SHE'S Project: a Report of Joint Works on Integration of Formal Specification Techniques

Michel Allemand[†], Christian Attiogbé[†], Pascal Poizat[‡], Jean-Claude Royer[†] and Gwen Salaün[†]

[†]IRIN, Université de Nantes
2 rue de la Houssinière, B.P. 92208, 44322 Nantes Cedex 3, France
email: {allemand,attiogbe,royer,salaun}@irin.univ-nantes.fr

[‡]LaMI - UMR 8042, Université d'Évry Val d'Essonne
Tour Évry 2, 523 Place des terrasses de l'Agora, 91000 Évry, France
email: poizat@lami.univ-evry.fr

1 Introduction

The use of formal specifications is now widely admitted but there are still some difficulties related with it. One of them is to specify complex systems by combining several formal notations. Indeed, the use of a single formalism is not sufficient enough to specify such systems; different aspects of complex software systems, such as data or concurrency, have to be specified using different notations, and have to be verified with suitable tools. Two major concerns with such systems are firstly to ensure the consistency between the different aspects, and secondly to provide specification and proof guidelines.

The SHE'S[1] project focuses on formal specifications using different formalisms. In the sequel, the main terms used to describe such specifications are *integrated*, *mixed*, or *heterogeneous* specifications. The SHE'S project started in 1999, and this paper aims at presenting the current state of our researches. One goal is to study several possibilities to integrate formal specifications expressing distinct aspects of systems. That leads to define the formal foundations of specification languages, the techniques to verify systems, and the tools to support them. Here, we do not wish to present a precise proposal, but rather give an overview of all our works. Therefore, the reader may consult the references to have more precise details.

[1] Specifications of HEterogeneous Systems

The remainder of this paper is organized as follows. Section 2 describes the Korrigan model mixing algebraic specifications, symbolic transition systems and temporal logic. Section 3 focuses on Configuration Machines, an integrated approach combining Z-based formalism and transition systems. Section 4 overviews current work around a general formal framework to combine process algebras and algebraic specification languages. Section 5 shows applications to make the use of UML more rigorous. Finally, a discussion is undertaken to draw up a balance sheet of the different proposals gathered in the project.

2 Korrigan: Combination of Algebraic Specifications, Symbolic Transition Systems and Temporal Logic

Korrigan is devoted to the structured formal specification of mixed components. Its aim is also to fill the gap between formal specifications and software engineering. Therefore, Korrigan provides the specifier with a model and the corresponding specification language equipped with an operational semantics [8], mechanisms to define reusable components with both textual and graphical notations inspired by UML [9], and a specification methodology for mixed specification languages with a software environment for verification and prototyping through specification translation and object oriented code generation [10].

The Korrigan model is based on structures we call *views* that integrate *symbolic transition systems* (STSs), algebraic specifications, and a form of temporal logic. The model comprises three different means for structuring specifications. The basic aspects (static and dynamic) of the components are defined within the *internal structuring*. An important feature of Korrigan is the use of specific STSs that extend the Graphical Abstract data Types ones (see Section 5) with dynamic terms on transitions (taking inspiration from LOTOS and SDL). Our STSs are used to define aspects at a high level of abstraction (no state explosion) using very readable (hence acceptable in a software engineering environment) notations. Moreover, our STSs are combined with algebraic specifications to relate behaviours and communication protocols with the abstract definition of an encapsulated datatype. The different kinds of composition (integration of aspects and concurrent composition of communicating components) are defined in a unified way within the *external structuring*. The use of both axioms and temporal logic makes Korrigan a very expressive language to define the gluing of reactive components. Moreover, communication patterns

with different kinds of communication and concurrency semantics may be defined. Finally, components may be reused through a simple form of *inheritance structuring*.

3 Configuration Machines: Integration of Z and Transition Systems

The *Configuration Machines* [4] approach integrates, in a homogeneous framework supported by natural deduction, model based specifications of data and transition system models of behaviours. In the other approaches of the project we consider algebraic specifications for the data part; here this part is treated with Z. The current approach falls into a series of works [22, 12] combining model-oriented formalisms, especially Z with process algebras CCS or CSP. However, in our approach we avoid fixing the behaviour of components using process algebras; instead, we use an approach comparable to *Actions Systems* [7] where the behaviour of components depends on guards.

The specification unit of our formalism is a *machine type*, *machine* for short. It gathers both data and behaviour parts, and describes the type of components having the same data space (described as a *configuration*) and the same behaviour. The behaviour of a machine is expressed as a set of guarded transition rules (called *transactions*) relating the descriptions of initial and final configurations. A machine evolves repeatedly from one configuration to another one following enabled transactions. Larger machines (or systems) are built by extending or composing other machines. Composed machines communicate synchronously, and interact via (parameterised) events exchange. Invariants are used to define global safety properties of systems. Compared to existing works, our contribution, apart from the definition and the composition of a specification unit which gathers both data and behaviours, resides in the use of very abstract states and guarded transitions to specify the behaviour part. It avoids the description of the whole set of states and thus the state explosion problem known for large systems. The mechanization of this approach is studied through embedding into frameworks which provide tool support for transition systems. Ongoing works focus on machines embedding into SAL/PVS [5] and *Event-based B*.

4 Generic Combination of Process Algebras and Algebraic Specifications

In a first attempt, we suggest a formalism [21] combining the process algebra CCS with CASL, the unified language of algebraic specifications. The central aim of our combination is an extension of Milner's value passing CCS. In a second step, we wish to take into account more languages so that the specifier may freely choose his/her preferred specification languages. Indeed, (s)he could prefer a formalism for different reasons: suitability with reference to the requirements, existence of development tools, or level of expertise. Accordingly, we extend our previous work to make a *generic combination* between any process algebra and any algebraic specification language possible [20]. This framework is based on a *formal kernel* gathering the essential constructions of the involved languages. These approaches are *control driven*, *i.e.* the process algebra (called the *main* formalism) gives the behaviour of the full specification. The algebraic specification language is the *secondary*, since it is used to define independent data types. Algebraic terms are handled by the processes described with the process algebra.

However, this approach encompasses some limitations which are the lack of powerful tools to verify our specifications, and the restriction to consider only integration of two basic languages. Recent works are devoted to embed our formal kernel into languages underlying higher-order logic tools like PVS to make proofs on the global specification. Furthermore, we work on another approach which allows the combination of heterogeneous specification modules [19]. This proposal defines and formalises the formal foundations of a gluing language (similarly to Korrigan in Section 2) to connect specification modules.

5 Applications to Object-Oriented Analysis and Design

One current area of application of these researches is the Unified Modelling Language (UML) which is a mixed language but lacks of formalisation[2]. We have done several experiments in this area using two formal models based on algebraic specifications. The first model is an abstract model to describe classes in an algebraic way. This model, the *Formal Class Model* [1], is used to present

[2]see the pUML group at `http://www.cs.york.ac.uk/puml/home.html`

static UML classes and also to make the translation of formal specifications into object-oriented languages easier. The second model, called *Graphical Abstract data Type* (GAT), is a STS [13] coupled with an algebraic specification of a partial abstract data type.

Currently, we propose in [2] a formalisation of the main elements of static diagram. Amongst the UML diagrams, we focus on the class and object diagrams since they are central to UML modelling. We use Larch Prover, a rewrite-rule based prover for algebraic specifications, to prove properties and to check the consistency of the generated specifications. A complementary work [16] presents an approach to translate dynamic UML models into formal specifications. This work deals with static classes, but also active classes, Statecharts and collaboration diagrams. We suggest to complete these UML diagrams with formal notations in order to get formal UML components in a more abstract and formal way than UML-RT. Once the UML diagrams are designed, a tool translates them into algebraic specifications that are the input language of the Larch Prover tool. This enables us to prove data type properties as well as temporal logic formulas using Larch Prover [18].

6 Discussion

The SHE'S project aims at improving the use of several formal languages when specifying complex software systems. We have completed the first step of the project in which we have experimented several possible approaches of integrated specifications. All our proposals share common concepts and advantages. Firstly, they allow the specification of the two major aspects appearing in systems: the data and the dynamic aspects, which are modelled using suitable notations. Secondly, within all our proposals, the basic languages are integrated within well-defined semantics; this is essential for the consistency of the whole system, but also enable proof techniques and reuse of verification tools. Finally, our approaches are user-friendly in the sense that they have been designed to make their use by developers easier.

Despite these common points, our approaches are differentiated in their structuring choices or underlying formal foundations. Firstly, Korrigan sets the priority on the separation of concerns according to the nature of specification units. This leads to independent specification modules and simplifies their reuse. Opposite proposals are those of Section 3 and 4 in which basic specification units gather different aspects within one module. That impacts on different topics, especially the reuse of components and the reuse of tools dedi-

cated to a specific aspect. A second way to compare our works is the definition (or not) of a global semantics. Except for the Section 5, *ad-hoc* semantics are defined for all our approaches. This leads to expressive and readable languages, but raises difficulties to reuse existing tools and to perform proofs on specifications written with these integrated formalisms. The last proposal uses an homogeneous context (algebraic specifications) to embed the different aspects. Thus, reuse of tools (LP in this case) is straightforward. Thirdly, the basic languages chosen in the different integrations are quite various. Nonetheless, three of the four experiments combine algebraic specifications for the data part whereas Configuration Machines experiments Z for this aspect. We emphasize that algebraic specifications are chosen to remain in an abstract context and to favour properties verification, while the approach with Z is rather oriented towards formal development. Finally, the proposal of Section 4 introduces a possible freedom in the choice of languages that is not allowed in the other proposals. It is quite interesting for developers to have this freedom. We claim that the choice of formalisms is a real issue because each specifier has his/her preferences.

We may also relate our approaches with well-known existing ones. On the one hand, some approaches for integrated specifications have been studied in the area of algebraic specifications [11, 3, 14]. On the other hand, many other works use model-oriented specifications, like Z or B, with process algebras or transition systems [12, 6, 22, 23]. A comprehensive description and comparison with related works would be too long; yet, the reader may refer to [17] for a more detailed state of the art.

After this balance sheet in our project, ongoing works head for two directions. The first one consists of improving the way to write specifications using composition of formal components. The interests are manifold: modelling the different aspects of systems with suitable languages, using an easy and formal language to glue modules, making the reuse of modules and dedicated tools easier. Korrigan is an interesting proposal in this way, and the general approach introduced at the end of Section 4 caters for these advantages and is one of the main direction in the following of our project. Tool aspects are the other concern of current work in our team. Indeed, we have proposed several integrated languages, but most of them have little or no verification possibilities. To fill this gap, we adopt the reuse of general purpose tools based on higher-order logic, such as PVS [15]. They possess very expressive input logic to embed specification languages. Accordingly, translations into this kind of tools need to be formalised, and proof guidelines may be defined.

References

[1] P. André, D. Chiorean, and J.-C. Royer. The Formal Class Model. In *Joint Modular Languages Conference, Modula, Oberon & friends*, pages 59–78, Germany, 1994.

[2] P. André, A. Romanczuk, J.-C. Royer, and A. Vasconcelos. Checking the Consistency of UML Class Diagrams Using Larch Prover. In T. Clark, editor, *Proceedings of the third Rigorous Object-Oriented Methods Workshop (ROOM'00)*, BCS eWics, 2000.

[3] E. Astesiano, M. Broy, and G. Reggio. Algebraic Specification of Concurrent Systems. In E. Astesiano, B. Krieg-Brückner, and H.-J. Kreowski, editors, *IFIP WG 1.3 Book on Algebraic Foundations of System Specification*, chapter 13, pages 467–520. Springer-Verlag, 1999.

[4] C. Attiogbé. The Access Control System: Specification with Configuration Machines. In *Proceedings of AFADL'2000*, pages 203–220, France, 2000. In french.

[5] S. Bensalem, V. Ganesh, Y. Lakhnech, C. Muñoz, S. Owre, H. Rueß, J. Rushby, V. Rusu, H. Saïdi, N. Shankar, E. Singerman, and A. Tiwari. An Overview of SAL. In C. M. Holloway, editor, *Proceedings of the Fifth NASA Langley Formal Methods Workshop (LFM'00)*, pages 187–196, USA, 2000.

[6] R. Büssow, R. Geisler, W. Grieskamp, and M. Klar. The μSZ Notation, Version 1.0. Technical report, Teschnische Universität Berlin, 1997.

[7] M. J. Butler and C. C. Morgan. Action Systems, Unbounded Nondeterminism, and Infinite Traces. *Formal Aspects of Computing*, 7:37–53, 1995.

[8] C. Choppy, P. Poizat, and J.-C. Royer. A Global Semantics for Views. In T. Rus, editor, *Proceedings of the International Conference on Algebraic Methodology And Software Technology (AMAST'2000)*, volume 1816 of *LNCS*, pages 165–180. Springer Verlag, 2000.

[9] C. Choppy, P. Poizat, and J.-C. Royer. Formal Specification of Mixed Components with Korrigan. In *Proceedings of the 8th Asia-Pacific Software Engineering Conference (APSEC'01)*, IEEE Computer Society Press, China, 2001.

[10] C. Choppy, P. Poizat, and J.-C. Royer. The Korrigan Environment. *Journal of Universal Computer Science*, 7(1):19–36, 2001. Special issue on Tools for System Design and Verification.

[11] H. Ehrig and F. Orejas. Integration and Classification of Dynamic Types and Process Specification Techniques. *EATCS Bulletin*, 53, 1998.

[12] C. Fischer. How to combine Z with a process algebra. In J. P. Bowen, A. Fett, and M. G. Hinchey, editors, *Proceedings of the 11th International Conference of Z Users (ZUM'98)*, volume 1493 of *LNCS*, pages 5–23, Germany, 1998. Springer-Verlag.

[13] M. Hennessy and H. Lin. Symbolic Bisimulations. *Theoretical Computer Science*, 138(2):353–389, 1995.

[14] ISO/IEC. LOTOS: A Formal Description Technique based on the Temporal Ordering of Observational Behaviour. ISO/IEC 8807, ISO2, 1989.

[15] J. Crow, S. Owre, J. Rushby, N. Shankar, and M. Srivas. A Tutorial Introduction to PVS. In *Proceedings of the Workshop on Industrial-Strength Formal Specification Techniques (WIFT'95)*, USA, 1995.

[16] L. Peng, A. Romanczuk, and J.-C. Royer. A Practical Translation of UML Components into Formal Specifications. In G. Schellhorn and W. Reif, editors, *Proceedings of the TOOLS East Europe Conference*, 2001.

[17] P. Poizat. *Korrigan: a Formalism and a Method for the Structured Formal Specification of Mixed Systems.* PhD thesis, Institut de Recherche en Informatique de Nantes, Université de Nantes, 2000. In french.

[18] J.-C. Royer. Formal Specification and Temporal Proof Techniques for Mixed Systems. In *Proceedings of the 6th International Workshop on Formal Methods for Parallel Programming: Theory and Applications (FMPPTA'01)*, IEEE Computer Society Press, USA, 2001.

[19] G. Salaün, M. Allemand, and C. Attiogbé. An Approach to Combine Heterogeneous Specification Modules. Technical Report IRIN 01.7, University of Nantes, 2001.

[20] G. Salaün, M. Allemand, and C. Attiogbé. Formal Framework for a Generic Combination of a Process Algebra with an Algebraic Specification Language: an Overview. In *Proceedings of the 8th Asia-Pacific Software Engineering Conference (APSEC'01)*, IEEE Computer Society Press, China, 2001.

[21] G. Salaün, M. Allemand, and C. Attiogbé. Specification of an Access Control System with a Formalism Combining CCS and CASL. In *Proceedings of the 7th International Workshop on Formal Methods for Parallel Programming: Theory and Applications (FMPPTA'02)*, IEEE Computer Society Press, Florida, USA, April 2002. To appear.

[22] G. Smith. A Semantic Integration of Object-Z and CSP for the Specification of Concurrent System. In J. S. Fitzgerald, C. B. Jones, and P. Lucas, editors, *Proceedings of the 4th Formal Methods Europe International Conference (FME'97)*, volume 1313 of *LNCS*, pages 62–81. Springer-Verlag, 1997.

[23] H. Treharne and S. Schneider. Using a Process Algebra to Control B OPERATIONS. In K. Araki, A. Galloway, and K. Taguchi, editors, *Proceedings of the 1st International Conference on Integrated Formal Methods (IFM'99)*, pages 437–457, UK, 1999. Springer-Verlag.