Protocol engineering using UML*

Claude JARD**
Jean-Marc JÉZÉQUEL**
Alain LE GUENNEC**
Benoit CAILLAUD**
and the UML group*

Abstract

Despite the irresistible growth of interest in formal methods and related validation and verification tools, the development of distributed systems seldom relies on them. We claim this is mainly due to formal methods lack of support for modern software life-cycles. The construction and maintenance of open distributed systems are mostly based on object-oriented software development. We investigate how frameworks may help to embed formal validation techniques in an object-oriented process based on the UML notation. We show how standard model-checking techniques can be used right now on UML models by exploiting informations available in class and deployment diagrams, and using an operational semantics of statecharts. We also present how the behavioural views of UML, including sequence or collaboration diagrams could be consistently managed using a common semantics model called BDL (standing for "behavioural description language") with which the various behavioural views of UML can be translated into one another. BDL is a reactive synchronous language with a true concurrency semantics. Basic object interactions are represented in BDL by partially ordered sets of events and the behaviour of a complete (or incomplete) system is expressed by composition of basic interactions. BDL offers new perspectives for a flexible verification of systems by modeling them as globally asynchronous networks of locally synchronous software components.

Key words: Software engineering, Communication protocol, Distributed system, Object oriented method, Validation, Simulation language, Semantics.

Contents

I. Introduction
II. Simulation techniques for validation and verification of distributed software
III. The unified modeling language
IV. Bringing validation in the oo life-cycle
V. Applying validation techniques
VI. Towards a semantic pivot for UML
VII. Conclusion
References (22 ref.)

* This paper presents a synthesis of a collective work at IRISA with many contributors. This group is composed of A. Benveniste, B. Caillaud, H. Canon, WM. Ho, C. Jard, J.-M. Jézéquel, A. Le Guennec, F. Pennaneach, J.-P. Talpin. This research is partially funded by Alcatel.
** IRISA, Campus de Beaulieu, F-35042 Rennes Cedex, France, Email: First name. Last name@irisa.fr, Web: www.irisa.fr/pampa

RÉSUMÉ

En dépit de l'intérêt croissant pour les méthodes formelles et leurs outils de validation et vérification associés, le développement des systèmes répartis les ignore le plus souvent. Cela est dû, d'après les auteurs, principalement à leur non-intégration dans les cycles modernes de développement logiciel. La construction et la maintenance des systèmes répartis ouverts sont pour la plupart fondés sur un développement. L'article étudie un cadre de conception UML (unified modeling language) pour équiper le processus de développement objet avec des outils de validation formelle et montrer comment des techniques de validation classiques peuvent être utilisées dès maintenant sur des modèles UML en exploitant des informations contenues dans les diagrammes de classes et de déploiement, et en utilisant une sémantique opérationnelle des diagrammes d'état. Il présente aussi comment les vues comportementales de UML, incluant les diagrammes de séquence et de collaboration, pourraient être traitées de façon cohérente en utilisant un modèle sémantique commun appelé BDL, permettant les traductions des différentes vues entre elles. BDL est un langage réactif synchrones fondé sur une sémantique de vrai parallélisme. Les interactions entre objets de base sont représentées en BDL par des ensembles partiellement ordonnés. Le comportement global est obtenu par composition des interactions de base. BDL ouvre de nouvelles perspectives de validation de systèmes formés de composants locaux synchrones, mis en interaction assyntrohne.

Mots clés : Génie logiciel, Protocole communication, Système réparti, Méthode orientée objet, Validation, Langage simulation, Sémantique.
I. INTRODUCTION

The techniques currently used in the development of distributed software on communication networks most often lead to flimsy and poorly maintainable software. Numerous software engineering methods are proposed to tidy up this practice. They have to cover the different phases starting from preliminary requirement specifications, then design phases and down to the use of software tools to perform verification, validation, test and code generation. A true protocol engineering craft based on mathematically founded methods has been consolidated during these past ten years. It relies on languages with formally defined semantics allowing for a whole set of provably correct transformations. In parallel to this consolidation, the development of general, non-distributed software promoted the use of object-oriented design and programming. In that context, the interest for the seamless oo development process has somewhat eclipsed formality. This can be seen as a retreat in the development of complex critical systems. The challenge is thus to introduce formal methods as soon as possible in the development process and to keep using them all along the process. Most formal description techniques (FDT for short) designed for protocol engineering cannot be easily used in an integrated oo life-cycle. They all have specific semantic models, suited for a given particular step of development. Their use does not prevent "model ruptures" (breaches in the continuity of models) which are detrimental to the soundness and efficiency of the development process. This explains the growing interest for designing "frameworks", specialized for telecommunication software, and providing a set of formal engineering tools. The present paper adheres to this timely attitude.

Object design methodologies are now gathered in the UML notation ("unified modeling language" [4]), which is becoming a standard and diffuses at high speed in industrial circles.

The objective of this paper is to cast a method and present tools which extensively re-use several formal validation techniques developed during the few past years in the context of protocol engineering and to apply them to an oo design method based on UML.

The plan of the paper is the following: We start by briefly recalling the techniques of validation based on simulation and their situation with respect to the necessary integration in a software life-cycle centered on object-oriented design. Then we present the current state of UML. The third part shows how to generate a simulation code by refinement. We then discuss the way of connecting the validation tools. The last section is devoted to the presentation of BDL and the prospects it opens.

II. SIMULATION TECHNIQUES FOR VALIDATION AND VERIFICATION OF DISTRIBUTED SOFTWARE

II.1. A set of complementary formal techniques

Basically, the designer may attack his/her software by three complementary techniques. We list here their advantages and major drawbacks:

- **formal verification of properties**: it gives a definite answer about validity by formally checking that all possible executions of the specification of the distributed software respect some properties (e.g. no deadlock). But existing methods, such as model-checking, which often imply the construction of the graph of all the states the distributed system could reach, can only be easily applied to the analysis of very simplified (abstracted) models of the considered problem [15].

- **intensive simulation**, using a simulated (and centralized) environment: it can deal with more refined models of the problem and can efficiently detect errors (even tricky or unexpected ones) on a reasonable subset of the possible system behaviours. Formally, it consists in randomly walking the reachability graph of the distributed software. The main difficulty is to formally describe and simulate the execution environment.

- **observation and test** of an implementation: here, the execution environment is a real one. But since there is a lack of tools to observe a distributed system as a whole, it will be difficult to actually validate the software. Anyway, producing the test cases for the distributed system is a costly task that can be alleviated only if one is able to automatically generate the tests from a formal specification of the system and a set of test purposes.

All these approaches are complementary. Most of these techniques have been developed in the context of the formal description techniques (FDTs) for protocols, where they have been successfully applied to various real problems. They are now disseminating in small industrial niches, mainly in the context of development of hardware or critical systems.

II.2. Difficulties in using FDTs

It is very disappointing to see that formal validation based on standard FDTs (such as SDL [9], Estelle [17] and LOTOS [16]) never acceded to a widespread use in the industry, despite excellent results on most of the pilot