Submodule construction tool[1]

J. Drissi*, G. v. Bochmann**
* Dept. IRO, Universite de Montreal, CP. 6128, Succ. Centre-Ville, Montreal, H3C 3J7,Canada, Phone: (514) 343-6161, Fax: (514) 343-5834,drissi@iro.umontreal.ca
** School of Information Technology & Engineering, University of Ottawa, Colonnel By Hall (A510), P.O.Box 450 Stn A,Ottawa,Ont.,K1N 6N5, Canada, Phone : (613) 562-5800 ext. 6205, Fax 562-5175 email:bochmann@site.uottawa.ca

Abstract. Using the programming language Java, we developed the Submodule Construction Tool, which implements algorithms for the submodule construction problem. The submodule construction problem (SCP) is to construct the specification of a submodule *X* when the specification of the system and all submodules but *X* are given. This problem is encountered in the hierarchical design of complex systems, in the synthesis of controllers and in the reuse of components. The tool requires a number of files as input and produces a file containing the result. For the input/output Finite State Machine model, we can obtain the generic solution, the minimal solution with respect to the number of states, we can check if a given FSM is a solution and we can find the minimal reduction of a given observable nondeterministic FSM. For the I/O automata model, we can find the generic solution for the safe realization relation and the subtype relation, we can check if a given I/O automaton is a safe realization or a subtype of another I/O automaton, we can compose I/O automata, we can find the resulting I/O automaton after hiding a subset of the alphabet and finally we can obtain the minimal trace equivalent I/O automaton.

1. Introduction

One common problem, encountered in the hierarchical design of complex systems, in the synthesis of controllers and in the reuse of components, is the submodule construction problem, also called factorization problem or equation solving problem. The submodule construction problem (SCP) is to construct the specification of a submodule *X* when the specification of the system and all submodules but *X* are given. Such a problem may be formulated mathematically by the equation $(C||X)\mathrm{R}A$, where *C* represents the specification of the known part of the system, *X* represents the specification of the component to be designed, *A* represents the specification of the whole system, || is a composition operator and R is a conformance relation. The SCP was first formulated and treated in [3], where specifications are expressed in terms of execution sequences, and trace equivalence was used as conformance relation. In [4], the author uses Milner's Calculus of Communicating Systems to model the same problem. Many other works [5] [6] have been done using labelled transition systems as a model for the specifications and the strong and/or the observational equivalences as conformance relations. In [1], we consider this problem in the context of the input/output Finite State Machine model (I/O FSM) [7]. The direct application of an approach based on the LTS model is not possible since the solutions obtained are not in general I/O FSMs. We have to add constraints on the environment behavior to obtain the system's behavior in the form of an I/O Finite State Machine. We have developed a method for constructing all the solutions when the specifications are given in the form of deterministic completely specified input/output Finite State Machines and the trace equivalence relation is used as conformance relation. This work was generalized in [8] to the case where the specifications are given in

1. This work was partially supported by the NSERC Strategic grant SRTGP200 "Methods for the systematic testing of distributed software systems" and an NSERC Research grant.

the form of nondeterministic completely specified input/output Finite State Machines and the reduction relation is used as conformance relation. In [2], we generalize our previous work by dealing with nondeterministic partially specified input/output Finite State Machines and by using other criteria such as complete trace equivalence, quasi-equivalence and reduction of nondeterminism. For this purpose, we consider partial I/O automata for systems specification, which is more general than input/output Finite State Machines. An I/O automaton corresponding to a given input/output Finite State Machine can always be obtained by unfolding each atomic input/output transition s-x/y->s' of the input/output Finite State Machine into two consecutive transitions s-x->s" and s"-y->s' of the corresponding I/O automaton.

This paper is structured as follows. In Section 2, we present the theoretical background used to develop the Submodule Construction Tool (SCT). In Section 3, we give a description of the tool. Section 4 illustrates with examples the use of the tool. Finally, in Section 5 we conclude the paper.

2. Theoretical background

The I/O automata model is defined in [9]. An I/O automaton (briefly *IOA*) *A*, is a 5-tuple (S_A, I_A, O_A, T_A, s_{oA}) where S_A is a finite set of states with s_{oA} as the initial state, I_A is a non-empty, finite set of inputs, O_A is a non-empty, finite set of outputs with $I_A \cap O_A = \emptyset$ and $T_A \subseteq S_A \times (I_A \cup O_A) \times S_A$ is a transition set. An element (s, u, s') in T_A is denoted by $s\text{-}u\text{->}s'$. If for each s in S_A and all x in I_A there exists s' in S_A such that $s\text{-}x\text{->}s'$, then *A* is said to be completely specified or input-enabled; otherwise A is partially specified. A fundamental property of the model of I/O automata is that there is a very clear distinction between those actions that are performed under the control of the automaton and those actions that are performed under the control of its environment. An automaton's transitions are classified as either "input" or "output".

In typed object-oriented languages the notion of subtype, that is, a conformity relation between types, is defined. A type P conforms to another type Q if P provides at least the operations of Q (P may also provide additional operations). Moreover, the types of the results of P's operations must conform to the types of the results of the corresponding operations of Q. Finally, the types of the arguments of Q's operations must conform to those of P's operations [10]. The idea behind the notion of subtype is the ability to use an instance of a subtype of a type T whenever an instance of type T is required to do a job.

While the subtyping relation of object-oriented languages is mainly concerned with the available operations and the types of their parameters, we are concerned with the dynamic behavior of objects, that is, of I/O automata, considering the allowed sequences of input and output operations. We define a subtype relation for the same purpose as in object-oriented languages, i.e. the possibility to replace any subsystem by an instance of its subtypes without changing the system's behavior.

When composing a collection of partial I/O automata, problems due to unspecified reception may appear when a receiving *IOA* does not have an input transition originating from the present state when the sending *IOA* executes a corresponding output transition. A composition of a collection of I/O automata is said to be safe if it does not contains unspecified receptions [11][12].

We define the safe realization of an *IOA* A by a composite *IOA* $B=B_1||B_2||...||B_n$, as follows: for any environment modeled by an *IOA* E, if the composition of A and E is safe then the composition of B_1, B_2,..., B_n and E is also safe. The safe realization criterion does not allow us to enforce mandatory output behaviors in certain given states, i.e. an *IOA* B which is a safe realization of an *IOA* A, will accept all the inputs accepted from the initial state of A and may produce no output. To deal with the problem of mandatory output behaviors and also to be able to represent the set of solutions to the equation $(C \,||X)R A$ as the set of subtypes of a particular type (or model) in the same modeling framework, we enhance the model of I/O automata by allowing the imposing of conditions on the set of traces from a given state. We call such a model an "I/O automata with optional complete traces". With each state *s* we associate two sets: the first set contains sets of traces from *s* such that at least one trace in each set must be present in any implementation of this automaton; the second set is a subset of the

traces from *s* and each time an execution, starting in *s*, is a prefix of an element of this set, the execution should progress to complete a trace in this set. We assume that an implementation is a normal (partial) *IOA*. We introduce a progress property which formalizes the preceding notion. Moreover, by requiring safe realization and realization of the progress property, we obtain the subtype relation. We show that the subtype relation is a generalization of the well known criteria trace equivalence, complete trace equivalence, quasi-equivalence and reduction.

Since in a composition of a collection of I/O automata the sets of inputs are not disjoint, we generalize the architecture (Figure 1) by allowing the component that will be designed, to observe some interactions between the environment and the context. This is done by adding to the equation a constraint on the required set of inputs of the solution. For the two criteria, safe realization and subtype, we propose for each an algorithm that produces an *IOA* solution to the equation ($C \| X$) R A under the constraint I_X=In if such a solution exists. We prove that the set of possible solutions is then either the set of safe realizations or the set of subtypes of the obtained solution, depending on which criterion was in the equation.

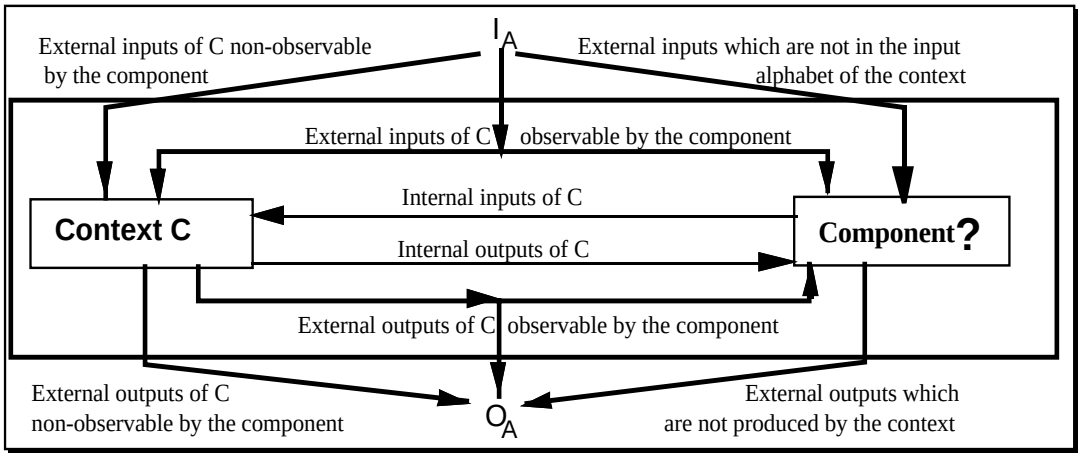


Figure 1: Composition architecture.

3. Description of the Submodule Construction Tool

The object-oriented programming language Java is used to implement the algorithms in [1][2]. The main goal of the tool is the generation of the generic solution for the submodule construction problem for specifications given as I/O FSMs or as I/O automata. A generic solution is a solution from which we can derive all the solutions. For the I/O FSM model, the operations present in the tool are :

- the determination of the generic solution for deterministic completely specified FSMs where the relation used is trace equivalence,

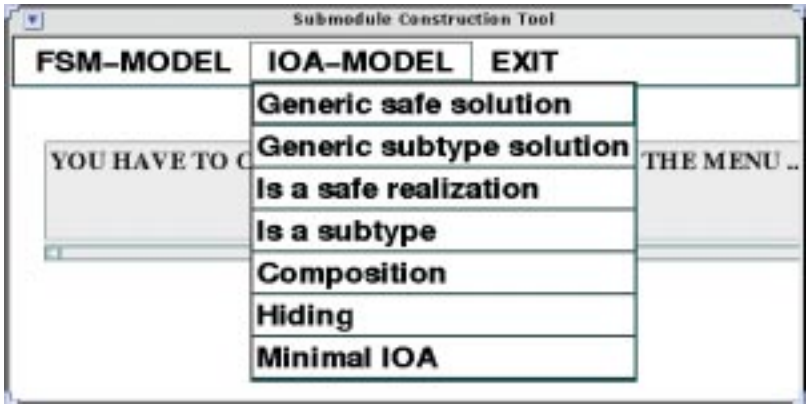


Figure 2: Tool interface

- the determination of the generic solution for nondeterministic completely specified FSMs where the relation used is reduction,
- the determination of the generic solution for nondeterministic partially specified FSMs where the relation used is quasi-equivalence,
- the derivation of the minimal solution with respect to the number of states,
- we can check if a given FSM is a solution of our equation,
- we can find the minimal reduction of a given observable nondeterministic FSM.

For the I/O automata model, the operations present in the tool are :

- we can find the generic solution for the safe realization criterion and the subtype criterion,
- we can check if a given *IOA* is a safe realization or a subtype of another *IOA*,
- we can compose I/O automata,
- we can find the resulting *IOA* after hiding a subset of the alphabet,
- we can obtain the minimal trace equivalent *IOA*.

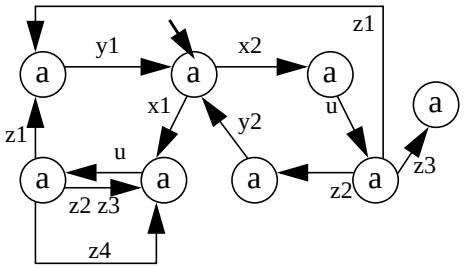
The tool is composed of a number of Java classes, the most important are :

- Menu class, it implements the user interface of the tool.
- IOAutomaton class, it deals with the representation of an *IOA* and the operation on IO automata like the composition, the hiding of actions and the minimization.
- IOAWOCT class, it deals with the representation of an I/O automaton with optional complete traces and the associated operations like the composition, the hiding of action and the deletion of prohibited traces.
- ReadFile and OutFile classes which are concerned with the transformation between the external and the internal representation of the specifications.
- SafeR class which implement the steps of the algorithm for the construction of the generic solution for the safe realisation relation.
- Subtype class which implement the steps of the algorithm for the construction of the generic solution for the subtype relation.
- IsSafeR and IsSubtype classes which allow to check if a given specification is a solution to a given equation.

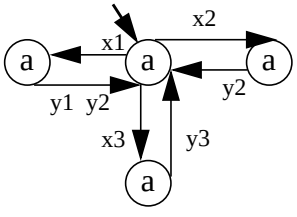
More informations and a copy of the tool can be obtained from the following Web site address : http://www.iro.umontreal.ca/~drissi.

4. Illustration of the use of the tool

In the following, we will illustrate with an example the work of the tool. We consider the IO automata model, and we resolve the equation ($C \| X$) RA where R represents the subtype relation. We consider for the context the *IOA C* of Figure 3 with I_C={x_1, x_2, z_1, z_2, z_3, z_4} and O_C={u, y_1, y_2}, for the whole system the IO automata with optional complete traces A corresponding to the IOA_A of Figure 3 with I_A={x_1, x_2, x_3} and O_A={y_1, y_2, y_3}, with the following constraints on the outputs in the states of A, MT_A={(1, Ø), (2, {{y_1, y_2}}), (3, {{y_2}}), (4, {{y_3}})}, an element (s, Y) of MT_A means that at least one element in Y is a mandatory output in the state s. We note that in the case of this example the subtype relation reduces to the reduction relation. The set of inputs required for the solution is In={x_1, x_3, u}, which means that the component to be designed will observe x_1 but not x_2, y_1 and y_2.



The context C



The whole system IOA$_A$

Figure 3: The specification of the context and the whole system.

In this case, the tool requires three files as input : the first file contains the specification of the context, the second file contains the specification of the whole system and the third file contains the required alphabet for the solution. Figure 4 illustrates the syntax of the preceding files.

```
// The specification of the context
//INPUT SET
x1;
x2;
z1;
z2;
z3;
z4;
END;
//OUTPUT SET
u;
y1;
y2;
END;
//STATE SET
a;
b;
c;
d;
e;
f;
g;
END;
//TRANSITION SET
a - x1 - b;
b - u - c;
c - z1 - d;
c - z2 - b;
c - z3 - b;
c - z4 - g;
d - y1 - a;
a - x2 - e;
e - u - f;
f - z1 - d;
f - z2 - g;
f - z3 - h;
g - y2 - a;
END;
```

```
// The specification of the whole system
//INPUT SET
x1;
x2;
x3;
END;
//OUTPUT SET
y1;
y2;
y3;
END;
//STATE SET
A1;
A2;
A3;
A4;
END;
//TRANSITION SET
A1 - x1 - A2;
A1 - x2 - A3;
A1 - x3 - A4;
A2 - y1 - A1;
A2 - y2 - A1;
A3 - y2 - A1;
A4 - y3 - A1;
END;
//OPTIONAL COMPLETE TRACES
END;
//MANDATORY TRACES
STATE 2: CHOIX : y1; y2;
STATE 3 : CHOIX : y2;
STATE 4 : CHOIX : y3;
END;
```

```
// The specification of the required input set
x1;
x3;
u;
END;
```

Figure 4: Illustration of the syntax of the input files required by the tool.

Since in this case the solution exists, the tool will display the message in Table 1 :

THE SPECIFICATION OF THE GENERIC SOLUTION IS IN FILE : SUBTYPEGENSOL.txt
THE INPUT ALPHABET IS : {x_1, x_3, u}
THE OUTPUT ALPHABET IS : {z_1, z_2, z_3, z_4, y_3}
THE NUMBER OF STATES IS : 5

Table 1: The message displayed by the tool

The file containing the specification of the generic solution *Sol* is in the current directory, it have the same syntax as the file containing the specification of the whole system. In our example the obtained generic solution is shown in Figure 5, and the sets of constraints are :
MT_{Sol} ={(**p**, {uz_2}), (**t**, Ø), (**q**, {$u(z_2u + z_3u)^*z_1$, $u(z_2u + z_3u)^*z_4$}), (**s**, {y_3}), (**r**, Ø)}, and
OCT_{Sol} ={(**p**, {uz_2}), (**t**, Ø), (**q**, {$u(z_2u + z_3u)^*z_1$, $u(z_2u + z_3u)^*z_4$}), (**s**, {y_3}), (**r**, Ø)}.

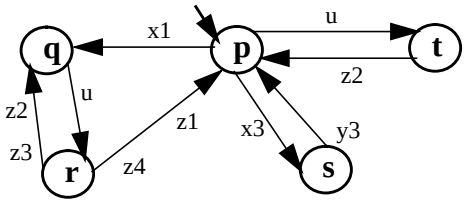


Figure 5: The IO automaton corresponding to the generic solution *Sol*.

We have used the tool to derive a controller for a system defined in [13]. We have also used the tool to derive the specification of the protocol entity Responder for the INRES protocol [14].

5. Conclusion

We have presented in this paper the Submodule Construction Tool. This tool was developed using the programming language Java. It implements algorithms for the submodule construction problem for the input/output Finite State Machine model and the I/O automata model for various conformance relations. For the input/output Finite State Machine model, we consider the trace equivalence, quasi-equivalence and reduction conformance relations. For the I/O automata model, we consider the safe realization and subtype conformance relations. The subtype relation is a generalization of the trace equivalence, quasi-equivalence and reduction conformance relations, while the safe realization relation is implied by all the above conformance relations. We continue experimenting with the tool in other application cases. We presently work on extending the tool to deal with Timed I/O automata model.

References

[1] J. Drissi, N. Yevtushenko, A. Petrenko and G. v. Bochmann, *On The design of a submodule based on the input/output FSM model*, Technical Report No. 1120, University of Montreal, April 1998.

[2] J. Drissi and G. v. Bochmann, *Submodule construction for systems modelled by I/O automata*, 1998 (in preparation).

[3] P.Merlin, G. v. Bochmann, *On the Construction of Submodule Specifications and Communication Protocols*, ACM Trans. on Programming Languages and Systems, Vol. 5, No. 1, pp. 1-25, Jan. 1983.

[4] M. W. Shields, Implicit System Specification and the Interface Equation, The Computer Journal, Vol. 32, No. 5, 1989.

[5] H. Qin and P. Lewis, *Factorisation of Finite State Machines under Strong and Observational Equivalence*s, Journal of Formal Aspects of Computing, Vol. 3, pp 284-307, July-Sept. 1991.

[6] E. Haghverdi and H. Ural, *An Algorithm for Submodule Construction*, Technical report of the Department of computer Science, University of Ottawa, 1996.

[7] A. Gill, *Introduction to the theory of Finite-State Machines*, Mc Graw-Hill Book Company, Inc, 1962.

[8] A. Petrenko and N. Yevtushenko, *Solving* asynchronous equations, To appear in the proceeding of FORTE/PSTV'98, Paris, 1998.

[9] N. A. Lynch and M. R. Tuttle, *AN INTRODUCTION TO INPUT/OUTPUT AUTOMATA*, MIT/LCS/ TM-373, Laboratory for computer science, Massachusetts Institute of Technology, Nov. 1998.

[10] A. Black, N. Hutchinson, E. Jul, H. Levy and L. Carter, *Distribution and Abstract Types in Emerald*, IEEE Transaction on Software Engineering, vol. SE-13, no. 1, pp. 65-76, January 1987.

[11] R. Negulescu and J. A. Brzozowski, *Relative Liveness : from intuition to automated verification*, Research report CS-95-32, University of Waterloo, Canada, 1995.

[12] S. G. H. Kelekar George W., *Synthesis of protocols and protocol converters using the submodule construction approach*, Proceedings of Protocol Specification, Testing and Verification, XIII, A. Danthine, G. Leduc, P. Wolper (Editors), 1994.

[13] Akira Fusaoka, Hirohisa Seki, Kasuko Takahashi, *A description and Reasoning of Plant Controllers in Temporal Logic*, Central Research Laboratory, Mitsubishi Electric Corporation, Amagasaki, Hyogo, Japan, 1986.

[14] D. Hogrefe, *OSI formal specification case study: the Inres protocol and service*, Institut fur Informatik, Univeritat Bern, Switzerland,1992.