TRANSPUTER COMMUNICATIONS, VOL. 1(1), 3–15 (AUGUST 1993)

Deriving parallel programs using SANDS tools

D. BUCHS, P. RACLOZ
Swiss Federal Institute of Technology, DI, 1015 Lausanne, Switzerland
M. BUFFO, J. FLUMET, E. URLAND
University of Geneva, CUI, 24, rue du Général Dufour, 1211 Genève 4, Switzerland

SUMMARY
This paper describes the techniques and the tools developed to construct CO-OPN specifications (Concurrent Object Oriented Petri Nets) [1] and to derive parallel programs from these specifications [2]. CO-OPN is a specification language which permits an abstract description of aspects of concurrent operations and data-structures of computer programs. The concurrent and operational part of the formalism is described by means of Petri nets, while the data-structures are described by means of algebraic abstract data types. In the CO-OPN formalism, theses associations are structured with the notion of objects. The CO-OPN specifications can be developed, analysed and simulated in its dedicated environment called SANDS (Structured Algebraic Nets Development System [3,4]). A special tool included in SANDS, TTool for Transformation Tool, has been developed and is used for obtaining parallel or distributed programs from original specifications, written in CO-OPN. TTool is based on sequences of transformations which use rewriting techniques and the notion of equivalency between specifications. Our methodology extracts concurrent behaviours from functional descriptions expressed by means of algebraic specifications.

KEY WORDS Formal specifications Prototyping Specification transformations Development tools Parallelization Object oriented development

1 INTRODUCTION

At present, one of the major difficulties faced in software engineering is the production of correct parallel or distributed products. Indeed, there is a gap between the requests of users for correct and efficient solutions to their problems and the hardware producers who offer a large scale of parallel machines. The usual approach in reducing this gap is to use ad-hoc methods. On the contrary, we propose a technique which can be applied indiscriminately to the target implementation requirements by means of a library of transformations. This technique is based on a sound theoretical background which guarantees the correctness of the final product.

The project CO-OPN (Concurrent Object Oriented Petri Nets) was undertaken in order to propose a formal specification language for the design of large, parallel systems. The latter were created by means of using a modular specification language in combination with Petri Nets. The underlying formalism, called hierarchic algebraic nets, used as a semantical description of CO-OPN language, can be seen as a modular extension of algebraic nets. In

1070–454X/93/010003–13$11.50

Received 12 January 1993
Revised 4 February 1993

CO-OPN, hierarchic algebraic nets modules are called objects. Therefore, the aim of the CO-OPN specification language is to assure the production of software of quality by means of a complete development method based on the life cycle, including the specification and implementation phases.
This paper describes the environment named SANDS developed to construct CO-OPN specifications but emphasizes on its new components, a transformation tool and a library of transformations for the parallelization of specifications. The transformations have a broad field of applications. However, in this paper they were defined in order to be applied in the area of parallelism. Each transformation step, by introducing parallelism, leads closer to implementation and respects some equivalency criteria, i.e. the original specification and its transformation are behaviourally equivalent (e.g. bisimulation equivalence relation [5,6]). It is worth mentioning that the process of transformation may be accomplished semi-automatically in the user guidance.
The paper is structured as follows. Section 2 gives a description of the CO-OPN specification language, the following section 3 introduces the SANDS environment, and section 4 presents the transformation technique and its tool is illustrated with a simple example. Concluding remarks can be found in section 5.

2 THE SPECIFICATION LANGUAGE: CO-OPN

2.1 Basic ideas

The CO-OPN specification language is based on two elegant concepts, *Algebraic data types* and *Petri nets*, which are merged into a more general model *Algebraic nets* [7]. In order to express efficiently certain operations on large instances, the standard constructors on data types, iterators on algebraically specified arrays, structuring capabilities and the genericity principle (both for objects and algebraic data types) are also included in CO-OPN.

2.2 Concepts

The formal model is based on an approach in which the encapsulation notion is ensured by methods and abstraction is supported by synchronization expressions between the methods. A module is an algebraic net, with parametrized transitions as methods and the possible markings of the places as states.

According to our terminology, the events which represent the evolution of the net can be of two types: *methods* or *internal transitions*. The methods of a module are activated externally whilst internal transitions are activated locally, as soon as their conditions are validated. The events modify the state of the system and the symbolic links between the modules express synchronization constraints between the methods or the internal transitions of these modules. The constraints are described by a combination of sequential, simultaneous, alternative or empty synchronization operators.
In summary, our formalism is based on the following components:

- Algebraic abstract data types
- Petri nets
- Modularity and synchronizations

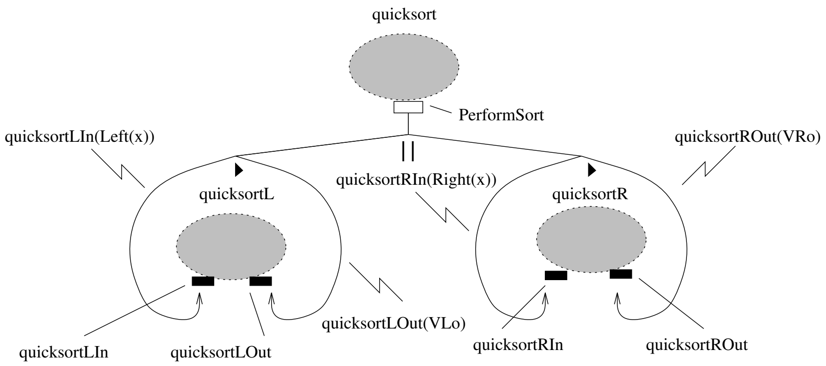

Figure 1. An abstract view of a CO-OPN application: the quicksort example

From a graphical point of view, a modelling can be seen at two different levels. The first level is an abstract view of the application which represents the CO-OPN objects with synchronizations between them (see Figure 1). The second level represents the algebraic Petri nets of each object which contains places, transitions and methods with parameters (see Figure 3). A method is the entry point for some synchronizations with other objects.

3 THE ENVIRONMENT: SANDS

The tools embedded in SANDS (see Figure 2) have been developed for covering all steps in the development of CO-OPN specification. It includes a graphics editor which handles the objects and their organization, a compiler to check and to produce parallel code, in accordance with the semantics of the system, a monitor for testing some part of the application, a verifier of temporal properties and a transformation tool. This transformation tool is presented in the following section.

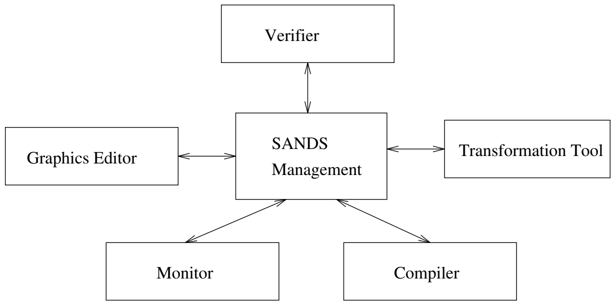

Figure 2. The SANDS environment

3.1 The graphics editor

The modularity of the specification language is supported by the graphics editor. The user defines the global organization of the objects (see Figure 1), their external methods and the synchronization links between them. A graphics window is associated with each object (see Figure 3), allowing the editing of its internal behaviour which is described by an algebraic net.

All the rules of the CO-OPN language specification are handled by means of graphics. Clarity can be maintained by avoiding information overloads. Algebraic specifications can be integrated by the editor and some operations can be executed on these. In this way, control of syntax and semantics may be accomplished during the graphics development. The graphics editor is built with the metaCASE tool Graphtalk[1]

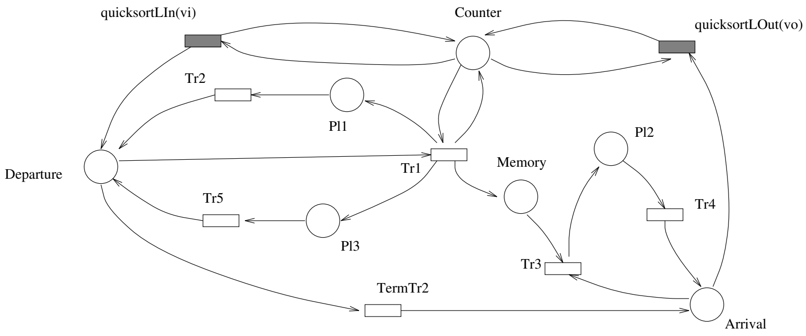

Figure 3. An algebraic Petri net: the simplified view of the "quicksortL" object (obtained after a transformation)

3.2 The simulation phase

CO-OPN specifications can be either simulated using the monitor or executed on a MIMD machine. In both instances, the CO-OPN code will have to be processed by a common front-end which performs lexical, syntactical and static semantic analysis. Then, in a case where a simulation is desired, specifications will be translated into Prolog. The resulting code can be loaded into the monitor to receive a highly interactive evaluation of the application.

3.2.1 The compiler

If the user wishes to have a more efficient execution of his specifications, he can take advantage of the MIMD compiler, which generates parallel C or standard C within the PVM environment, according to the target platform: a transputer (Volvox) or a distributed machine, ranging from a network of workstations to the Cray T3D supercomputer. The compiler accepts only a subset of the CO-OPN language. Thus, certain rules concerning the level of abstraction of the specifications have been established in order to gain maximal

[1] Graphtalk is a product of the Xerox company

performance during their execution. The idea is that the compiler is to be used to generate code for simplified CO-OPN systems obtained through the transformation process.

3.2.2 The monitor

The reachability graph of the system is built during its simulation. When there is no simulation in process, the states graph can be explored. Different facilities are offered to improve this exploration.

The environment provides two main means for interaction: a step-by-step evolution and an evolution controlled by the expressions of synchronization between the objects in the system. In a step-by-step evolution, the enability of the methods of the objects are tested in the current state of the system. In a synchronized evolution, a constraint expression, involving events (methods or transitions) of objects, in association with notions of sequentiality, parallelism and alternatives is specified by the user.

The development of the graph depends on the choice a particular state from which its subsequent successors may be computed. The choice is left to the user or may be selected at random by the monitor. The choice of the state from which the reachability graph is developed is totally free, i.e. it can be chosen from the set of already-computed states. Therefore, a development of a branch of the graph can be abandoned for the benefit of another.

During simulation, the user can also test the behaviour of the abstract data type part of the system, i.e. he can evaluate a term expression where variables may occur. Equality between two expressions, possibly with free variables, can also be proved by using resolution techniques. Roughly speaking, the evaluation is performed through the orientation of the axioms.

Due to the use of variables within the terms of the contents of places, a marking can be viewed as a generic marking, i.e. each instantiation of a free variable generates a particular marking. This ability also has an impact on the reachability graph (see Figure 4).

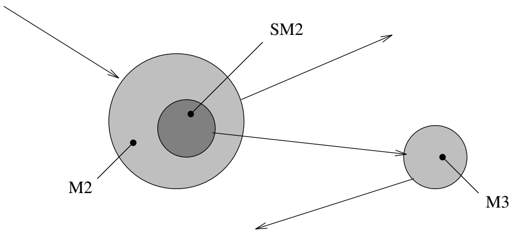

Figure 4. The generic marking SM_2 is the submarking of M_2 such that it enables the step to the generic marking M_3

In some cases, a synchronization expression may produce a very large or infinite number of successors. If needed, some constraints can be specified in order to overcome this difficulty. Parameters such as depth, parallelism and width set a limit on, respectively, the number of steps from the current state, the number of transitions fired simultaneously in a single step, and the number of successors of a marking (see Figure 5).

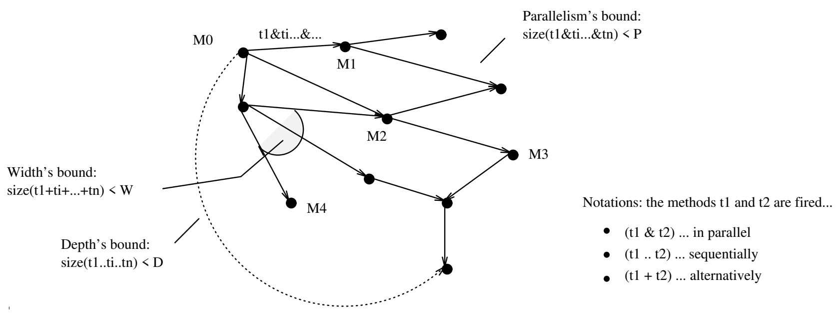

Figure 5. Bounds (P, W, D) associated with a synchronization evaluation

3.3 The verifier

A method for proving temporal properties of systems modelled by the CO-OPN formalism has been developed [8,9]. The temporal properties are expressed with a branching time logic [10]. These properties deal with the occurrence of events which, for us, correspond to the firability of a transition/method in a net. Our method is based on the notions of predicates (sets of satisfiable states), predicate transformers and symbolic checking. Thus far, modular modellings are not handled by the verifier tool, it can be applied to usual Petri nets [11] and has some extensions to algebraic nets.

4 THE TRANSFORMATION TOOL

4.1 The principles

Prior to the description of the transformation tool package, we must introduce some definitions and notations.
Let S be a set of specifications and $\sim$ an equivalence relation on $S \times S$. Let $T = S \times S$ and $T_c \subseteq T$, such that $(a, b) \in T_c \Leftrightarrow a \sim b$, where $a, b \in S$. The set T is the set of *transformations*. A transformation $t \in T$ is *correct* if $t \in T_c$. The application of the transformation $(a, b) \in T$ is the transformation step which passes from the specification a to the specification b. Further, we define *transformation rules* and *transformation methods*. The transformation rules are a generic way of transforming a specification. Their semantics are based on re-writing systems. A transformation rule R induces a family of transformations $T_R \subseteq T$. We say that a transformation rule R is *correct* if and only if $T_R \subseteq T_c$. The transformation methods permit the general description of a sequence of transformation steps. Their semantics are based on Petri nets. The use of transformation methods has the advantage of grouping simple transformations into more complex ones, with the use of a common methodology [2].
The overall process of the transformations is illustrated in Figure 6 , in which each step consists of the simple rewriting of a part of the specification.

The equivalence between two specifications is the main condition for the acceptance of a transformation step as being correct. This condition depends on the choice of the equivalence relation $\sim$. A overly restrictive equivalence criteria can induce a bad implementation while, on the other hand, a overly permissive one can introduce significant differences in

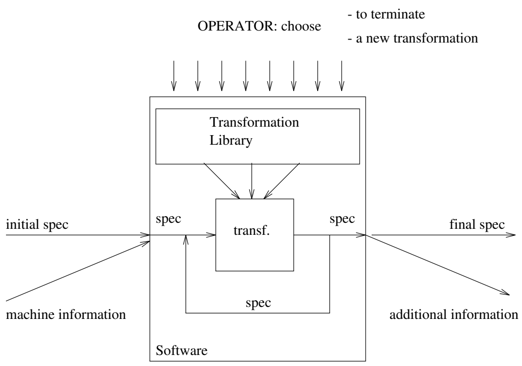

Figure 6. Transformation step scheme

the solution obtained. The family of observational equivalences [5,6] seems to be adequate for this purpose.
The adequacy to the chosen execution platform is realized semi-automatically by the choice of a hardware model which can be a real machine architecture, as well as a theoretical model. This choice guides the selection of the transformations during the overall process.
The original specification does not contain any algorithmic descriptions. Such construction will be introduced during the transformation process. According to the original specification and the target hardware model, the suitable algorithmical skeleton of a family of solutions for the original problem is deduced. The refinement of the solution, that is the further iteration of transformation steps, will then restrict the set of the possible solutions into the final one. This development scheme also includes efficiency aspects and parallelism of the solution obtained.
The transformation process, supported by a user-friendly interface, is performed step by step and provides the following functions:

- stops the transformation process at an arbitrary level,
- backtracks if the derived specification is not satisfactory,
- chooses another available, correct transformation,
- evaluates the adaptation of the current specification according to the execution platform,
- performs a new transformation on the current specification.

4.2 The transformation library

The transformation library is used by the transformation tool during the transformation process. From this point of view, the tool is an open system with respect to further improvements and developments, accomplished by simply adding new transformation rules and methods to the transformation library. The current library contains modules of transformation rules and modules of transformations methods. The transformations found in this library cover the following algorithmics principles [2]:

- branch and bound
- divide to conquer
- probability
- searches in graphs (dfs,bfs,simple backtracking,. . .)
- simplifications (binary search, tree pruning, . . .)
- approximations

4.3 Example

As an illustration of our methodology, we describe briefly how to reach a parallel efficient solution for the quicksort problem, using a virtual session of the SANDS system. Another example, illustrating the TSPP technique with more details, can be found in [6].
We start using the graphics editor, and create the first formal description of the problem, composed of two modules. The first one describes the data structure. The second module describes the root process, which invoke the above function.
We then use the monitor to confirm the correctness of the newly-created modules. We start with the evaluation of several algebraic terms, in order to verify the exactitude of some functions, after which we begin a simulation session of the whole system. This monitor session leads us to the certitude that our system is correct with respect to the first informal description of the problem we wish to solve.
Now we must refine our specification and reach a concrete algorithmic description of the solution. This goal is achieved by using the TTool package, which introduces in a semi-automatic way parallel, distributed and possibly probabilistic aspects into the solution, and which guarantee the correctness of the result with respect to its first formal description. This TTool session itself can be split into three phases.

- During the first phase, the abstract problem is analysed, and convenient and efficient algorithm is found. In our case, the “divide to conquer” technique can be applied and the proper algorithm is derived.
- In the second phase, a concrete form to this algorithm is given (see Figure 7), creating the processes we must have depending on to the hardware of the execution platform. In our example, a computation tree with 13 processes (among these are 7 terminal processes, i.e. the leaves of the tree, and one is the root) is created. This phase is performed according to the chosen algorithm of the precedent phase.
- During the last phase, each process is optimized separately according to the real processors in which they will take place. In our case, assuming we want to execute the result in a transputer hypercube, generic low-level optimizations for MIMD machines are performed. Thus, a concrete parallel specification, ready to be compiled and executed, has been computed.

The compiler is then used to reach concrete executable code. In our example, we ask it to compile for 7 processors (the leaves of the computation tree) in a transputer hypercube,

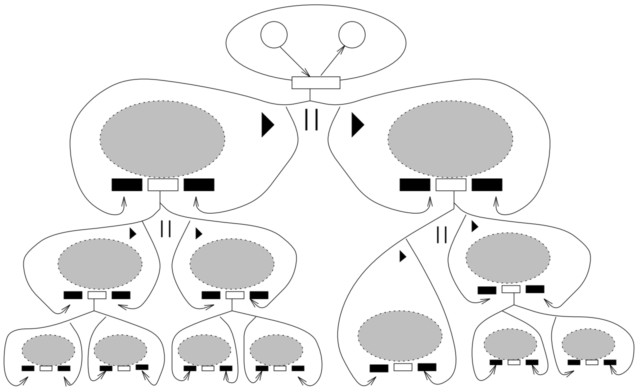

Figure 7. Resulting Petri net model with unbalanced distributed CO-OPN objects

and to monitor the execution and suitable input/output (see Figure 7).
Following the execution of the program, we can finally monitor and analyse the transputer session, using the Paragraph software [12]. This software provides several displays for analysing the behaviour of our program, such as a clock, a Gantt chart, a space-time communication display, a network display, a processor utilization summary, and so on.
Figure 8 shows the speed-up (as a function of the number of processors, with respect to the original, unstructured problem) concretely obtained with our 32-transputers hypercube for our quicksort example.

# processors	2	4	8	16
speed-up	1.3	2.6	6.1	11

Figure 8. Speed-up obtained

This result seems to be promising, due to the fact that for such specific simple examples, where computation time is lower than communication time, multiplication of the processors leads, in fact, to multiplication of useless communications. Nevertheless, an almost linear speed-up has been achieved.

5 CONCLUSION

By way of conclusion, we stress that future research has been planned with respect to CO-OPN and SANDS.
The current version of the CO-OPN language will be expanded with full object-oriented features, as proposed in [13,14] into a new specification language termed CO-OPN/2. The use of the category theory for new semantic models based on compositionality and the development of an operational model for object-oriented specification languages which

takes into account the fault tolerance aspects of the distributed systems are also under consideration. As the transformation tool is an open system, we continue the design of new transformation rules and methods which cover a large spectrum of algorithmical principles. Some probabilistic and approximation approaches are also under consideration.
Future work with the MIMD compiler is directed towards developing a prototyping tool and a method adapted to parallel applications. In particular, it would be very useful to provide the user with an opportunity to mix, at any level of the software life-cycle, tool-generated compilation units with a hand-written code. The expected benefits would be multiple: higher efficiency, better user interface, instant realization of integration tests, ideal support for systematic refinement schemes. This research direction will certainly lead to converting the compiler to a C++ and Ada95 code generator.
The other future efforts will be focused on the applications of temporal proofs to algebraic nets and on the improvement of operational semantics for the algebraic nets.

REFERENCES

[1] Didier Buchs and Nicolas Guelfi, 'A Concurrent Object Oriented Petri Nets approach for system specification', in *12th International Conference on Application and Theory of Petri Nets*, ed., M. Silva, pp. 432–454, Aahrus, Denmark, (June 1991).

[2] Mathieu Buffo, 'Collection de transformations (associées au logiciel TTool)', Cahiers du CUI 79, Centre Universitaire Informatique, CUI, Université de Genève, 24, rue du Général Dufour, 1211 Genève 4, Suisse, (September 1993).

[3] Didier Buchs, Jacques Flumet, and Pascal Racloz, 'SANDS: Structured Algebraic Net Development System', Cahiers du CUI 71, University of Geneva, (1993).

[4] Didier Buchs, Jacques Flumet, and Pascal Racloz, 'SANDS: Structured Algebraic Net Development System', in *14th International Conference on Application and Theory of Petri Nets, Tool presentation abstracts*, ed., Buy Ugo, pp. 25–29. Chicago, USA, (June 1993).

[5] Robin Milner, Prentice-Hall International Series in Computer Science, 1989.

[6] Mathieu Buffo, Didier Buchs, and José Rolim, 'Developing parallel programs based on rewriting techniques', in *Proceedings of the IEEE/USP Workshop on High Performance Computing 1994*, ed., João Zuffo, pp. 185–200, São Paulo, Brasil, (march 1994). IEEE / USP, USP-LSI. CDU 681.322.06.

[7] Wolfgang Reisig, 'Petri nets and algebraic specifications', in *Theoretical Computer Science*, volume 80, 1–34, Elsevier, (1991).

[8] Pascal Racloz, *Symbolic proof of temporal properties for various kinds of Petri nets*, Ph.D. dissertation, University of Geneva, April 1994.

[9] Pascal Racloz and Didier Buchs, 'Symbolic proof of CTL formulae over Petri nets', in *8th International Symposium on Computer and Information Sciences*, eds., G. Levent, O. Raif, and G. Erol, pp. 189–196. Istanbul, Turkey, (November 1993).

[10] E.A. Emerson, *Temporal and Modal Logic*, Elsevier, 1990.

[11] Pascal Racloz, 'A temporal specification checker for Petri nets', in *IEEE International Conference on Systems, Man and Cybernetics (Invited Session on Progress in Petri Nets and Applications), To appear*, Vancouver, Canada, (1995).

[12] M.T. Heath and J.A. Etheridge, 'Visualizing performance of parallel programs', Technical Report ORNL/Tm-11813, Oak Ridge National Laboratory, (1991).

[13] Olivier Biberstein and Didier Buchs, 'An Object-Oriented Specification Language based on Hierarchical Algebraic Petri Nets', in *Working papers of the international Workshop on Information System Correctness and Reusability IS-CORE'94*, eds., R. Wieringa and R. Feenstra, pp. 47–62, Amsterdam, The Netherlands, (September 1994). vrije University.

[14] Nicolas Guelfi, *Les réseaux algébriques hiérachiques: un formalisme de spécifications structurées pour le développement de systèmes concurrents*, Ph.D. dissertation, Université Paris-Sud, LRI, September 1994.