A formal framework for model verification in System Engineering: UPSL

Vincent Chapurlat

Internal Research Report / Rapport Interne de Recherche n° RR/10-02

A formal framework for model verification in System Engineering: UPSL

V.Chapurlat

Ecole Nationale Supérieure Mines Alès -- 6 avenue de Clavière, 30319 Alès cedex
LGI2P - Laboratoire de Génie Informatique et d'Ingénierie de Production, Parc scientifique Georges Besse, F30035 Nîmes

A formal framework for model verification in System Engineering: UPSL

V.Chapurlat

LGI2P - Laboratoire de Génie Informatique et d'Ingénierie de Production, Site EERIE de l'EMA, Parc Scientifique George Besse, 30035 Nîmes cedex 1- tél. (+33) 466 387 066

Abstract: the aim of this paper is to present and to illustrate a formal model verification framework called UPSL (Unified Property Specification Language) applied here in System Engineering verification activities context. The concepts and principles, the associated languages and supporting platform specification are presented. This framework is based on formalization of requirement under the form of properties and the use of proof mechanisms of properties taking into account the multi models, multi disciplinary, multi domains, multi paradigms, and last the collaborative characteristic of a system engineering project.

Keywords: System Engineering, Modelling, Verification, Property, Formal proof

Introduction

Systems Engineering (SE) is « *an interdisciplinary approach and means to enable the realization of successful systems. It focuses on defining customer needs and required functionality early in the development cycle, documenting requirements, and then proceeding with design synthesis and system validation while considering the complete problem* » (INCOSE 2007, ISO 2002, CDT 2007)
Verification and validation (V&V) activities represent today a huge but necessary investment in SE processes and impacts Integration System processes. However, a system engineering project is by definition a multi disciplinary, multi actors, multi paradigms i.e. multi modelling approaches and tools candidate, and last a collaborative process during which actors with different objectives, skills, competences and tools have to interact in order to realize a system-of-interest. This is may be the reasons for which verification activities remain poorly formal.
This article presents a formal verification framework named UPSLaiming to associate modeling and verification concepts, tools and techniques in a coherent and shareable collaborative working space. The first part describes and illustrates the problematic to be cover. The second part presents the core concepts of UPSL. The third part presents principles of formal verification approaches. Last the fourth part presents and illustrates the methodology of use of UPSL.

Problematic

Verification is defined as "*the process of determining whether or not the products of a given phase of the system/software life cycle fulfill the requirements established during the preceding phase*". In the same way validation is defined as "*the process of determining that the requirements are the correct requirements and that they form a complete set of requirements this is done at the early stages of the development process*". In other words verification and validation must prove the system satisfies the requirements (verification) and that it meets the users needs (validation). So, these activities are requested all along engineering process because they formalize the dependence between modelling and decision tasks. Modelling tasks provide system models allowing to understand, to specify, to communicate between actors, to evaluate rapidly solutions and so on. Decision tasks aim to design, to manage the differences between solutions and then to confirm some of these solutions, to launch development of prototypes and so on. For this different techniques are used:

- Analyze: if system do not exist or if a prototype cannot be designed, model analysis can be performed in confidence by using in general simulation, even partial, from these models.
- Inspection: this aims to provide some proof by examining for example prototypes of the system.
- Demonstration: the proof consists to gather some results considered as sufficient for a given purpose, of some trials made on a prototype or a demonstrator.

- Test: the proof is then a very detailed result considering performance measures.

The problem considered here is to improve confidence in models used all along a project for being able to concentrate engineers' efforts on model analysis techniques. Numerous more or less formal techniques summarised in Fig. 1 can be found in the bibliography (Love 2000, Jagdev 1995, RPG 2001, NASA 2001). Adopting one of them depends from different criteria: type and nature of system, relevance and well known technique in the field, existing best practices, skills and experience, tools more or less "easy to use", project constraints (costs, duration)...
This paper focuses on the use of a formal technique as proposed in other domains (Van Lamsweerde 2002, Bérard et al. 2001, Balci et al. 2002), particularly model checking and theorem proving approaches. They are based on mathematical concepts, reasonning rules and mechanisms that guarantee the absence of ambiguity in the produced models, the possibility to reason about specifications in order to discover potential incompleteness, inconsistencies. The proposed framework called Unified Properties Specification Language (UPSL) focus on properties proof and covers three main needs:

- Describing a system requires building several models (requirements, functional, organic, behavioural …) by using several modeling languages which can be not sufficiently formal to support checking techniques. Their verification requires being able to translate without loss and ambiguities these models under more formal ones and to check the pointed out properties on these models.
- These properties can concern consistency (no contradiction in the information contained in one or several models of the same system), completeness (considering the expectations of the entire project) and relevance (i.e. completeness vs. interest or fidelity) of the model vs. system-of-interest. They can also formalise the requirements having to be verified by the models. Last, they can formalise physical, chemical, electrical … laws from environment of the system which remain always mandatory to respect all along a SE process. It is then necessary to be able to formalize and to handle these properties.
- Last, the proposed framework must stay open to potential verification tools.

Focus on (Information, Resource, Organisation, Behaviour)

Techniques

	Static aspect	***Dynamic aspect***
Informal	Reference models and reference architectures utilization, Audit, Human expertise, Face Validation , Reviews (models, project), Walkthroughs, Desk Checking, Inspections, Turing Test Automated documentation generation, Models comparison (by human expert)	
Semi-formal	Cause-Effect Graphing Data Analysis (data dependency, data flow) Interface Analysis (model interface, user interface) Structural Analysis Syntax Analysis Control Analysis (calling structure, concurrent process, control flow, state transition) Fault/Failure Analysis Semantic Analysis Symbolic Evaluation Traceability Assessment Automated documentation generation	Acceptance Testing, Assertion Checking, Bottom-Up Testing Compliance Testing (authorization, performance, security standards) Execution Testing (monitoring, profiling, tracing, reporting) Field Testing, Graphical Comparisons, Object-Flow Testing Predictive Validation, Regression Testing, Statistical Techniques Structural testing (White-Box), Functional testing (Black-Box) Testing (branch, condition, data and controls flow, loop, path, path condition, statement,) Debugging (symbolic, classic) , Comparison Testing Fault/Failure Insertion Testing, Interface Testing (data, model, user) Partition Testing, Product Testing, Sensitivity Analysis Special Input Testing (boundary value, equivalence partitioning, extreme input, invalid input, real-time input, self-driven input, stress, trace-driven input) Sub-model/Module Testing, Top-Down Testing, Visualization/Animation
Formal	Formal methods utilization (B,Z,VDM; other) and associated tools Induction, deduction, abduction, model checking, theorem proving, Inference, Inductive Assertions, Proof (correctness, completeness, consistence), Properties proof, Model mapping Predicate transformations, Predicate Calculus, Logic (temporal, propositional, first order, etc.), Algebra (linear, process algebra, dedicated algebras), Lambda calculus Simulation (when based on formal behavioral rules and models), Bi simulation	

Fig. 1: an overview of verification techniques

UPSL: a formal verification framework

UPLS aims to enlarge the engineer tools box with a complementary way with other approaches such as simulation, expertise or test. In this it has to help a user to specify, to organise and to check a set of formal properties according to the different models of the system-of-interest. UPSL provides concepts and mechanisms described and illustrated in the next parts:

- Support requirements specification. This requires a requirement modelling language whatever may be the requirement origin called **Requirement Model**.

- Support properties formalisation including the reusability of properties defined in other projects. This requires a property modelling language called **CREDI** model and a properties reference data base called **Property Reference Repository**.
- Be able to take into consideration different purposes of modelling and different modelling languages used during SE process. It maust then provide a set of **mechanisms supporting modelling languages conceptual modelling, conceptual enrichments** and **models transformation**.

Last, using UPSL requires an implementation approach based on three main phases of work summarised in Fig. 2:

- The first phase consists for experts from domain to provide a meta model of the different used modelling languages during SE process (eFFBD, State diagrams, SysML, MODELICA...). This allows first to propose conceptual enrichments of these languages in order to take into account the CRED model. It permits second to set up a list of concepts and relations, attributes and constraints (cardinality, rules) and operational semantic required for the re writing process. This phase is done for each modelling language.
- The second phase consists to model in same time system-of- interest and to formalize its requirements under the form of properties when it is possible. Indeed, some properties can be checked by using formal tools (model checkers and theorem provers), can be used for determining and adjusting a simulation or can also dedicated only to expertise point of view if the systems models cannot be checked due to an inappropriate level of formalisation.

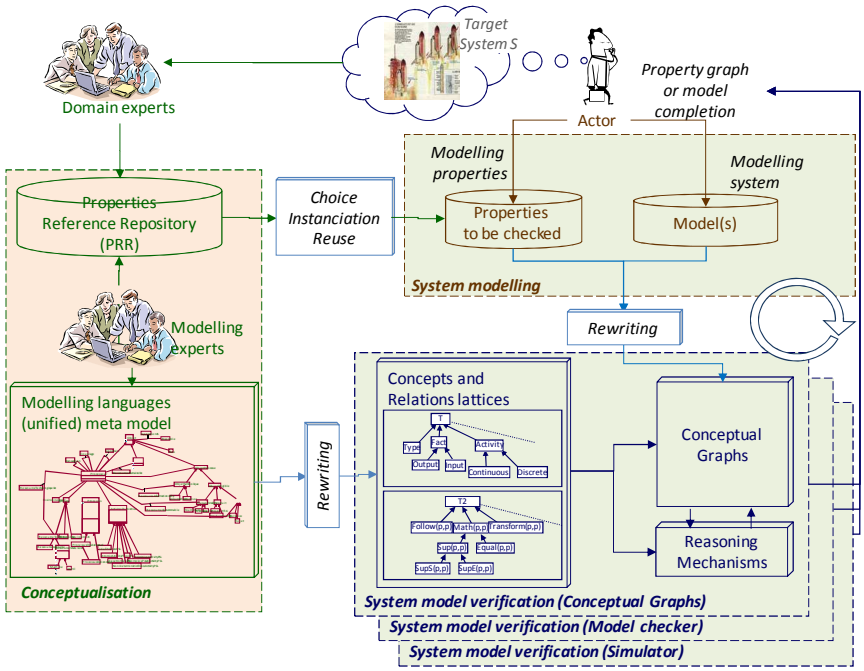

Fig. 2: UPSL principles

- The third phase consists to translate the system-of-interest relevant models (i.e. selected for formal proof) and the associated properties (i.e. properties associated to these models) to checking tools. This phase loops with previous phase regarding verification results. This aims to improve models and will end when all the properties provable from formal manner are verified.

Following this implementation approach, UPSL framework is proposed in Fig. 3 and its main elements are now presented.

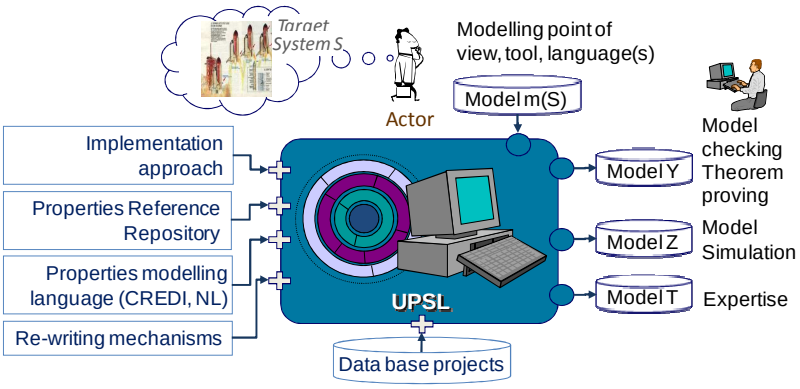

Fig. 3: a simplified view of the UPSL framework

Requirement model

Taking into account the model usage in SE process, a requirement having to be verified can consider the system-of-interest itself or its model(s):

- System requirements corresponding to customer and stakeholders needs (functional and non-functional: safety, performance, etc.) or induced by technological choices.
- Constraints induced by normative rules, natural laws of physics, etc.
- Model expectations: consistency (intra and inter models, languages and inter views required for describing the system-of-interest: functional, behavioural and organic views), completeness (partial) and if possible, rules having to be respected in order to begin validation of model relevance vs. system.

Usually, informal specifications technique are prefered in industry. Several formal requirements techniques (Easterbrook 2002) can be also used such as KAOS (Van Lamsweerde *et al*. 1998) which is very attractive for its possibility to define, analyse and reason about requirements. These formal approaches remain however difficult to use in conjunction or applied taking into system-of-interest model account. Proposed requirement model is based on a tree model containing one kind of node and two types of link as shown in Fig. 4. A node describes an expectation expressed considering a level of detail i.e. abstraction level at which the requirement is expressed. For an abstract level (by convention, more abstract level is from level 0), it is described by using natural language. At a high level of detail i.e. when system-of-interest models can be expressed, highlighting the behaviour, the functions or the structure of it, a node is then described by using CREDI modelling language presented in the next part. This permits to handle modeling variables, parameters and predicates extracted from the models.
Link between two nodes can be from first from one node N from a level D and a node N' from level immediately more precise D+1 (respecting level notation convention). In this case, link are associated to a conditional function by using logical operators in order to describe the role of the node N' in the verification of the node N. This allows decomposing a requirement in more precise requirements with potential alternatives. A final node, i.e. a requirement from the more elevated level of detail, can be:

– Temporized: requirement depends from time evolution.
– A-temporal: requirement characterizes only the structure or the functional aspect of the system without taking into account the time.
– Simple requirement: cause is empty and effect has to be checked in every case.
– Composite requirement: cause and effect are interacting.

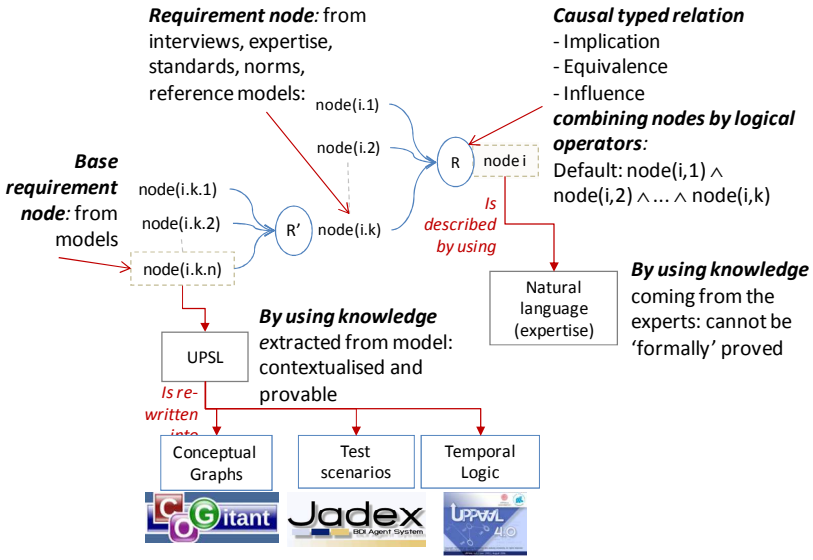

Fig. 4: requirement model principles and use

An example of a requirements tree is proposed in Fig. 5 following a classification into requirements concerning models, system-of-interest or project management.

CREDI property modelling language

A property is defined as knowledge (an expectation, a requirement or attribute definition and value) which characterizes a (set of) given model(s) of a system-of-interest and which must be respected in order to assumer the quality of this (these) model(s) during the project.

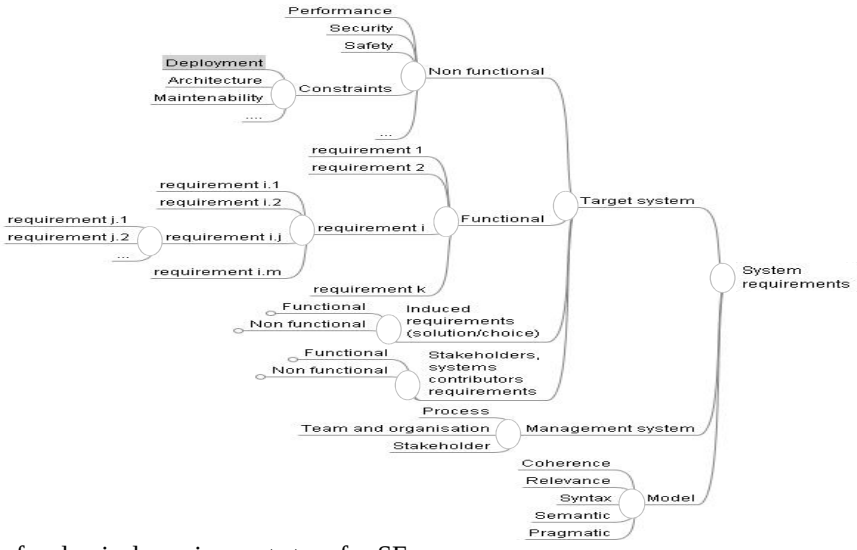

Fig. 5: example of a classical requirements tree for SE

A property aims then to model as formally as possible requirements and to support different proving approaches:

- Formally proof (that is to say respected by the model by using formal proof mechanisms – model checking, theorem proving - taking into account the modelled system characteristics as far as possible).
- Expertise
- Used in complement with other V&V techniques based on model execution i.e. simulation or emulation.

They are several property modelling languages such as temporal logics (LTL, TCTL...) (Emerson 1990) or PSL (Accelera 2004) for example. They remain much specialised and require some expertise to use. The goal of CREDI property modelling language is then to encourage us to take into account more precisely the aims of a property presented above and to make more effective their usage.
A property is described in a first way by using natural language to be more comprehensive. It is then modelled

by a composite entity made up with a group of causes (denoted C) linked up with a group of effects (denoted E) via a parameterised relation (denoted R). This relationship formally describes how the set of causes C induces a modification on the whole set of effects R by respecting the conditions described in R. R may be from a combination of the following types:

- Logical: the occurrence of C implies or is equivalent to the occurrence of E,
- Temporal: the occurrence of C must get there before or strictly after the occurrence of E,
- Emerging: C describes how different objects can interact in order to bring out E which may be observable at a lower level of abstraction but not directly deductible from the causes,
- Influence: C and E are linked together and each of the two sets depends on the other one. The influence must be interpreted as beneficial or harmful.

The degree D (optional) of shrewdness allows us to define the pointed out abstraction level of the model concerned by the property. Last the set I (optional) is composed of of a set of criterium which can be evaluated for characterising the thruhfulness of the property (in case of simulation or for guiding the expertise). The reader will find more detailed model and grammar of CREDI modelling language in (Chapurlat 2007). Last, a property can be from different types:

- Static (a-temporal): consistency and coherence of the model regarding its meta model, coherence between models (inter views and inter languages) and between levels of detail D, requirements independent from time evolution. In this case, a proof tool has been developed (Chapurlat et al. 2009) around Conceptual Graphs (Sowa 1984) with the Cogitant tool (Genest 2003)
- Dynamic (time / timed / event): behavioral expectations about the model bahaviour taking into account the operational semantics of the modeling language, time-dependent requirements about the expected behaviour of the system-of-interest. In this case, formal proof tools (Yahoda 2005) are used (for example, UPPAAL (Behrmann et al. 2004), SMV (McMilan 2000), SPIN (SPIN 2005), COQ (Coq 2005) or STEP (Bjorner et al. 1998)).

Properties Reference Repository

Using formal languages to specify properties remains difficult and sometimes even impossible without huge effort and practice from the engineer. The Properties Reference Repository aims then:

- **To dispose of existing formulation of properties**. These ones can be extracted from other studies. They can also be considered as generic. A generic property describes facts which are commonly recognised as relevant for given systems or models. It is formulated at a high level of abstraction independently of any scope but depending from system-of-interest modelling languages so the repository remains dedicated. The user can select, instanciate and set a generic property according to the model elements to be analyzed before proving it by using modelling variables, parameters and predicates issued from the model. The CREDI modelling language is used to describe properties: using only one language ensures consistency and uniqueness of representation in the PRR.
- **To structure the properties** into relevant user-defined classifications. The repository is then considered as a knowledge data base gathering and mapping a set of fundamental properties making these ones available for the engineer to support and organize its work. Last a repository remains dedicated to a given system modelling framework.

The repository can isolate three classes of properties

- **Axiomatic property**: property allowing to take into consideration part of a norm or standard, physical law, etc. which may then be considered as an axiom of knowledge having to be checked absolutely.
- **System property**: property which describes (part of) system requirements such as deployment constraints, geographical, architectural, performance, dependability, confidentiality, maintainability, etc.
- **Model property**: property that characterizes the meta model (language model-ling employee), the structure or semantic behavioural rules of the model itself. It is generally known properties of liveliness, safety, security and reachability. It allows for example to ensure consistency, synchronization, sequence, boundary, cycle...

A PRR is built and validated by a group of experts of the system-of-interest modelling framework by using for example different approaches to help the emergence of new properties: Brain Storming, Brain writing, mind map, five questions....

UPSL implementation approach

For each domain, a USPL framework must be developed following the approach described in the next.
1 - **Establish the needs of modeling** (concepts and relationships to be added to existing system modelling languages. This consists to define the purpose of the system-of-interest models analysis, its objectives (to limit the time required for the study (min, max), to achieve a level of confidence, to analyse only models coherence…), the used terms, concepts and relationships between the concepts and to formalize them in a meta-model (UML class diagram). This metamodel will be completed by taking into account potential modelling framework (multi languages and multi views which is used and the metamodel of each used modelling language. Some examples of models may be also required by experts in charge of this task. Last, the existing skills abilities of actors having to handle checking tools, the chosen tools and their input.output modelling language have to be determined. The metamodel is then enriched again with the CREDI metamodel and validated with the help of the experts.
2 - **Establish functional requirements for handling properties** taking into account the area covered. This induces to consider the following questions:

- How the actors wish to manipulate and prove properties?
- What interfaces actors suggest?
- What are modeling envrionement having to be taken into account (e.g. GMF in Eclipse, GME, CORE...)? It is then necessary to provide a meta model or grammar of language input / output of each modeler
- What are checking tools to take evidence into account (e.g. UPPAAL, STEP, Spin...)? As for the modelling environment, a meta model or grammar of language input / output of each tool is required.

3 - **Structuring / Organizing / Formalizing** data and manipulation mechanisms highlighted in the previous questions. .
4 – **Developing rewriting mechanisms** of system-of-interest enriched modelling languages and CREDI language to input language of chosen prover. This is done by respecting MDA principles as summarised in Fig. 6

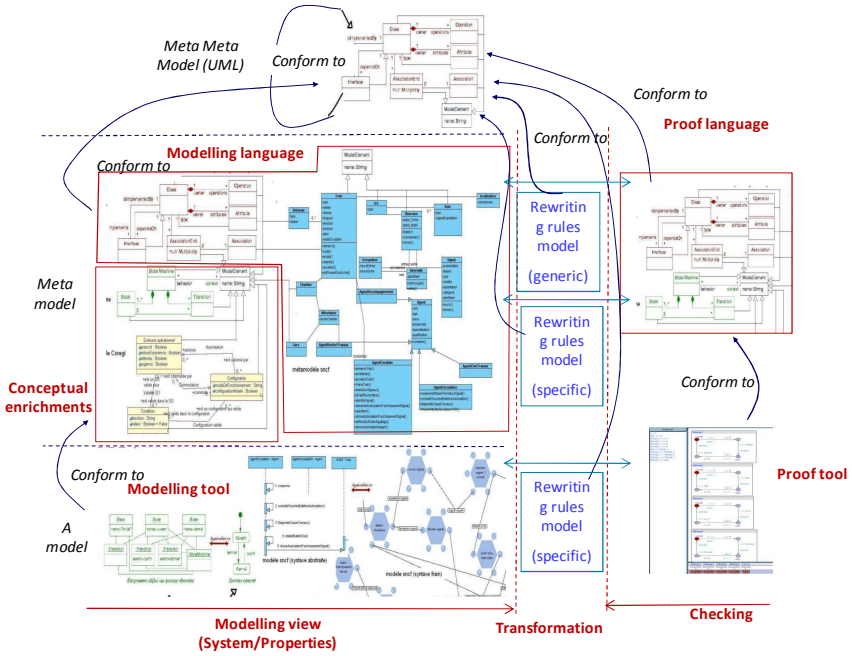

Fig. 6: model transformation principles

4 - **Proceed** with development of a dedicated domain platform UPSL
5 - **Design the dedicated PRR**.
Once the framework elements (modelling languages enrichment, PRR and re-writing mechanisms) and the domain platform UPSL available, a model study can be performed following 6 steps:

- **Framing**: this requires system modelling by using the enriched modelling languages and choosing the models having to be checked. The objectives of the verification study are then defined taking into account project objectives and constraints.
- **Specifying properties** by selecting and reusing (i.e. interpret, set) existing properties in the PRR or specifying them by using classical textual editor allowing to handle CREDI model.
- Analyze:
  - By proof. In this way, system-of-interest model(s) are rewritten into Conceptual Graphs (xxx) in case of

static properties or to model checkers if the model semantic corresponds to the input language of the tool in case of dynamic properties.

- By running the model if the modelling language provides an operational semantic i.e. a set of rules defining how the model can be simulated or emulated. In this way, a study under development concerns the development of a Multi Agents Platform based on JADEX (Rebai et al. 2009)
- By expertise if a property cannot be checked from a formal manner but can be verified by analysing the model behaviour and structure by an expert.

Conclusion and perspectives

UPSL offers different concepts and mechanisms that can be helpful for SE project verification tasks and particularly in Model Based System Engineering context. The main works currently under development consist to define interest and to apply these concepts on SysML and MODELICA, to define rewriting mechanisms to model checker UPPAAL and to set up a Reference Properties Repository for interoperability requirements (Chapurlat et al. 2009, Mallek et al. 2009).

Bibliography

(Accelera 2004) Accelera Formal Verification Technical Committee (FVTC), PSL Property Specification Language Reference Manual, Version 1.1 (see http://www.eda.org/vfv/), 2004

(Balci et al. 2002) O.balci, W.Ornwsby, Expanding our horizons in verification, validation and accreditation research and practice, 2002 Winter Simulation Conference, E. Yücesan, C.-H.Chen, J. L. Snowdon, and J. M. Charnes (éditeurs), 200

(Behrmann et al. 2004) Behrmann G., David A., Larsen K. G., A tutorial on Uppaal, Department of Computer Science, Aalborg University, Denmark, 2004

(Bérard et al. 2001) Bérard B., Bidoit M., Finkel A., Laroussinie F., Petit A., Petrucci L., Schnoebelen Ph. McKenzie P. Systems and Software verification: model checking techniques and tools, Springer, 2001

(Bjorner et al. 1998) N.Bjorner, Z.Manna, H.Sipma, T.Uribe, Deductive Verification of Real-Time Systems Using STeP, Technical Report STAN-CS-TR-98-1616, Computer Science Department, Stanford University, 1998

(CDT 2007) System Engineering guide book for transportation systems, California Department of Transportation, Division of research and transportation, 2007

(Chapurlat 2007) Chapurlat V., Vérification et validation de modèles de systèmes complexes : application à la modélisation d'entreprise, Research Habilitation Direction manuscript, Montpellier University, France, 2007 (in French)

(Chapurlat et al. 2009) Chapurlat V., Roque M., Interoperability constraints and requirements formal modelling and checking framework. Advances in Production Management Systems, APMS 2009, Bordeaux, France, 2009

(Coq 2005) The Coq Proof Assistant - Reference Manual, Version 8.0, Janvier 2005 (see http://coq.inria.fr/)

(Easterbrook 2002) S.Easterbrook, Introduction to Formal Modeling in Requirements Engineering, 10th Joint International Requirements Engineering Conference, in Essen, Germany, 2002

(Emerson 1990) E.Emerson, Temporal and modal logic, Handbook of Theoretical Computer Science, vol. B. MIT Press. Editeur: J. van Leeuwen ISBN 0262220393, pp. 955-1072, 1990

(Genest 2003) D.Genest, CoGITaNT Version-5.1 : Reference manual (in French), 2003

(INCOSE 2007) INCOSE, System Engineering (SE) Handbook Working Group, System Engineering Handbook, A « How To », Version 3.1, Guide For All Engineers, 2007

(ISO 2002) ISO/IEC 15288: 2002(E) – Systems engineering – System life cycle processes

(Love et al. 2000) G.Love, G.Back, Model Verification and Validation for Rapidly Developed Simulation Models: Balancing Cost and Theory, Project Performance Corporation, 2000

(Mallek et al. 2009) Mallek S., Daclin N., Chapurlat, Developing interoperability in collaborative process: an Anticipative Effects-Driven Approach, Advances in Production Management Systems, APMS 2009, Bordeaux, France, 2009

(McMilan 2000) K.L.McMilan, The SMV System for SMV Version 2.5.4: SMV Manual (see http://www-2.cs.cmu.edu/~modelcheck/smv.html), 2000

(NASA 2001) NASA, VV&A Recommended Practices Guide - Glossary, 2001

(Rebai & al. 2009) A.S.Rebai, V.Chapurlat, System interoperability analysis by mixing system modelling and MAS: an approach, ATOP/AAMAS, International Workshop on Agent-based Technologies and applications for enterprise interoperability, Eighth International Joint Conference on Autonomous Agents & Multi-Agent Systems, Budapest, Hungary, 2009

(RPG 2001) RPG, V&V Techniques, RPG reference Document (accessible à l'adresse http://vva.dmso.mil/), DMSO (Defence Modelling and Simulation Office), 2001

(Sowa 1984) J.F.Sowa, Conceptual structures: information processing in mind and machine, New York (U.S.A.): Addison-Wesley, 1984

(SPIN 2005) SPIN Primer and Reference Manual, Addison Wesley, ISBN 0-321-22862-6, 2005

(Van Lamsweerde 2002) A.Van Lamsweerde, Formal Specification: a Roadmap, The Future of Software Engineering, A. Finkelstein (ed.), ACM Press, 2002

(Van lamsweerde et al. 1998) Van Lamsweerde, A., Darimont, R. & Letier, E. Managing conflicts in goal-driven requirements engineering. IEEE Transactions on Software Engineering, 24(11): 908-926, 1998.

(Yahoda 2003) Formal verification tools overview web site (see http://anna.fi.muni.cz/yahoda/), 2003