		Considerations on telecom modelling languages
	Location:	N/A
	Meeting date:	N/A
	Responsible:	Anastasius Gavras, Eurescom GmbH, gavras@eurescom.de
	Confidentiality:	MODA-TEL Consortium confidential
MODA-TEL IST-2001-37785	Distribution:	IST Programme participants, OMG
	Document Ref.:	/private/deliverables/D3.add2

MODA-TEL

(Deliverable D3.add2)

Considerations on telecom modelling languages

Editor: Anastasius Gavras, Eurescom GmbH

Suggested readers

Information technology systems architects, senior information technology scientists, research staff especially in the telecommunications domain.

Research and development engineers involved in the OMG standardisation process around all issues relating to Model Driven Architecture

Abstract

Following the recommendation of the project technical reviewers, Mr. Rick Reed and Mr. Svein Hallsteinsen, to more explicitly take into account the established modelling languages in the telecom domain, this document consolidates information of relevance and establishes the relationship of Model Drive Architecture (MDA) and the Unified Modelling Language (UML) to the telecom languages.

The languages addressed are the languages that are standardised by the ITU-T (partly in collaboration with ETSI), namely SDL (Specification and Description Language), MSC (Message sequence chart), ASN.1 (Abstract Syntax Notation One), TTCN (Testing and Test Control Notation), eODL (Extended Object Definition Language), and URN (User Requirements Notation).

In the project MODA-TEL the telecom languages are considered where appropriate, although most of the work will clearly be based on UML, especially in view of UML 2.0. The integration and compatibility of the ITU-T languages with UML 2.0 is underway in the appropriate standardisation groups. It should be noted however that the definition of a language based on a meta-model (as it is the case with eODL) is greatly easing the integration. From this integration the telecom industry as a whole is benefiting with the emerging availability of common tool frameworks that can handle meta-model based languages.

Disclaimer

This document contains material, which is copyright of certain MODA-TEL consortium parties and may not be reproduced or copied without permission. The information contained in this document is the proprietary confidential information of certain MODA-TEL consortium parties and may not be disclosed except in accordance with Section 12 of the consortium agreement.

The commercial use of any information in this document may require a licence from the proprietor of that information.

Neither the MODA-TEL consortium as a whole, nor a certain party of the MODA-TEL consortium warrant that the information contained in this document is capable of use, or that use of the information is free from risk, and accept no liability for loss or damage suffered by any person using the information.

List of Authors

Name	Company	e-mail
João Paulo Almeida	Univ Twente	almeida@cs.utwente.nl
Mariano Belaunde	France Télécom R&D	Mariano.belaunde@rd.francetelecom.com
Erik Berg	Telenor R&D	erik.berg@telenor.com
Klaus-D. Engel	Fraunhofer FOKUS	engel@fokus.fraunhofer.de
Anastasius Gavras	Eurescom	gavras@eurescom.de
Sune Jakobsson	Telenor R&D	sune.jakobsson@telenor.com
Luís Ferreira Pires	Univ Twente	pires@cs.utwente.nl
Julia Reznik	Fraunhofer FOKUS	reznik@fokus.fraunhofer.de

Table of Contents

List of Authors.....	3
Table of Contents.....	4
Abbreviations.....	5
1 Introduction	6
2 MDA and SDL and MSC.....	6
2.1 Alignment UML and SDL	6
2.2 MDA tools and SDL and MSC.....	8
3 Abstract Syntax Notation One – ASN.1	8
3.1 What is ASN.1?	8
3.2 MDA approach when dealing with ASN.1	8
4 Testing and Test Control Notation – TTCN	9
4.1 UML Testing Profile.....	10
4.2 Relation between UML Testing Profile and TTCN-3.....	10
5 Extended Object Definition Language – eODL.....	12
6 User Requirements Notation – URN	12
6.1 URN capabilities.....	13
7 Conclusions	14
References	15
Appendix I.....	17

Abbreviations

ASN.1	Abstract Syntax Notation One
BER	Basic Encoding Rules
DECT	Digital European Cordless Telephone
eODL	Extended Object Definition Language
ETSI	European Telecommunication Standards Institute
GRL	Goal-oriented Requirement Language
GSM	Global System for Mobile Communication
INAP	Intelligent Network Application Protocol
IUT	Implementation under Test
MSC	Message Sequence Chart
N-ISDN	Narrow Band Integrated Services Digital Network
NFR	Non-Functional Requirement
ODL	Object Definition language
OSI	Open Systems Interconnection
PER	Packed Encoding Rules
PIM	Platform Independent Model
PSM	Platform Specific Model
SDL	Specification and Description Language
SUT	System under Test
TETRA	Trans-European Trunked Radio
TTCN	Tree and Tabular Combined Notation (TTCN-2) Testing & Test Control Notation (TTCN-3)
UCM	Use Case Maps
URN	User Requirements Notation
VOIP	Voice over IP

1 Introduction

While the MDA documentation defends that models should be preferably represented using the OMG core technologies, namely the Meta-Object Facility (MOF) [1], the Unified Modelling Language (UML) [2] or the Common Warehouse Metamodel (CWM) [3], this position is not defensible when MDA developers are confronted with design cultures based on other well-established modelling languages. Particularly relevant for the context of the MODA-TEL project are the techniques that are used traditionally in the telecom domain. These include the Specification and Description Language (SDL) [17] and the Message Sequence Chart (MSC) [20] with respect to the modelling of the behaviour of reactive systems, and the Abstract Syntax Notation One (ASN.1) [21], [22], the Testing and Test Control Notation (TTCN) [23], [24], [25] and the User Requirements Notation (URN) [26] with respect to the modelling of data structures, testing and user requirements respectively.

This document is structured as follows: Section 2 discusses relations between MDA and SDL and MSC, Sections 3, 4, 5, and 6, consider the role of ASN.1, TTCN, eODL and URN respectively. In the Appendix I, we provide a table that summarizes a comparison between the ITU-T and the OMG languages.

2 MDA and SDL and MSC

Since MDA is a technique that lends itself to application in a number of different scenarios, it is possible to consider different applications of MDA to a design culture based on the use of SDL and MSC. We identify two categories of approaches:

1. Approaches based on the alignment between UML 2.0 and SDL and MSC; and
2. Approaches based on the reuse and integration of MDA (UML and MOF based) tools for the ITU-T languages considered and associated tools.

2.1 Alignment UML and SDL

The UML is intended as a language that can be specialized for different purposes, including mechanisms for extensions in the language definition. This opens up the possibility to specialize the language for special needs, but is also a potential source of ambiguity in the interpretation of UML models.

Particularly important for the users of SDL is its ability to model the behaviour of implementation-oriented designs. The formal semantics of SDL allows simulation and some formal validation, such as deadlock detection, exhaustive or random exploitation of state space and MSC verification (whether an MSC is an acceptable execution of the SDL specification).

In contrast, the semantics of the behaviour specifications in UML is not formally defined, and a number of semantic variations points defined in the language have to be committed in order to ensure an unambiguous interpretation of UML descriptions. For example, the following semantic variation points defined in the UML 2.0 [5] are important when considering the use of UML 2.0 state charts for behaviour specification:

“The means by which requests are transported to their target depend on the type of requesting action, the target, the properties of the communication medium, and numerous other factors. In some cases, this is instantaneous and completely reliable while in others it may involve transmission delays of variable duration, loss of requests, reordering, or duplication.” and.

“No assumptions are made about the time intervals between event occurrence, event dispatching, and consumption. This leaves open the possibility of different semantic variations such as zero-time semantics.”

The extension capabilities of UML and the SDL formal semantics and integration with MSC can be exploited to enhance the precision and the simulation and validation capabilities of UML. In principle SDL and MSC play similar roles with respect to each other as state charts and interaction diagrams, respectively. However, due to the lack of a standardised action language and formal semantics, simulation and formal validation of UML behaviours are not possible unless a (proprietary) action language is used for this purpose and the relevant semantic variation points are decided. An alternative is, therefore, to use SDL constructs in

combination with state charts, and use the SDL formal semantics as execution semantics to state charts (extended with an action language).

SDL-2000 has been largely aligned [12] with UML 1.5: SDL includes composite states similar to UML; associations have been introduced into SDL; UML Class symbols may be used to represent type references in SDL; and a way to use UML with SDL and mapping to SDL has been defined in recommendation Z.109 [19]. With the development of Z.109, SDL became the first standardized “real-time” UML 1.x profile. Furthermore, a mapping from the SDL language to the UML 1.5 Action Semantics has been defined in [13].

The development of UML 2.0 further increases the possibilities for alignment between the two languages. Profiling UML 2.0 state charts so that their semantics corresponds or approximates the semantics of SDL state machines may enable developers to define transformations from SDL specifications to UML specifications, enabling the reuse of SDL specifications. Figure 1 shows a suggestion for the representation of actions in state charts that has been put forward in the UML 2.0 specification (where this view is named “transition-oriented”).

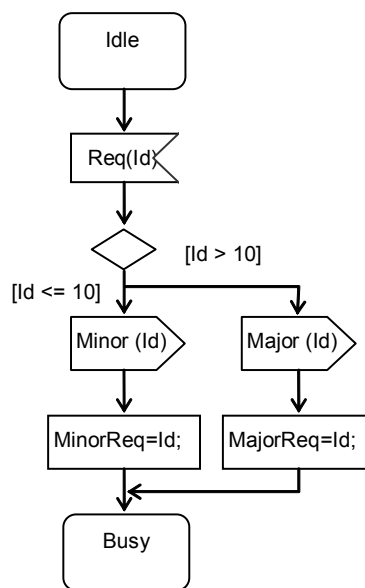


Figure 1 Proposed symbols for representing actions in UML 2.0

In this suggestion we can notice a strong influence of SDL, such as in the symbols that have been proposed to represent signal receipt, choice pseudo state, signal sending and action sequence. SDL also provides a textual action language not defined in UML.

There has been significant alignment in UML 2.0, with respect to the structural constructs in SDL. UML 2.0 now supports hierarchical decomposition of parts with interfaces, which are comparable to SDL agents (blocks and processes). UML 2.0 allows the definition of the internal structure and ports of a certain class. UML 2.0 also allows one to specify collaboration in terms as a set of entities possibly related to some behaviour. A connector in a composite structure is extended to represent a link (instance of an association) that enables communication between two or more entities. The property concept has been introduced to define explicitly the set of instances that can be owned by a classifier instance that contains these instances. The port concept allows one to represent an interaction point of a classifier instance, for connecting it to its environment or to relate it to its internal parts.

The connector concept has been introduced in UML 2.0 to denote communication between entities. A delegation connector relates the external contract of a component (typically an interface) with the internal realisation of the component behaviour, and an assembly connector relates a component that provides a service to another component that requires the service. Furthermore, the realisation relation has been specialised to denote explicitly that a component is an abstraction of a set of classifiers, or, in other words, that a set of classifiers realise a component.

There has been significant alignment in UML 2.0 sequence diagrams, with respect to MSC. UML 2.0 also introduces capabilities to define interaction fragments (smaller sequence diagrams) and to combine them together to form more complex sequence diagrams. These capabilities include (conditional) branching of interaction fragments, and references to interaction occurrences that can be defined separately. The gates

concept has been introduced to connect different interaction fragments. The most important benefit of these new capabilities is the possibility of structuring sequence diagrams in terms of smaller fragments, increasing in this way the readability of sequence diagrams, mostly of importance in the case of complex diagrams. These capabilities make it possible to define behaviours more concisely and completely, approaching in this way the purpose and expressiveness of statecharts.

Timing diagrams have been introduced in UML 2.0 to represent state changes and conditions on a timeline. Timing diagrams are expected to be useful for systems that have stringent timing constraints. UML 2.0 also defines the so-called interaction overview diagram, which allows sequence diagrams to be combined in the scope using the operator of activity diagrams. This implies that more alternative scenarios can be represented in a single diagram. Interaction overview diagrams bring interaction diagrams closer to activity diagrams, by allowing them to represent behaviours in a more complete way.

The following are on-going efforts on the alignment SDL and UML:

- ITU-T Study Group 17, Question 17/17 - Unified Modelling Language (UML) combined with ITU-T languages [14], which includes revision efforts for Z.109.
- ETSI Specialist Task Force (STF) 250: UML profile for Communicating Systems [11]. The purpose of this STF is to produce a UML profile for communicating systems. The output of this STF will be a UML profile for communicating systems based on, but not limited to, SDL. The requirements for this profile include the ability to perform functional verification and generate executable applications. This STF will be active until February 2004.

2.2 MDA tools and SDL and MSC

It is possible to represent the SDL meta-model in MOF, enabling the reuse of MOF-based model repositories, model transformation languages and tools, and XMI model interchange for SDL specifications. When this solution is not accompanied by profiling the UML, tool support for the SDL notation has to be specifically designed.

3 Abstract Syntax Notation One – ASN.1

3.1 What is ASN.1?

ASN.1 is a formal notation used for describing data transmitted by telecommunications protocols, regardless of language implementation and physical representation of these data, whatever the application, whether complex or very simple. ASN.1 only covers the structural aspects of information. There are no operators to handle the values once these are defined or to make calculations with. Therefore it is not a programming language.

One of the main reasons for the success of ASN.1 is that this notation is associated with several standardised encoding rules such as the BER (Basic Encoding Rules), or more recently the PER (Packed Encoding Rules), which prove useful for applications that undergo restrictions in terms of bandwidth. These encoding rules describe how the values defined in ASN.1 should be encoded for transmission, i.e. how they can be translated into the bytes 'over the wire' and reverse, regardless of machine, programming language, or how it is represented in an application program. ASN.1's encodings are more streamlined than many competing notations, enabling rapid and reliable transmission of extensible messages – an advantage for wireless broadband. Because ASN.1 has been an international standard since 1984 (current version in force as of July 2002 [21], [22]) its encoding rules are mature and have a long track record of reliability and interoperability.

For more details on ASN.1 see the introduction material and tutorial at <http://asn1.elibel.tm.fr/>

3.2 MDA approach when dealing with ASN.1

When a platform expects a concrete data encoding based on ASN.1, then it is necessary it will be necessary to produce an ASN.1 data description. There at least two possible approaches:

- The data is described in the Platform Independent Model (PIM) model independently of the ASN.1 formalism using for instance class diagram based notation. Then mapping rules are to be provided to generate automatically the ASN.1 description. In this case ASN.1 is seen as a Platform Specific Model (PSM) for modelling data.
- The PIM model adopts ASN.1 as the way to describe data. In such case the ASN.1 may be provided either using the standard textual notation or could be provided graphically using a dedicated ASN.1 UML profile. ASN.1 can also be used within a MOF-based repository tool. For that purpose it is necessary to define a meta-model describing all the concepts of this formalism. Model-to-model transformation techniques can then be used to inspect or produce automatically ASN.1 data descriptions.

4 Testing and Test Control Notation – TTCN

The Tree and Tabular Combined Notation (TTCN) [27] is a language for writing test specifications. TTCN was first published as an ISO standard in 1992 [28], where OSI conformance testing is understood as functional black-box testing i.e. an Implementation Under Test (IUT) is given as a black-box and its functional behaviour is defined in terms of inputs and outputs from the IUT. Since TTCN was standardised, the use of the language has steadily grown within the telecommunications industry. TTCN has been used to specify tests for technologies such as GSM, DECT, INAP, N-ISDN, Q-Sig, TETRA, VB-5, Hiperlan, 3G (terminals) and VOIP.

Recently the European Telecommunication Standards Institute (ETSI) funded a team to evolve TTCN into a language whose look and feel is of a modern programming language with test specific extensions, called the TTCN third edition (TTCN-3) (Testing and Test Control Notation) [23], [24], [25]. These extensions consist of: test verdicts, matching mechanisms to compare the reactions of the IUT with the expected range of values, timer handling, distributed test processes, ability to specify encoding information, synchronous and asynchronous communication, and monitoring. In providing these extensions it is expected that staff specifying tests and engineers will find this general-purpose language more flexible, more user-friendly and easier to use than its predecessor. As illustrated in Figure 2, TTCN-3 also encompasses different presentation formats: tabular format and a Message Sequence Chart (MSC)-like format [14].

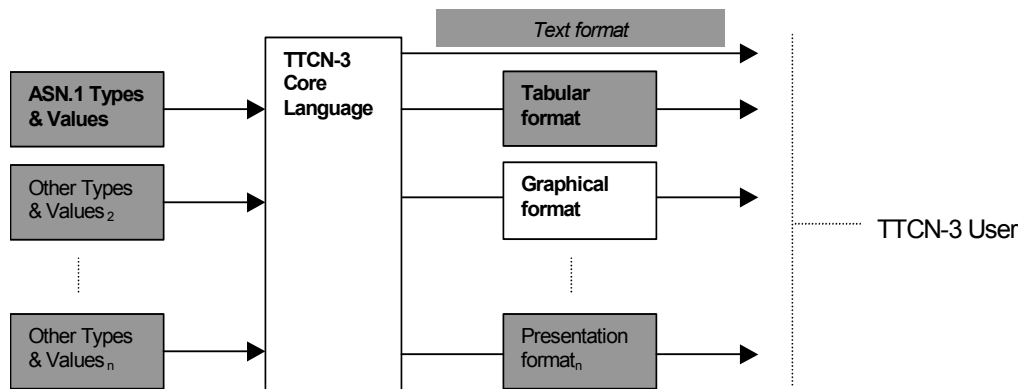


Figure 2: Users view of TTCN-3 and the various presentation formats

TTCN-3 is intended for the various application areas like protocol testing (e.g. mobile and internet protocols), supplementary service testing, component testing, module testing, testing of CORBA based platforms, testing of API's, robustness testing, performance testing, regression testing etc.

TTCN-3 is on syntactical and methodological level a drastic change to TTCN-2, however, the main concepts of TTCN-2 have been retained and improved and new concepts have been included, so that the expressive power and applicability of TTCN-3 are increased. New concepts are e.g. test execution control to describe relations between test cases such as sequences, repetitions and dependencies on test outcomes, dynamic concurrent test configurations and test behaviour in asynchronous and synchronous communication environments. Improved concepts are e.g. the integration of ASN.1, the module and grouping concepts to improve the test suite structure, and the test component concepts to describe concurrent and dynamic test set-ups.

The top-level unit of a TTCN-3 test suite is the module, which can import definitions from other modules. A module consists of a definitions part and a control part. The definitions part of a module covers definitions e.g. for test components, their communication interfaces (so called ports), type definitions, test data templates, functions, and test cases. The control part of a module calls the test cases and describes the test campaign. For this, control statements similar to statements in other programming languages (e.g. if-then-else and while loops) are supported. They can be used to specify the selection and execution order of individual test cases.

Test cases describe the probes during the test campaign i.e. they specify the test behaviour. One can express a variety of test relevant behaviour within a test case such as the alternative reception of communication events, their interleaving and default behaviour to cover, for example unexpected reactions from the tested systems. In addition to the automatic test verdict assignment, more powerful logging mechanisms e.g. for a detailed tracing, are provided.

4.1 UML Testing Profile

The UML Testing Profile [6] defines a UML 2.0 based meta-model as well as MOF model which enables the use of the Testing Profile independent of UML. Within the specification mappings to test execution environments like JUnit and TTCN-3 are outlined. This enables the reuse of an existing e.g. TTCN-3 infrastructure. The profile only addresses black-box conformance testing.

The profile is organized in four sections describing the *test architecture*, *test behavior*, *test data* and *time concepts*.

- The *Test Architecture* section contains the concepts to organize the *test cases* for a *SUT* (system under test) in a *test suite*. The SUT provides a set of operations via publicly available interface(s). The test suite also contains an implementation of an *arbiter*, which determines the final verdict of a test case, which may be *pass*, *inconclusive*, *fail* or *error*.
- The *Test Behavior* Section defines concepts to describe the behavior of a test case in the context of a test suite. Public test case operations represent the interface to the testers. There may also be private or protected test cases, which are only used as utilities in the realization of a public test case. A test case will return a verdict as described above. Concepts used to describe the behavior of the test suite are: the *test case*, the *verdict*, the *test objective*, *Finish-*, *Log-* and *ValidationAction* etc.
- The *Test Data* sub package covers the concepts for data sent to the SUT as stimulus and received from the SUT as observations. Mechanisms in order to change and compare test data are used to enable precise and succinct test specifications. Coding rule and wildcard specifications are the main concepts defined within this section.
- Time concepts are essential to provide complete and precise test specifications. The testing Profile provides a set of small and useful *Time Concepts*. *Timers* and *Timeouts* can be handled. *Timezones* can be used as grouping mechanism.

4.2 Relation between UML Testing Profile and TTCN-3

The concern of the UML Testing profile is limited to black-box conformance testing which is only a part of the area addressed by TTCN-3. All concepts of the UML Testing Profile [6] have direct correspondence or can be mapped to TTCN-3 testing concepts. This correspondence / mapping is directly addressed in the UML Testing Profile specification by the following table:

UML Testing Profile	TTCN-3
---------------------	--------

Test Behavior:

Test Control	The control part of a TTCN-3 module.
Test Case	A TTCN-3 test case. The behaviour of a test case is defined by functions, which are generated via mapping functions applied to the behavioural features of a test suite. The main test component (MTC) is used like a "controller" that

	creates test components and starts their behaviour. The MTC controls also the arbiter.
Test Invocation	The execution of a TTCN-3 test case.
Test Objective	Not part of TTCN-3, just a comment to a test case definition.
Stimulus	Sending messages, calling operations, and replying to operation invocations.
Observation	Receiving messages, operation invocations, and operation replies.
Coordination	Message exchange between test components.
Verdict	The default arbiter and its verdict handling is an integral part of TTCN-3. For user-defined, a special verdict type and updating the arbiter with set verdicts is needed.
Validation Action	External function or data functions resulting in a value of the specific verdict type.
Log Action	Log operation.
Test Trace	Not part of TTCN-3, but could be mapped just as a strict sequential behavioral function.

Test Architecture:

Test Suite	TTCN-3 module definition part covering all test cases and related definitions of a test suite, having a specific TSI component type (to access the SUT) and a specific behavioural function to set up the initial test configuration for this test suite.
Test Configuration	Configuration operations create, start, connect, disconnect, map, unmap, running and done for dynamic test configurations. Behavioural function to set up the initial test configuration.
Test Component	TTCN-3 component type1, used for the creation of test components and their connection to the SUT and to other test components.
System Under Test (SUT)	The test system accesses the SUT via the abstract test system interfaces (TSI). The SUT interfaces result in port types used by TSI. One additional port is needed to communicate with a user-defined arbiter. Potentially additional ports are needed to coordinate/synchronize test components.
Arbiter	The Testing Profile default arbiter is a TTCN-3 built-in. User-defined arbiters are realized by the MTC.
Utility Part	External constants and/or external functions to refer and make use of the utility part, which is outside the TTCN-3 module.

Test Data:

Wildcards	TTCN-3 matching mechanisms.
Data pool	An external constant (referring to the data in the data pool) or external functions to get access to the data pool.
Data partition	TTCN-3 matching mechanisms can be used to handle data partitions for observations. For stimuli however, user defined functions are needed to realize the test case execution with different data to be sent to the SUT.
Coding rules	TTCN-3 encode and encode variant attributes.

Time Concepts:

Timezone	Cannot be represented in TTCN-3.
Timer	TTCN-3 timer.

5 Extended Object Definition Language – eODL

The ITU-T Rec. Z.130 (eODL), which supersedes the previous 1999-version of the ITU-T Object Definition Language, defines most of the computational concepts based on the RM-ODP [29], [30] terminology. In contrast to ODL, eODL is defined based on a meta-model instead of the traditional abstract syntax approach.

ITU-T eODL is used for a component-oriented development of distributed systems from a perspective of four different but related views: a computational, implementation, deployment, and target environment view. Each view is connected with a specific modelling goal expressed by dedicated abstraction concepts.

Naming and scoping is defined based on the *Container-Contained* relation.

The **Computational Concepts** of the recommendation are based on CORBA IDL concepts. The concepts of *Data Types*, *Interface Types*, *Operations* and *Attributes* are extended by the introduction of new concepts not offered by CORBA IDL. These concepts comprise *Signal* and *Signal Parameters* to be able to handle signal based interactions, *Media type*, *Medium* and *Mediaset* for the exchange of continuous media in a distributed system, *Sink* and *Source* as the interaction elements for the consumption and production of *Mediasets*, *Consume* and *Produce* as the interaction elements for signals. *CO Types*, *Interface Types* and the *support and requires Relation* are used to specify the functional decomposition of a system.

The **Implementation Concepts** of the Recommendation are driven by the *Artefact and Instantiation Pattern*.

The **Deployment Concepts** offer the possibility to describe *Software Components* and *Component Dependencies*, *Assemblies* and *Initial Configurations* and to define and attach *Properties* and *Constraints* to model elements.

With the **Target Environment Concepts** *Target Environments* with their *Nodes* and *NodeLinks* can be described. *InstallationMaps* associate software components with nodes, while *InstantiationMaps* represent the way concrete instances of *CO Types* are distributed on a target environment.

6 User Requirements Notation – URN

ITU-T Study Group 10 (now Study Group 17) has started, in November 2000, work towards the standardization of a User Requirements Notation (URN) that targets the representation of requirements for future telecommunication systems and services. At this early stage of its development, URN contains one notation for the representation of Non-Functional Requirements (NFRs) and a complementary scenario notation for functional requirements. As of February 2003 ITU-T Rec. 150, User Requirements Notation (URN) - Language requirements and framework, is in force [26].

NFRs are seldom captured explicitly in design processes, and URN is a first standardisation effort in that direction. The NFR notation being proposed is the Goal-oriented Requirement Language (GRL). The scenario notation is based on Use Case Maps (UCM) and hence abstracts from message exchanges.

Currently there exist draft Recommendation Z.151 (GRL: Goal-oriented Requirements Language) and draft Recommendation Z.152 (UCM: Use Case Map notation), but both drafts are not expected to be submitted for consent at the ITU-T before March 2004. Currently progress is being made towards formalisation of GRL and UCM with the aim to ease compatibility with UML 2.0. A third draft document linking URN to other notations (Z.153 - URN: Methodological Approach) is currently at a very early stage.

The reader is advised to visit <http://www.usecasemaps.org/urn/> for further information and extensive documentation on URN and related activities.

6.1 URN capabilities

The URN is defined to have the following capabilities:

- Describe scenarios as first class entities without requiring reference to system sub-components, specific inter-component communication facilities, or sub-component states;
- Capture user requirements when very little design detail is available;
- 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;
- Have dynamic refinement capability with the ability to allocate scenario responsibilities to architectural components;
- Entities;
- Facilitate detection and avoidance of undesirable interactions between features;
- Provide insight at the requirements level that enables designers to reason about feature interactions and performance trade-offs early in the design process.
- Provide facilities to express, analyse and deal with goals and non-functional requirements;
- Provide facilities to express the relationship between goals and system requirements;
- Provide facilities to capture reusable analysis and design knowledge related to know-how for addressing non-functional requirements;
- Provide facilities to trace and transform requirements to other languages (especially ITU-T notations and UML);
- Provide facilities to connect URN elements to external requirements objects;
- Provide facilities to manage evolving requirements.

The current URN proposal includes two complementary languages: Goal-oriented Requirement Language (GRL) for describing and reasoning about business goals and non-functional requirements, and Use Case Maps (UCM) for capturing functional requirements in the form of causal scenarios.

GRL

The Goal-oriented Requirement Language (GRL) addresses most of URN's additional objectives. Goal-oriented modelling has been proposed in the requirements engineering community for a number of years and several approaches have been published. GRL is a rather new addition to this growing list of techniques but builds on the well-established NFR framework, and the agent-oriented language *i** [15] for the modelling, analysis, and reengineering of organisations and business processes.

GRL captures business or system goals, alternative means of achieving goals, either objectively or subjectively, and the rationale for goals and alternatives. The notation may be applied to non-functional as well as functional requirements.

UCM

Use Case Maps (UCMs) are a scenario-based software engineering technique that have a history of application to the description of object-oriented systems and reactive systems in various areas, including wireless, mobile, and agent domains.

UCMs are used as a visual notation for describing causal relationships between responsibilities of one or more use cases, optionally allocated to a structure of abstract components. UCMs are most useful at the early stages of software development and are applicable to use case capturing and elicitation, use case validation, as well as high-level architectural design and test case generation.

7 Conclusions

While the MDA documentation defends that models should be preferably represented using the OMG core technologies, this position is not defensible when MDA developers are confronted with design cultures based on other well-established modelling languages, such as, e.g., SDL and MSC, ASN.1, TTCN and URN in the telecom domain. We have defined a number of opportunities for the use of these technologies within the context of the MDA.

It is important to note, however, that the telecom domain is large and encompasses design cultures that are associated to object-oriented technologies and languages, in particular, UML. For these design cultures, technologies defined by the OMG already predominate and the ITU-T technologies addressed in this document are not used.

References

- [1] Object Management Group. Meta Object Facility (MOF), Version 1.4, April 2002, <http://www.omg.org/docs/formal/02-04-03.pdf>
- [2] Object Management Group, Unified Modelling Language (UML), Version 1.5, March 2003, <http://www.omg.org/docs/formal/03-03-01.pdf>
- [3] Object Management Group, Common Warehouse Metamodel (CWM), Version 1.1, March 2003, <http://www.omg.org/docs/formal/03-03-02.pdf>
- [4] Object Management Group, UML 2.0 Infrastructure RFP (ad/03-09-15), Sept. 2003.
- [5] Object Management Group, UML 2.0 Superstructure RFP (ad/03-08-02), Aug. 2003.
- [6] Object Management Group UML Testing Profile (final submission) (ad/03-03-26), March 2003
- [7] M. Andersson, A. Ek and N. Landin. Utilizing UML in SDL-based development. *Computer Networks* 35:613-625. 2001.
- [8] Telelogic Products – Telelogic Tau SDL suite. <http://www.telelogic.com/products/tau/sdl/index.cfm>
- [9] Holz, E. A Meta-Model based Approach for the Combination of Models in multiple Languages. <http://www.informatik.hu-berlin.de/~holz/Literatur/Paper-380-140.pdf>
- [10] Braek, R. and Meisingset, A. The ITU-T Languages in a Nutshell, *Telektronikk*, Volume 96 No. 4 – 2000, ISSN 0085-7130 http://www.item.ntnu.no/fag/ttm4115/IntroMethodology/Telek4_2000%20ITU-T%20languages.pdf
- [11] Specialist Task Force 250: UML profile for Communicating Systems http://portal.etsi.org/stfs/ToR/TOR250v05_MTS_UML_Profile.doc
- [12] Joachim Fischer, Eckhardt Holz, Martin v. Löwis, Andreas Prinz. SDL-2000: A Language with a Formal Semantics. Available at <http://www.informatik.hu-berlin.de/~holz/Literatur/room3.PDF>
- [13] Morgan Bjorkander, Ileana Ober, and Thomas Weigert. SDL Mapping for the UML Action Semantics. OMG document ad/00-08-01, Aug. 2000. Available at <http://www.omg.org/docs/ad/00-08-01.pdf>
- [14] P. Baker, E. Rudolph, I. Schieferdecker: Graphical Test Specification - The Graphical Format of TTCN-3. Proc. of the 10th SDL Forum 2001, Copenhagen, June 2001.
- [15] Yu, E. (1997) “Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering”. Proc. of the 3rd IEEE Int. Symp. on Requirements Engineering (RE’97), Washington, D.C., USA, 226-235.
- [16] International Telecommunications Union (ITU-T). Question 17/17 - Unified Modelling Language (UML) combined with ITU-T languages. Available at <http://www.itu.int/ITU-T/studygroups/com17/sg17-q17.html>

ITU-T documents

- [17] ITU-T Rec. Z.100 (08/02) Specification and Description Language (SDL)
- [18] ITU-T Rec. Z.100 Annex F1 (11/00) SDL formal definition: General, Z.100 Annex F2 (11/00) SDL formal definition: Static semantics and Z.100 Annex F3 (11/00) SDL formal definition: Dynamic semantics
- [19] ITU-T Rec. Z.109 (11/99) SDL combined with UML
- [20] ITU-T Rec. Z.120 (11/99) Message sequence chart (MSC)
- [21] ITU-T Rec. X.680 (07/02) Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation
- [22] ITU-T Rec. X.680-X.693 (07/02) Information Technology - Abstract Syntax Notation One (ASN.1) & ASN.1 encoding rules

- [23] ITU-T Rec. Z.140 Testing and Test Control Notation version 3 (TTCN-3): Core language (also available as ETSI document 201 873-x)
- [24] ITU-T Rec. Z.141 Testing and Test Control Notation version 3 (TTCN-3): Tabular presentation format (also available as ETSI document 201 873-x)
- [25] ITU-T Rec. Z.142 Testing and Test Control Notation version 3 (TTCN-3): Graphical presentation format (also available as ETSI document 201 873-x)
- [26] ITU-T Rec. Z.150 User Requirements Notation (URN) - Language requirements and framework
- [27] ITU-T Recommendation X.290 (1995): “OSI Conformance Testing Methodology and Framework – General Framework”
- [28] ISO/IEC 9646-3 (1998): “Information technology - Open systems interconnection - Conformance testing methodology and framework - Part 3: The Tree and Tabular combined Notation (TTCN)”
- [29] ITU-T Recommendation X.902 (1995) – ISO/IEC 10746-2: 1995, Information Technology – Open Distributed Processing - Reference Model: Foundations.
- [30] ITU-T Recommendation X.903 (1995) – ISO/IEC 10746-3: 1995, Information Technology – Open Distributed Processing - Reference Model: Architecture.

Appendix I

This table has been adapted from [10] and updated to reflect relevant developments in UML 2.0, URN, the UML testing profile, eODL, etc.

UML	ITU-T Languages
<p>Use case diagram; depicts relationships between (users as) actors and their use cases (as tasks or functions).</p> <p>No semantics attached</p>	<p>The Use Case Maps (currently work in progress in URN) provide a graphical scenario notation for describing casual relationships between responsibilities. Furthermore scenario elements may be linked to components</p>
<p>Class diagrams; depict classes and their various relationships, including associations and interfaces. Classes can include attributes, operations and methods. SDL capabilities to define agent diagrams that define the composite object structure of agents and agent types has been added in UML 2.0.</p>	<p>Agent diagrams and package diagrams can contain class diagrams that partially define the types. UML classes are specialised (by stereotypes in UML) into ITU-T SDL (UML 1.4 in Z.109). The stereotypes are <<system>>, <<block>>, <<process>>, <<procedure>>, <<interface>>, <<object>>, <<value>>, <<state>>. UML compositions correspond to SDL decompositions.</p>
<p>Interaction diagrams; can be of the following kinds: Sequence diagrams; depict the time ordering of messages exchanged between objects. Collaboration diagrams; depict interactions between objects in an alternative form. Structuring capabilities in UML 2.0 corresponding to MSC.</p>	<p>ITU-T MSCs can depict the information in sequence diagrams.</p> <p>MSC support Simple MSC diagrams, High-level MSC diagrams, and MSC documents.</p>
<p>State chart diagrams; depict optional state machine behaviour associated with classes. Activity diagrams; are a special kind of state chart showing the flow between activities only. Semantic variation points to be defined in specific profiles.</p>	<p>SDL state diagrams specify (state machine) behaviour associated with agents including creation and deletion of agent instances, data operations and timer operations.</p>
<p>Component diagrams; depict organisation and dependencies between implementation components. Components typically map to classes, interfaces and collaborations. See class diagrams.</p>	<p>ITU-T ODL defines objects types with multiple interfaces for both operational and stream data.</p>
<p>Deployment diagrams; depict the configuration of processing nodes and the components running in them.</p>	<p>Not supported. The issue may be addressed in a new Question on Deployment and Configuration Language, DCL.</p>
<p>Data syntax is not supported by UML.</p>	<p>ASN.1; defines data syntax for protocol and other data. ASN.1 is typically used together with SDL.</p>
<p>A Testing Profile is under specification which only addresses black-box conformance testing.</p>	<p>TTCN; defines test cases for protocol testing. TTCN is conveniently used together with MSC.</p>