and their management are relevant during all phases of system development. One reason is requirements creep. Another is the need for requirements traceability as discussed below. Yet another reason is the complexity of the development process itself. System development typically does not proceed in the neat fashion suggested by the waterfall model. Several development cycles can proceed in parallel. Even within one development cycle feedback, loops exist that can trigger re-engineering and a review of commitments to requirements. For all of these reasons, URN shall support requirements management during all phases.

6.12 Traceability

The URN-NFR shall allow expressing requirements as ill defined and tentative at the beginning and provide support for refining those requirements to a more precise specification. It shall also support using requirements to guide the decision making process during the forward engineering of design and implementation. The design and implementation processes can cause evolution in the understanding of requirements and possibly trigger reformulations of, or even commitment to, particular requirements. The system developers must be aware of when their activities impact requirements and be ready to go back to stakeholders with issues and must ensure that the requirements specification remains consistent with the design and implementation of the system. By doing so, the developers will ensure that the requirements specification plays its proper role in system compliance testing. To accomplish these objectives requires that URN-NFR specifications be "connectable" to other development process artefacts.

6.13 Ease of use and precision

A URN is used by many different stakeholders during the requirements specification and development processes. For some stakeholders, ease of use and comprehensibility are paramount, while for others, precision in expressing requirements is of greater importance. A URN should provide support for both types of URN users through supporting degrees of formality in its language and by making clear how a user of a URN can refine from informal expressions of requirements to more formal ones. One focal point should be ease of use for practitioners and comprehensibility for customers and intended users of the system, while another focal point should be the ability to specify requirements more precisely for developers and testers.

6.14 Modularity

A URN-NFR shall support the modular description of goal and NFR models. This will permit the hierarchical decomposition of large sets of goals and NFRs and will improve overall manageability and scalability of complex models.

6.15 Reusable requirements

A URN-NFR should support the reuse of parts of requirements specifications, which are known to express certain objectives, when such or similar objectives recur in other projects. Such a URN-NFR would provide facilities to capture, structure and reuse knowledge related to recurring requirements. Knowledge about achieving functional and, in particular, non-functional objectives would be stored in knowledge catalogues together with applicability conditions stating under what circumstances the knowledge can be reused. Such knowledge could potentially accelerate the requirement engineering process for particular projects.

7 Language requirements for URN-FR

This clause describes the language requirements for a URN that deals mainly with functional requirements.

7.1 System trigger and termination conditions

A functional requirements specification, even if it contains nothing else, contains a mapping of input events and preconditions to output events and postconditions. Preconditions and postconditions relate to both environmental states and target system states. The environmental set of preconditions and postconditions are kept separate from the system set by the fact that one set of scenarios models the environment and one set models the system. The URN-FR is used to model both the environment and the target system. The start points of the system scenarios are connected to the end points of scenarios occurring in the environment, and the end points of the system scenarios are connected to the start points of scenarios occurring in the environment.

The URN-FR shall allow stakeholders to distinguish the many mappings of input events and preconditions to output events and postconditions for a particular system in whatever degree of detail seems appropriate.

A URN-FR shall support notation for specifying:

- the set of input events at a scenario start point;
- the set of output events at a scenario end point;
- preconditions at scenario start points;
- postconditions at scenario end points;
- input sources, that is, whether the sources are human or machine;
- output sinks, that is, whether the sinks are human or machine;

The URN-FR shall define a data model so that preconditions, input events, postconditions and output events can be formally defined and managed.

7.2 System operations and responses

A responsibility is an activity representing something to be performed. A URN-FR shall provide means to define responsibilities and to reference them.

The URN-FR shall allow users to specify system operations and responses as a causal flow of responsibilities. The execution of a responsibility is said to cause the execution of a subsequent responsibility. Inter-responsibility communication is not specified.

A system response is what the system does to transform the input events, under some preconditions, into output events and then satisfy postconditions. Given the many possible mappings between input events and preconditions on the one hand, and output events and postconditions on the other, how to manage the specification of system responses for each of these mappings becomes an issue.

One candidate solution to this issue is to group scenarios according to event classes. The event classification is based on common processes and criteria of relatedness. An example of an event class is that of bit patterns on a receiving link where a synchronous data protocol is being used. The system response may differ somewhat based on which event in the class is received. To express this difference, notation for condition-based decision-making (branching) must be used. The handling of preconditions also requires branching. Preconditions express a system state. For example, the system may be in an operational state relative to a particular event class when it receives the event, or it may be out of service. Branching is used to express the different response the system makes depending on its state. Branching is also called OR-forking. The URN-FR shall define a data model and expression evaluator so that conditions on OR-forks can be formally expressed.

Another name for a candidate URN-FR scenario specification is a map because in its graphical form it looks like a geographical road map. The flows of responsibilities are paths.

Events in the same class may be handled in much the same way except for slight differences. The map must show where the common processing segments are as well as where the branching segments are. It is possible that after a branch, the system handling for two events may again be the same for a while. The notation shall be capable of expressing this situation and does so using an OR-join.

The notation shall be able to express parallelizm when specifying the handling of an event. For example, the detection of a loss of signal on a receiving link causes two parallel actions to be taken. The first action is to send an alarm out on the transmit link to the far end, and the second action is to send an alarm to the human user interface.

The notation shall be able to express synchronization when specifying the handling of an event. For example, some bank vaults can only be opened when two people physically out of touch have inserted and turned their keys. The system waits until both events have occurred before proceeding. The notation shall be able to express a wait-forever condition as well as a timed wait with action attendant on a timeout. Synchronization can be within a scenario or between scenarios.

The notation shall be able to specify repetitive action. Collecting digits during a call-set-up is a classic example of a repetitive action that can be expressed as a loop. The URN-FR shall define a data model and expression evaluator so that conditions on loops can be formally expressed.

7.3 Complex and lengthy behaviour

The notation shall allow the user to specify complex and lengthy system responses in a comprehensible way. One way to support comprehensibility is to support abstraction, that is, hide irrelevant detail. Another way is to support hierarchical decomposition of the scenario specifications.

The notation shall support hierarchical decomposition of scenarios. A subscenario container replaces a sequence of responsibilities in a higher-level scenario. The replaced sequences (subscenarios) are represented in a lower level scenario. A subscenario shall be similar in form to a scenario, that is with trigger symbols and termination symbols. The notation shall support the specification of preconditions at the entry points to a subscenario, as well as postconditions at the exit points from a subscenario.

The notation shall distinguish between cases: a static container with only one subscenario, and a dynamic container if two or more subscenarios are defined. In the latter case, a selection policy (related preconditions) determines which of the alternate subscenarios executes at run-time.

Containers and subscenarios can be used to encapsulate behaviour that is found in many places within one scenario or across scenarios.

7.4 Relationships among scenarios

A URN-FR shall support the description of individual scenarios as well as relationships among scenarios. Several forms of relationships exist, many of which are discussed in 7.2 and 7.3. One form of relationship is grouping a set of scenarios that deal with a class of events into a single specification. Another form of relationship is synchronization among scenarios. A third form of relationship is the connection of a subscenario to a parent scenario in a hierarchical decomposition.

When a URN-FR specification becomes complex, with many relationships among scenarios, it becomes essential to be able to recover individual scenarios which can be used to understand particular behaviour or situations. A URN-FR shall include a mechanizm, based on the data models discussed in 7.1 and 7.2, to extract individual scenarios from a group or integrated set of scenarios. Individual scenarios shall be able to express sequence and concurrency. Individual scenarios could then be transformed into other representations more suitable in later stages of the development process (e.g., MSC for design, TTCN for testing, etc.).

The URN-FR shall allow the user to express desirable feature interactions and discover undesirable ones. For example, under certain conditions, a particular service may receive priority treatment, causing the interruption and delay of a lower priority service under way.

7.5 Component definition

A URN-FR shall allow the user to specify scenarios without reference to components as well as with reference to them. The URN-FR can thus be used in situations where no component architecture has yet been defined, and where there is a desire to put no architectural constraints on implementors. It can also be used where a component architecture has been defined, and the activity is to define requirements for system evolution.

Component definition internal to the system is more appropriate to high-level design than to requirements specification because it involves allocation of responsibilities to components. Allocation of responsibilities to components is a high-level design activity, and many criteria are applied to determine a good architecture. Nevertheless, software engineers may feel more comfortable if they can reference entities in the specification. These entities should be considered abstract, functional entities and not instructions to implementors on responsibility allocation unless the entities are Commercial-Off-The-Shelf (COTS) components.

Component definition is appropriate when the system environment is specified in terms of existing components, and the functional model encompasses both the existing components and the new system.

In general, the functional model focuses on behaviour, and component definition is left to the high-level design phase.

Components shall also support dynamic aspects used to capture roles in an organization (which can be filled at different times by different people), to represent mobile entities, or to create/delete entities dynamically. Such aspects are particularly relevant to agent systems and other advanced object-oriented applications.

7.6 Environment specification

The stakeholders shall be able to specify the behaviour of the system's environment in URN-FR as well as the system. All of the capabilities of the URN-FR that can be used to model the system shall be available for modelling the environment. The environment model then becomes the driver for the system model and vice versa.

Component definition can come into play here. The system can be identified as a single component and is connected to existing systems modelled as black box components in the new system's environment. The value in this level of component definition is that it clarifies what behaviour in the overall scenario specification belongs to the new system, and what belongs to the system's environment.

A URN-FR shall also contain a special type of component used to denote actors external to the system under design, in its environment.

8 Other language requirements for URN

8.1 Requirements traceability

In a software engineering process, traceability is the property that defines how elements contained in different system models relate to each other. This allows linking model elements that are semantically related.

In the specific context of the URN, requirement traceability is of particular importance. Requirement traceability is the property that allows linking system artefacts defined in the different models as well as design decisions to requirements.

In a software development process, the definition of requirement traceability relations is important for many reasons:

- *To evaluate requirement coverage*. An important question that a developer must be able to answer is: "Are all requirements addressed in the current version of the system?" In order to answer this question, one must be able to determine precisely the set of requirements that are referenced in the different system models. If requirements traceability relations have been maintained during the whole design process, this question can be easily answered. Moreover, the set of requirements that have not yet been addressed can then be automatically determined.
- *To evaluate the impact of requirements modifications*. Another important question that a developer must be able to answer is: "What are the model elements that are related to a specific requirement?" This question must often be answered in the case where modifications are made to requirements. The existence of traceability relationships allows evaluating the impact of modifications on the different models, and making the changes to the affected models in a consistent manner. Thus, if a modification is made to a requirement, say R1, designers can evaluate the impact of the modification by analyzing the elements of the different models that are linked to R1.
- *To allow requirements-based testing***.** In a scenario-driven (or use case-driven) process, requirements are associated with specific scenarios. Therefore, in order to test that the current implementation of a system is correct with respect to a specific requirement, one must first determine the set of scenarios that are related to the requirement. Then, the set of scenarios can be executed, and the result of the execution can be analyzed to see if the requirement is correctly addressed or not. For this purpose, it is important to establish traceability relations between elements of the scenario descriptions in the URN and stakeholder requirements.
- *To allow the identification of conflicting requirements*. The causes of system errors are various. One important cause of errors is conflicting requirements. This type of error is often difficult to prevent and is only discovered late in the development process. For this reason, when an error is found in the system, it is important to be able to trace it back to the different models, and ultimately to requirements, and see where the error has been introduced. If the error comes from conflicting requirements, then these requirements can be precisely identified.

- *To reduce maintenance efforts*. An important part of the cost of system maintenance is related to the evaluation (or the non-evaluation) of the impact of modifications. If one can determine precisely the set of model elements that can be impacted by the modification of a specific requirement (or model element), then the cost of modification could be significantly reduced.
- *To preserve the rationale for design decisions*. Knowing the original reasons for design decisions helps engineers who maintain or enhance systems to evaluate whether implementation should be changed in the light of new circumstances. Such re-engineering of implementations may be essential to keeping a product vital and competitive in the marketplace. The URN-NFR notation should provide the ability to present the rationales for a specific choice together with the arguments for it, in a concise and readable form.

URN shall support both *backward traceability* relations from the URN, and more specifically URN elements, to their source (documents, stakeholders requirements, problem domain analysis, etc.), and *forward traceability* relations from the URN to the other models used in the development process. This forward traceability is preferably achieved by backward traceability relations from the other models to the URN. If traceability exists between the other models and implementation, the existence of these two types of traceability relations would transitively ensure a complete traceability from implementation to the source of requirements.

Since requirements are open-ended, and may include factors that cannot be expressed in URN (e.g. the colour of a terminal), URN shall provide facilities to connect element of its models to external requirements objects.

URN model elements shall also be traceable and transformable to elements of other languages in the ITU-T family of languages and of UML. This also contributes to the support of round-trip engineering processes where URN is involved.

Traceability within URN models is also important. Operational aspects of goals in URN-NFR models shall be traceable to responsibilities or scenarios in URN-FR models, and vice-versa. Performance constraints identified in URN-NFR models shall be traceable to responsibilities, scenarios, or response-time requirements in URN-FR models, and vice-versa.

8.2 Requirements test case specification

URN shall support the testing of requirements as well as testing based on requirements. A requirement test case specification describes scenarios found, or expected to be found, in the URN-FR specification. The URN-FR specification is assumed to include operationalization of relevant non-functional requirements; hence part of the URN-NFR specification (for example, quantitative performance attributes) is indirectly tested at the same time. The requirement test case specification aims to enable the following types of testing:

- *Validation testing* used to capture individual or small-grained client and user scenarios so that the integrated set of requirements can be determined to be valid by the clients and users. Stakeholders can use this type of testing to establish contract satisfaction.
- *Conformance testing* used to verify designs and implementations against the requirements. Such test cases should be created in a way that would improve compatibility with the ITU-T testing language, TTCN-3.
- *Regression testing* used at the requirements level to ensure a certain degree of compatibility with key system properties during the evolution of requirements.
- *Dynamic assessment* is used in dynamic systems that need to assess capabilities of components and other systems with which they communicate (for instance, does this unknown component support this quality of service?). This dynamic assessment may involve testing of the component or other system in question. URN should provide means for describing such dynamic assessment tests.

The testing of non-functional requirements in general is also desirable but may not be achievable through the use of scenarios. The URN notation need not be able to support this type of testing.

8.3 Performance analysis of requirements

URN should enable at least a preliminary analysis of performance properties, such as response delay or throughput capacity, based on workload and environment parameter estimates attached to the URN-FR specification. Performance properties are of critical importance in telecommunications, and current work indicates that the analysis is feasible. The necessary workload parameters that shall be supported by URN include:

- *Scenario triggering parameters* such as period of initiation, distribution of delays between initiations, etc.
- *Frequencies* of alternative paths.
- *Processing demands* of scenarios, and of operations within scenarios.
- *Demands for system services* other than processing, made by scenarios and operations.

The environment parameters should approximately describe the processing capacity, network delays and the services provided by the environment (for instance, a response delay for a remote service).

Performance requirements such as expected response times shall be expressible in terms of target fragments of scenarios in URN-FR.

Performance results will be delays along defined processing paths, or the range of possible throughputs of some scenarios. The intention of the analysis is to estimate the degree of conformance with stated performance requirements, and to identify problem areas and sensitivities. The data and the results are expected to be approximate. The analysis can be performed in various ways:

- *Point analysis* considers one set of conditions.
- *Sensitivity analysis* considers a range of conditions, and the variation of performance measures with parameter values. This could include sensitivity of the system to its workload parameters or to the environment.

8.4 Change management

It should be possible to version-control URN models and manage evolving requirements. To that end, URN shall provide identifiers for the elements of its models as well as descriptions of document versions.

8.5 Concrete representations

In order to provide aids for communicating requirements, URN shall support the graphical representation of requirements models as well as a tool-oriented interchange format.

The graphical representation allows the modelling of requirements in an iconic and spatial manner which facilitates an intuitive understanding, and emphasizes the informal aspect of the URN. It includes textual annotations for elements which are not intrinsically graphical, such as data and parameters. These annotations shall be traceable to URN graphical elements and be displayable on conventional media (paper or computer screen).

The tool-oriented interchange representation provides for the ability to exchange requirement data between different tools in a standardized manner. W3C's XML and ITU-T's ASN.1 are examples of candidate target interchange formats.

A particular abstract representation of the URN language, achieved using abstract grammars, meta-models based on OMG's MOF, or other means, is not defined by this Recommendation.

8.6 Usability

URN shall be usable by various stakeholders. Usability is subjective and is therefore difficult to measure so that evaluation of how usable is URN, is also subjective. However, the usability of a URN, and of the tools that implement it, is a key aspect of acceptability of URN.

9 Language requirements summary

The purpose of this clause is to summarize the language requirements that the URN shall fulfil.

9.1 Requirements table format

Table 1 presents the table format used to list each of the language requirements defined for the URN (FR and NFR) notations.

Each language requirement possesses a unique identifier (ID) and is typed. A language requirement is of type FR if it relates exclusively to functional requirements. A language requirement is of type NFR if it relates exclusively to non-functional requirements. A language requirement is of type URN if it is common to both functional and non-functional requirements. Language requirements are also defined as being essential (E), i.e. it shall be implemented in this study period, or desirable (D), i.e. it could be delayed to a future study period. Each language requirement is cross-referenced to the clauses where the language requirement is discussed (Ref), and to the objectives to which it contributes (see 1.1). A language requirement is expressed as a capability the URN has.

Table 1/Z.150 – Language requirements table format

ID	Language Requirement	Type	E/D	Ref	Obj
789	Support requirements engineering	URN	E	1.2	a,b,c,d

In the artificial example of Table 1, an essential language requirement of type URN is provided. Its identifier is 789, it is discussed in 1.2, and it contributes to the objectives a, b, c, and d discussed in 1.1.

9.2 URN requirements table

Table 2 lists each of the language requirements defined for the URN (FR and NFR).

Table 2/Z.150 – URN language requirement table

ID	Language Requirement	Type	E/D	Ref	Obj
1	Specify tentative and ill-defined requirements	NFR	E	6.1, 6.12	h
2	Specify refinement of goals and NFRs	NFR	E	6.2, 6.12	h
3	Specify alternative refinement of goals and NFRs	NFR	E	6.2	h
4	Specify alternative functional (operational) requirements	NFR	E	6.2	h
5	Specify satisficeability of goals and NFRs	NFR	E	6.2	h
6	Support (qualitative) goals and NFRs that do not have clear metrics and measurements for their achievements	NFR	E	6.3	h
7	Support quantitative goals and NFRs	NFR	E	6.3	h
8	Specify tradeoffs in goals and NFRs	NFR	E	6.3	h
9	Specify argumentation during modelling	NFR	E	6.4	h
10	Specify business, organizational, and system objectives	NFR	E	6.5	h

Table 2/Z.150 – URN language requirement table

ID	Language Requirement	Type	E/D	Ref	Obj
11	Specify links between high-level objectives and lower-level specifications	NFR	E	6.5	h, i
12	Specify multiple stakeholders' requirements and interests	NFR	E	6.6	h
13	Specify synergies and conflicts among goals and NFRs	NFR	E	6.6	f, g, h
14	Support requirements priorities	NFR	E	6.7	h
15	Support negotiation for solving conflicting goals and NFRs	NFR	E	6.6, 6.7	h
16	Support requirements evolution and changes	NFR	E	6.8	m
17	Handle functional and non-functional requirements concurrently	NFR	E	6.9	i
18	Specify selection criteria when choosing among alternative functional requirements	NFR	E	6.9	h
19	Support incremental commitments of requirements	NFR	E	6.10	h, m
20	Support requirements management during all development phases	NFR	E	6.11	h, m
21	Have model elements that are identifiable and connectable to artefacts in external models	NFR	E	6.12	i
22	Support multiple levels of formality	NFR	E	6.13	b, h
23	Provide ease of use for customers and system users	URN	E	6.13, 8.6	h
24	Provide precise requirements for developers and testers	NFR	E	6.13	h
25	Support modular descriptions of goal and NFR models	NFR	E	6.14	h
26	Support the reuse of goals, NFRs, and knowledge in general	NFR	D	6.15	j
27	Support the mapping of input events and preconditions to output events and postconditions in various degrees of detail	FR	E	7.1	a
28	Specify the set of input events at a scenario start point	FR	E	7.1	a
29	Specify the set of output events at a scenario end point	FR	E	7.1	a
30	Specify preconditions at scenario start points	FR	E	7.1	a
31	Specify postconditions at scenario end points	FR	E	7.1	a
32	Specify input sources (human or machine)	FR	E	7.1	a
33	Specify output sinks (human or machine)	FR	E	7.1	a
34	Specify responsibilities and references to these responsibilities	FR	E	7.2	a
35	Specify system operations as causal flows of responsibilities (paths)	FR	E	7.2	a, b
36	Specify alternative paths	FR	E	7.2, 7.4	a
37	Specify common paths	FR	E	7.2, 7.4	a
38	Specify condition-based decision-making at branching points	FR	E	7.2	a
39	Define a data model and expression evaluator to express and evaluate conditions at branching points	FR	E	7.2	a, c, f
40	Specify parallel or concurrent paths	FR	E	7.2	a
41	Specify synchronisation of paths within a scenario	FR	E	7.2	a

Table 2/Z.150 – URN language requirement table

ID	Language Requirement	Type	E/D	Ref	Obj
42	Specify synchronisation between paths from multiple scenarios	FR	E	7.2, 7.4	a
43	Specify timed synchronization, with a timeout path	FR	E	7.2	a, d
44	Specify repetitive actions within a scenario	FR	E	7.2	a
45	Support hierarchical decomposition of scenarios	FR	E	7.3, 7.4	a
46	Specify subscenarios as scenarios	FR	E	7.3	a
47	Specify subscenario preconditions and postconditions	FR	E	7.3	a
48	Specify scenario containers with multiple subscenarios	FR	E	7.3	a, d
49	Define a data model and expression evaluator to select subscenarios in dynamic containers	FR	E	7.3	a, d, f
50	Group-related scenarios	FR	E	7.4	a
51	Extract individual scenarios from grouped scenarios	FR	E	7.4	a, f, g
52	Specify individual scenarios using a data model and initializations	FR	E	7.2, 7.3, 7.4	a, f, g
53	Express desirable feature interactions in scenarios	FR	E	7.4	g
54	Detect undesirable feature interactions in scenarios	FR	E	7.4	f
55	Specify components and references to these components	FR	E		c
56	Specify scenarios without reference to components	FR	E	7.5	a
57	Specify scenarios where scenario elements are allocated to components	FR	E	7.5	c, d
58	Specify abstract components and COTS	FR	E	7.5	c
59	Specify dynamic entities	FR	E	7.5	d, e
60	Specify system boundaries	FR	E	7.6	c, d
61	Specify the behaviour of the system's environment	FR	E	7.6	c
62	Specify actors external to the system	FR	E	7.6	c
63	Support backward traceability from URN to source documents	URN	E	8.1	k
64	Support forward traceability from URN to the other models used in the development process	URN	E	8.1	k
65	Support facilities to connect URN elements to external requirements objects	URN	E	8.1	l, m
66	Enable transformations to elements of other languages in the ITU-T family of languages and of UML	URN	D	8.1	k
67	Support traceability between operational aspects of goal/NFR models and responsibilities/scenarios in scenario models	URN	E	8.1	i
68	Support traceability between performance constraints in NFR models and responsibilities/scenarios/response-time requirements in scenario models	URN	E	8.1	g, i
69	Support the testing of requirements	URN	E	8.2	f, h
70	Support testing based on requirements	FR	E	8.2	f, k
71	Enable preliminary analysis of performance properties	URN	E	8.3	g

Table 2/Z.150 – URN language requirement table

ID	Language Requirement	Type	E/D	Ref	Obj
72	Attach performance/workload annotations to scenario elements	FR	E	8.3	g
73	Specify the environment's processing capacity, network delays, and services provided	FR	E	8.3	g
74	Specify response times in terms of target fragments of scenarios	FR	E	8.3	g
75	Specify identifiers for the model elements	URN	E	8.4	k, l, m
76	Specify document versions	URN	E	8.4	m
77	Support a graphical representation of requirements	URN	E	8.5	a, h
78	Support a tool-oriented interchange format	URN	E	8.5	a, h
79	Support textual annotations traceable to graphical elements	URN	E	8.5	a, g, h
80	Support textual annotations displayable on conventional media	URN	E	8.5	a, h

Annex A

Compliance to this Recommendation

Descriptions that claim to be compliant to ITU-T Rec. Z.150 (URN) shall conform to the notation grammars defined by ITU-T Recommendations for URN-NFR and URN-FR (whose language requirements are stated in Z.150), with the semantics as defined in these Recommendations. A description is non-compliant if it includes notation grammar which is not allowed by these Recommendations or has analyzable semantics which can be shown to differ from these Recommendations.

A software tool that claims to support Z.150 (here after calleda "tool") should be capable of creating, editing, presenting and analysing compliant Z.150 descriptions.

A tool that implements the graphical or textual Z.150 (URN) is a valid Z.150 (URN) tool, but it is not required to handle both.

A tool that handles a subset of Z.150 is a valid Z.150 URN, URN-FR or URN-NFR (according to the definitions used in this Recommendation) tool only if it supports all essential features of URN, URN-FR or URN-NFR respectively. When a tool supports only URN-FR or only URN-NFR, it is expected to be used in a framework with a tool for URN-NFR and URN-FR respectively.

A tool is not valid if its compliance analysis function fails to detect non-compliance of a description with Z.150. If the tool handles a superset notation, then it is allowed to categorize non-compliance as a warning rather than a failure.

A conformance statement clearly identifying the Z.150 language features and requirements not supported should accompany any tool that handles a subset of Z.150.

Appendix I

Requirements engineering activities

This appendix discusses the major set of activities related to requirements engineering and how these relate to an approach oriented towards goals and scenarios.

Requirement analysis, and more broadly speaking, requirements engineering (RE), covers multiple intertwined activities to arrive at a requirements specification of the intended system. We can suggest the following major activities involved:

- Domain or early requirements analysis;
- Eliciting requirements;
- Modelling and analyzing requirements;
- Documenting and communicating requirements among stakeholders over the system life-cycle;
- Agreeing on and validating requirements;
- Specification;
- Specification analysis;
- Verifying, managing and evolving requirements.

Each of these activities has distinct concerns and may be helped by a goal-oriented and process-oriented URN.

Domain or early requirements analysis: During this phase, the existing environment in which the system should be built is studied, and the relevant stakeholders, who are affected by the intended system, are identified. This may include the intended users, the clients who commissioned the system, 3rd parties, and stakeholders within the development organisation, international standards bodies, and the like. At this early stage, the stakeholders' interests, and how they might be addressed or compromised by various system-and-environment alternatives, is explored. Each alternative may explore different boundaries between the software-to-be and its environment.

Providing for goals (and to a certain extent, for a representation of stakeholders) within the URN-NFR, allows capturing the high-level objectives of pertinent stakeholders. Linking those goals to elements of the requirements specification, design and implementation, aids in better understanding and managing of how changes in these high-level objectives affect the system during development, maintenance and evolution. Capturing and linking goals in such a way also aids in dealing with understanding how software systems either facilitate or hinder co-operation among organisations that wish to create alliances in order to pursue co-operative objectives.

Eliciting requirements: During this phase, alternative models for the target system are further elaborated and explored such that stakeholders' objectives are met. Particular requirements and assumptions on the various organizational actors who would interact with the intended software system, the system, and, to a certain extent, pertinent high-level components of the system, are established. Often the users of systems are not able to articulate their requirements. Describing the tasks, scenarios or use cases for the current and/or the intended system can help users in making their requirements explicit.

Goals, and the ability to refine them towards potential alternative system specifications, may provide guidance when eliciting requirements. Knowledge-based approaches that capture know-how about achieving goals may be invoked during the elicitation process, to suggest further goal refinements and certain functional, structural and organizational requirements for achieving goals. The synergy between scenarios and goals is discussed in the literature. Linking goals to scenarios facilitates checking if all goals have been met, thus establishing completeness of the requirements

specification. The ability to ask "why" for scenarios may provide opportunities to identify new goals, while "how else" questions may yield alternative scenarios for achieving goals.

Modelling and analysis: This is an activity that is found during all phases of requirements engineering and appears to be a core process in requirements engineering. The existing system needs to be modelled in one way or another, while the hypothetical alternative systems need to be modelled as well. Models serve as the basic common interface for the various requirements engineering activities. Models also provide the basis for documentation and evolution.

The ability to provide goals as an explicit modelling construct, together with the ability to refine goals and link them to various requirements, design and implementation-related modelling elements, facilitates making clear how these elements produced in each phase relate to elements of previous phases. In addition, linking elements may provide an anchor point for reasoning about design decisions, justifying or refuting them during the development process. Goals and process support thus provide good groundwork for dealing with requirements evolution and change over the life cycle of the system, i.e., requirements management.

Documenting and communicating requirements: This activity deals with capturing the various decisions made during the RE process, together with their underlying rationale and assumption, and with effectively communicating requirements among stakeholders. Part of the documentation effort that becomes increasingly recognized as crucial, is requirement management which is the ability not only to write requirements, but also to do so in a form that is readable and traceable over the life cycle.

Similarly to the modelling and analysis support, the ability to provide for goals and link them to requirements, design and implementation, provides traceability links from the source of requirements (i.e. stakeholders' goals) to the requirements specification, design and implementation. Goals would appear in all phases and would be related through refinements links, from high-level organizational goals to low-level design goals. This would also allow for managing change during the development life cycle. Documenting NFRs, within a semi-formal or formal framework, also improves the ability to communicate pertinent NFRs among the relevant stakeholders by clustering them together systematically in hierarchical manners, rather than having them spread out informally within text-based documentation.

Achieving agreement on and validation of requirements: Since the source for requirements of a system are the various stakeholders affected and/or involved, disagreement among stakeholders may lead to differences in expectations of what the intended system should provide. This problem is compounded when stakeholders have divergent goals that they wish to have achieved by the system. Explicitly describing the stakeholders' objectives, and how they relate to the system's requirements specification, is a necessary precondition for detecting, negotiating and resolving conflicts among stakeholders. Another aspect of "agreement", besides dealing with conflicts among stakeholders, is the validation of system requirements. This involves the agreement of stakeholders that the documented requirements are in fact meeting their stated objectives.

Capture of goals with the URN allows early detection of apparent conflicting system requirements and may, as a result, help to initiate negotiations for arriving at compromises and agreement. Conflicts among organizational and system-related goals may surface during the detailed requirements specification, but also during the design and implementation phase when particular choices for achieving certain goals exclude the ability for achieving others. Life-cycle support within the URN would then enable identifying and dealing with such conflicts. Negotiation techniques may focus on trying to identify the most important goals of stakeholders, and ensure that they are met, such that the best trade-off among alternatives receives agreement from all parties involved. Goals may aid in determining how, and how well, the objectives of stakeholders were in fact met by the requirements specification.

Specification and specification analysis: These are activities where requirements and assumptions are formulated in precise ways, and checked for deficiencies (such as inadequacies, incompleteness, or inconsistencies) and for feasibility in terms of resources required, development costs etc.

Goals in general, and NFRs-related goals in particular, may aid here in selecting among alternative specification elements, and in justifying such specification decisions. Analysis of specifications may then be placed in the context of organizational goals that are (or are not) achieved throughout the detailed specification. Furthermore, prioritizing goals may allow choosing particular areas within the system where precise and formal specification is more desirable than in other, often less crucial areas, where a semi-formal, or perhaps informal, specification is sufficient. Often, security or performance NFRs play a role when formal methods are needed to prove such properties for the system.

Evolution: The requirements are modified to accommodate corrections, environmental changes, or new objectives. During evolution, both functional behaviour and architecture elements need to accommodate changes in the requirements specification which often originate from changed organizational objectives**.**

As already elaborated earlier, having the ability to describe rationales for the existence of elements within the specification, design and implementation, supports tracing how changes in the objectives of the organization affect the rest of the development. Similarly, how changes in design and implementation artefacts, may affect organizational objectives and their corresponding stakeholders.

A URN should permit the expression of different degrees of formality to reflect the shift in understanding as the users apply refinements to the model during the course of analysis and specification. For example, during early requirements stage, emphasis may be given to relationships among functional and non-functional elements, and stakeholders and their objectives, rather than precise definitions of those elements while, during later stages of requirements, more formality would be necessary for providing more precise requirements descriptions.

Appendix II

Guidelines for the maintenance of URN

II.1 Maintenance of URN

This appendix describes the terminology and rules for maintenance of Z.150 agreed at the Study Group 17 meeting in November 2002, and the associated "change request procedure".

Terminology:

a) An *error* is an internal inconsistency within Z.150.

b) A *textual correction* is a change to text or diagrams of Z.150 that corrects clerical or typographical errors.

c) An *open item* is a concern identified but not resolved. An open item may be identified either by a Change Request, or by agreement of the Study Group or Working Party.

d) A *deficiency* is an issue identified where the semantics of URN are not (clearly) defined by Z.150.

e) A *clarification* is a change to the text or diagrams of Z.150, which clarifies previous text or diagrams that could be ambiguously understood without the clarification. The clarification should attempt to make Z.150 correspond to the semantics of URN as understood by the Study Group or Working Party.

f) A *modification* is a change to the text or diagrams of Z.150 that changes the semantics of URN.

g) A *deprecated feature* is a feature of URN that is to be removed from URN in the next revision of Z.150.

h) An *extension* is a new feature which must not change the semantics of features defined in Z.150.

II.2 Rules for maintenance

In the following text, references to Z.150 shall be considered to include annexes, appendices, and supplements, as well as any addendum, amendment or corrigendum or Implementors guide.

a) When an error or deficiency is detected in Z.150, it must be corrected or clarified. The correction of an error should imply as small a change as possible. Error corrections and clarifications will be put into the Master list of Changes for Z.150 and come into effect immediately.

b) Except for error corrections and resolution of open items from the previous study period, modifications and extensions should only be considered as the result of a request for change that is supported by a substantial user community. A request for change should be followed by investigation by the Study Group or Working Party in collaboration with representatives of the user group, so that the need and benefit are clearly established and it is certain that an existing feature of URN is unsuitable.

c) Modifications and extensions not resulting from error correction shall be widely publicised and the views of users and toolmakers canvassed before the change is adopted. Unless there are special circumstances requiring such changes to be implemented as soon as possible, such changes will not be recommended until Z.150 is revised.

d) Until a revised Z.150 is published, a Master list of Changes to Z.150 will be maintained covering Z.150 and all annexes, except the formal definition. Appendices, addenda, corrigenda, Implementors' guides or supplements will be issued, as decided by the Study Group. To ensure effective distribution of the Master list of Changes to Z.150, it will be published as COM Reports, and by appropriate electronic means.

e) For deficiencies in Z.150, the formal definition should be consulted. This may lead to either a clarification or correction that is recorded in the Master list of changes to Z.150.

II.3 Change request procedure

The change request procedure is designed to enable URN users from within and outside ITU-T to ask questions about the precise meaning of Z.150, make suggestions for changes to URN or Z.150, and to provide feedback on proposed changes to URN. The URN experts' group shall publish proposed changes to URN before they are implemented.

Requests for changes should either use the Change Request Form (see next page) or provide the information listed by the form. The kind of request should be clearly indicated (error correction, clarification, simplification, extension, modification or deprecated feature). It is also important that for any change other than an error correction, the amount of user support for the request is indicated.

The ITU-T Study Group responsible for Z.150 should formally process all change requests at scheduled meetings. For corrections or clarifications, the changes may be put on the list of corrections without consulting users. Otherwise, a list of open items is compiled. The information should be distributed to users:

- as ITU-T white contribution reports;
- as electronic mail to URN mailing lists (such as the ITU-T list *URN@itu.int*);

- by others means as agreed by the Study Group 17 experts.

Study Group experts should determine the level of support and opposition for each change, and evaluate reactions from users. A change will only be put on the accepted list of changes if there is substantial user support for the proposal and no serious objections to it are received from more than just a few users. Finally all accepted changes will be incorporated into a revised Z.150. Users should be aware that until changes have been incorporated and approved by Study Group responsible for Z.150 they are not recommended by ITU-T.

URN Change Request Form

Please fill in the following details		
Character of change:	□ error correction	□ clarification
	□ simplification	□ extension
	□ modification	□ decommission
Short summary of change request		
Short justification of the change request		
Have you consulted other users	□ yes	□ no
Is this view shared in your organization	□ yes	□ no
How many users do you represent?	□ 1-5	□ 6-10
	□ 11-100	□ over 100
Your name and address:		

Please attach further sheets with details if necessary

URN (Z.150) Rapporteur, c/o ITU-T, Place des Nations, CH-1211, Geneva 20, Switzerland. Fax: +41 22 730 5853, e-mail: *URN.rapporteur@itu.int*

Bibliography

This bibliography contains references to related ITU-T standards and to literature on requirements engineering and related topics.

- AMYOT, D. and MUSSBACHER G., URN: Towards a New Standard for the Visual Description of Requirements, In: *3rd SDL and MSC Workshop (SAM'02)*, Aberystwyth, U.K. 2002.
- GOTEL, O. and FINKELSTEIN, A., An analysis of the requirements traceability problem, In: *First Int. Conference on Requirements Engineering* (ICRE'94), Colorado Springs, USA, 94-101, 1994.
- ISO 13407:1999, *Human-centred design processes for interactive systems*. Technical Committee/Sub-Committee: TC159/SC4. Geneva.
- ISO 14598-5:1998 *Information Technology Software Product Evaluation – Part 5: Process for Evaluation*. Technical Committee/Sub-Committee: JT1/SC7. Geneva.
- ITU-T Recommendation I.130 (1988), *Method for the characterization of telecommunication services supported by an ISDN and network capabilities of ISDN*.
- ITU-T Recommendation Q.65 (2000), *The unified functional methodology for the characterization of services and network capabilities including alternative object-oriented techniques*.
- ITU-T Recommendation Q.1200 (1997), *General series Intelligent Networks Recommendation Structure*.
- ITU-T Recommendation Z.100 (2002), *Specification and Description Language* (SDL).
- ITU-T Recommendation Z.100 Supplement 1 (1997), *SDL+methodology: Use of MSC and SDL (with ASN.1)*.
- ITU-T Recommendation Z.105 (2003), *SDL Combined with ASN.1* modules (SDL/ASN.1).
- ITU-T Recommendation Z.109 (1999), *SDL combined with UML*.
- ITU-T Recommendation Z.110 (2000), *Criteria for use of formal description techniques* by ITU-T.
- ITU-T Recommendation Z.120 (1999), *Message sequence chart* (MSC).
- ITU-T Recommendation Z.140 (2003), *Testing and Test Control Notation version 3* (*TTCN-3*)*: Core language*.
- ITU-T, Draft Recommendation Z.450, *Quality Aspects of Protocol-related Recommendations.*
- LIU, L. and YU, E., From Requirements to Architectural Design –Using Goals and Scenarios. In: *From Software Requirements to Architectures Workshop (STRAW 2001)*, Toronto, Canada, 2001.
- NIELSEN, J., *Usability engineering*. San Francisco, USA. Morgan Kaufmann, 1993.
- NUSEIBEH, B. and EASTERBROOK, S. Requirements Engineering: A Roadmap. In: Finkelstein, A. (ed.) *The Future of Software Engineering*. Special track of the *2*nd *Int. Conference on Software Engineering (ICSE'2000)*, ACM Press, 2000.
- OMG (2002), *Meta Object Facility Specification (MOF),* version 1.4. http://www.omg.org/technology/documents/formal/mof.htm

- OMG (2001), *Unified Modeling Language Specification (UML)*, version 1.4. http://www.omg.org/technology/documents/formal/uml.htm
- VAN LAMSWEERDE, A. Requirements Engineering in the Year 00: A Research Perspective. In: *Proc. 22*nd *Int. Conference on Software Engineering (ICSE'2000)*. Limerick, June 2000, ACM press.
- W3C, *Extensible Markup Language (XML) 1.0* (Second Edition). http://www.w3.org/TR/REC-xml

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series B	Means of expression: definitions, symbols, classification
Series C	General telecommunication statistics
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	TMN and network maintenance: international transmission systems, telephone circuits, telegraphy, facsimile and leased circuits
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks and open system communications
Series Y	Global information infrastructure and Internet protocol aspects
Series Z	**Languages and general software aspects for telecommunication systems**

Geneva, 2003