A TOOL FOR THE SYNTHESIS OF DISCRETE CONTROLLERS FOR DISTRIBUTED SYSTEMS

Vincenza Carchiolo, Alberto Faro, Michele Malgeri

Istituto di Informatica e Telecomunicazioni
Facolta' di Ingegneria - Universita' di Catania
Viale A. Doria 6, CATANIA - Italy
tel. + 39 95 339449, telex 970255 UNIVCT I

ABSTRACT

This paper deals with the direct synthesis of controllers for distributed systems. In particular, the paper takes into account the controllers belonging to the process class known as "discrete event processes". The discrete event processes are mainly characterized by the asynchronous and non-deterministic behaviour. Here, the authors present a tool for the automatic synthesis of the controllers. The key concept of the synthesis methodology is the use of a formal description technique as starting point for the direct synthesis. The formal technique used by the tool is an extension of Milner's CCS.

1. INTRODUCTION

In the last decade a great interest has been devoted to Formal Description Techniques (FDTs) and a great effort has been made in order to define FDTs tailored for specifying discrete event systems. That is the systems that are discrete, asynchronous and possibly non-deterministic.

On the other hand, in parallel with the evolution of FDTs suitable for specifying distributed systems, a lot amount of work has been devoted to develop automated tools for making easy the application of such FDTs.

The use of FDTs for modelling discrete event systems has been successfully investigated and many FDTs have been proposed to cover this item. The more popular of them are Petri Net (PNT), Temporal Logic (TPL), Hoare's CSP (MIL), Trace Theory (SNE), and Milner's CCS (MIL).

Moreover, ad-hoc FDTs have been defined by ISO (International Organization for Standardization), viz. LOTOS (LOT) and ESTELLE (EST), for distributed system specification. On the other side of the coin, the European Community has supported the project ESPRIT-SEDOs (Software Environment for Distributed Open System) in order to define suitable automated tools for the design of distributed systems. Compilers, syntax checkers, simulators and verifiers for both ESTELLE and LOTOS have been implemented in SEDOS project.

On the contrary, only few applications of FDT's exist in the field of the synthesis of discrete event processes, and in particular for the synthesis of discrete controllers. Some examples can be found in [RAM] and [SNE].

This paper deals with the suitability of CCS language for computing the possible controllers of distributed systems. In [CAR1] the authors has sketched a methodology for the synthesis of such controllers by using an extension of Milner's CCS. In this paper the authors present an automated tool able to design discrete controllers by using the above referred methodology.

In particular, Section 2 and Section 3 briefly recall the Extended CCS (ECCS) and the synthesis methodology, respectively. Section 4 deals with the description of the synthesis tool. Section 5 shows an application of the proposed tool by means of some toys examples. Finally, some conclusive remarks are discussed.

2. THE LANGUAGE

The Extended Calculus for Communicating Systems (ECCS) introduced in [CAR2] is an FDT based on Milner's CCS tailored for describing asynchronous distributed systems.

ECCS is a calculus based on two key concepts:

- communication
- observation.

The system to be specified (called process in the following) consists of a black box communicating with its environment through...
interaction points named gates. The system specification is the ordering in time of the communication events taking place at the above gates as observed from the black box environment.

Two kinds of event can be identified:
1) a communication between the black box and its environment;
2) the absence of observable communication.

The first kind of event is referred to as "observable action". It is the atomic form of communication. The 'observable actions' of a process are its offers to communicate with the environment. An observable action is defined by:
- the gate where communication is offered;
- the direction, input or output from the viewpoint of the process.

An observable action may take one of the following forms:

- 
  \[
  \text{g}^? \quad \text{output at the gate g}
  \]
- 
  \[
  \text{g}^\sim \quad \text{input at the gate g}
  \]

The second kind of event is referred to as "internal action" denoted by \( \tau \), for it no agreement with the environment is requested.

An ECCS process expression can be built with the syntax in Table 1.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Process Expression</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inaction</td>
<td>nil</td>
<td>nil</td>
</tr>
<tr>
<td>Action guard</td>
<td>a : P</td>
<td>a is an action</td>
</tr>
<tr>
<td>Choice</td>
<td>P + Q</td>
<td></td>
</tr>
<tr>
<td>Parallel</td>
<td>P</td>
<td>Q</td>
</tr>
<tr>
<td>Composition</td>
<td>P (S)</td>
<td>S is a relabelling</td>
</tr>
<tr>
<td>Relabelling</td>
<td>P \ A</td>
<td>A is a gate list</td>
</tr>
<tr>
<td>Restriction</td>
<td>P \ A</td>
<td></td>
</tr>
<tr>
<td>Disable</td>
<td>P &lt; Q</td>
<td></td>
</tr>
<tr>
<td>Behaviour</td>
<td>p</td>
<td>p is an identifier</td>
</tr>
</tbody>
</table>

Table. 1. ECCS syntax

The kernel ECCS constructs are:
- Inaction process
- Action guard operator
- Choice operator

Any process can be described using only these operators.

Process Inaction is the simplest ECCS operator, and it is the process that does nothing. Action guard operator is used to describe a process a : P that can execute an action a and transform itself into another process P. Choice operator is used to describe a process P + Q that can choose to behave either like process P or Q.

Other relevant ECCS operators are Parallel Composition and Restriction operator. Parallel composition operator P|Q describes the interleaving of the events that can be executed by P and Q. Restriction operator P \ G is used to hide the gates of P belonging to set G. The use of Parallel Composition operator and Restriction operator together (with the form (P|Q)\G) allows the specifier to describe communication between processes P and Q through the gates belonging to set G. Parallel Composition and Restriction operators can be also called "derived operators". In fact, an expansion theorem is defined to transform (P|Q)\G into a process expression consisting only of Inaction processes, Action guards, and Choice operators.

Let us note that the ECCS capability to associate a process expression with an identifier allows us to describe recursive processes.

An informal meaning of all ECCS operators can be found in [CAR2].

The ECCS formal semantics is given in terms of inference rules: P -a-> P' means that process P can execute action a and transform itself into process P'. The inference rules of Action Guard and Choice operator are the following:

Action Guard a : P -a-> P

Choice Operator P -a-> P' then P + Q -a-> P' + Q'a-> Q' then P + Q -a-> Q'

The complete formal semantics of the ECCS operators can be found in [CAR2].

Let us note that the semantics of an ECCS process can be directly mapped onto a derivation tree. Where each node represents a process expression, and each edge represents an action (observable or not). For example, the process

\[
\text{g}^? : \text{nil} + \text{h}^\sim : \text{P}
\]

can be represented by the tree in Fig.1

```
   g^?
   /   \
  /     \n/       h^\sim
\text{nil}     \text{P}
```

Let us note that the existence of an adequate equivalence notion in the FDT plays a important role for the solution of the analysis and synthesis problems.

185
The ECCS equivalence notion, named "observation equivalence", is suitable to this purpose. A definition of this kind of equivalence can be given in terms of a bisimulation relation $R$. The process pair $<P,Q> \in R$ iff

\[ \forall P \Rightarrow P' \exists Q' : Q \Rightarrow Q' \land <P',Q'> \in R \]

\[ \forall Q \Rightarrow Q' \exists P' : P \Rightarrow P' \land <P',Q'> \in R \]

where:

$s$ is a sequence of observable actions (possibly the empty sequence), for instance $a_1 a_2 ... a_n$;

$=s= \rightarrow \tau \rightarrow * \rightarrow \tau \rightarrow * \rightarrow an \rightarrow \tau \rightarrow *$

3. SYNTHESIS METHODOLOGY

The design of the controller of a given plant can be viewed as consisting of the following main steps:

1. specification of both the plant and its desired behaviour;
2. computation of a set of the possible controllers on the basis of the specification mentioned in step 1;
3. choice of the more adequate controllers among those computed in step 2.

The specification problems dealing with the first item are outside the scope of this paper. The reader can found some specification examples in [CAR3].

The second item represents the kernel of this paper. The use of a formal approach to specify both the plant $P$ and its desired behaviour $B$ provides the tool with a powerful way to calculate the controller $C$. In fact, the controller specification can be obtained by solving the following equation:

\[ P \circ P C = B \]  \hspace{1cm} (1)

where:

$B$ is the ECCS specification of the expected behaviour;

$P$ is the ECCS specification of the plant;

$C$ is the unknown ECCS specification of the controllers;

$\circ$ has the meaning of "any valid ECCS operator".

Actually, the usual problems are correctly represented by the parallel composition operator with restriction, so equation (1) looks like

\[ (P \circ C) \upharpoonright G = B \]  \hspace{1cm} (2)

where the restriction operator cuts away (or hides from the external point of view) the gates belonging to set $G$ used by the Controller to drive the Plant.

Equation (2) can be solved by the four steps described below.

The first step is to put the process expressions specifying the plant $P$ and the desired behaviour $B$ in a minimal form. Note that the minimal form is the specification with minimum number of choices and minimal $\tau$ sequences in the class of the equivalent specifications. This step is only to improve the effectiveness of the method.

The second step consists in computing the gate set $G(B)$ and $G(P)$. That is, the sets of gates of process $B$ and $P$ respectively.

The third step deals with the analysis of the gate sets $G(B)$ and $G(P)$ and the determination of the gate set $G(C)$, (that is, the set of gates of process $C$), as follows:

1) $G = G(P) - (G(B) \cap G(P))$ 

$G(B) \subseteq G(P)$

In this case the model of the possible Controller is the following:

![Fig.2](image)

and $G(C) = G$

2) $G = G(P) - (G(B) \cap G(P))$ 

$G(B) \subseteq G(P)$

In this case the model of the possible Controller is the following:

![Fig.3](image)

and $G(C) = (G(B) \cup G(P)) - (G(P) \cap G(B))$

3) $G \not\subseteq G(P)$ 

In this case no possible Controller exists.

Before describing the fourth step we introduce the following notation:

- $g_i$ the gates belonging to set $G$ used by the Controller to drive the Plant.
By this notation we can rewrite the components of equation (2) as follows:

- Plant
  \[ P = Pr + Pp + Pg \]
  where
  \[ Pr = E_i \tau ; P_i ' \]
  \[ Pp = E_i p_i ; P_i ' ' \]
  \[ Pg = E_i g_i ; P_i ' ' ' \]

- Desired Behaviour of the plant
  \[ B = Bt + Bp + Bc \]
  where
  \[ Bt = E_i \tau ; B_i ' \]
  \[ Bp = E_i p_i ; B_i ' ' \]
  \[ Bc = E_i c_i ; B_i ' ' ' \]

- Controller
  \[ C = Cc + Cg \]
  where
  \[ Cc = E_i c_i ; C_i ' \]
  \[ Cg = E_i g_i ; C_i ' ' \]

Consequently equation (2) can be rewritten as follows:

\[
(Pr + Pp + Pg) \{ (Cc + Cg) \} \backslash G = Bt + Bp + Bc \quad (3)
\]

The fourth step consists of an algorithm [CAR1] to verify the following four conditions obtained by assuming, as hypothesis, equation (3) satisfies Milner's bismulation relation:

(a) \( P' \{ C \} \backslash G = B' \), \( P' \{ C \} \backslash G = B' ' \), \( P' \{ C' \} \backslash G = B' ' ' \), \( P' \{ C' ' \} \backslash G = B' ' ' ' \),

(b) for each derivation \( \tau \) of \( P \), it must exist a correspondent derivation \( \tau \) of \( B \) such that \( P' \{ C \} \backslash G = B' \).

(c) for each derivation \( \tau \) of \( B \), which has no correspondent derivation \( \tau \) in \( P \) according to case b, it must exist a correspondent derivation \( \tau \) due to the communication between processes \( P \) and \( C \) such that \( P' \{ C' \} \backslash G = B' ' \).

(d) \( Cc \{ C \} = Cc \{ C \} \backslash G = B' ' ' \), \( Cg \{ C \} = Cg \{ C \} \backslash G = B' ' ' ' \),

The verification algorithm has to be applied to each of the above equations; this recursive procedure successfully terminates when no new equation exist to be verified. Of course, if some equation cannot be solved, the algorithm proves that the controller does not exist.

4. TOOL

This section presents an automated tool to design controllers of distributed discrete systems by direct synthesis. As above said this tool should solve equation (3) according to the recursive algorithm presented in Section 3. However, we choose to solve this synthesis equation by an algorithm consisting of a three steps approach as follows: firstly we identify, if any, a controller candidate to satisfy equation (3), then we compute the ECCS expression of this controller, finally we check if it really satisfies equation (3).

The tool is implemented in PROLOG programming language. In fact, it is easy to map ECCS expressions onto PROLOG predicates. Moreover, the PROLOG backtracking facility makes easy the implementation of the synthesis tool.

To solve the first two steps of the above approach we propose in the following a modified version of another existing tool presented by the authors in [CAR4], called BIP (Bisimulation Prover). The third step will be performed only by BIP. We recall that BIP is a tool able to automatically verify the observation equivalence between two given ECCS processes by using the bisimulation relation proposed by Milner.

The existence of the above said controller needs that the tool has to successfully verify all the conditions (a), (b), (c) and (d) of Section 3. The test of existence of the controller is given by the procedure represented by the flow diagram shown in Fig.4, whose main points are the following:

1) to verify if the set \( G(P) \subseteq G(\)B)\).

2) to verify if the plant does not need any controller, in other words, if the plant has to be controlled by the process nil (i.e. null controller). To this aim, we note that if our synthesis tool is involved only in checking conditions (a) and (b), this means that all conditions (b) are satisfied; otherwise, it should be necessary to verify condition (c).

Then
the solution of equation (2) depends only on if all conditions (a) are satisfied. We can have two possibilities: i) the plant needs the null controller, ii) it cannot be controlled in any way to give the desired behaviour. In this case the synthesis tool simply behaves as a verifier tool, like BIP, able to prove the equivalence between processes \((P \cup \text{nil}) \setminus G\) and \(B\).

To solve the above point 3), in the case shown in Fig.2, let us consider a modified version of BIP, called SYNT1 in the following, that behaves like BIP but handles the events of \(P\) belonging to \(G\) as internal events when this is requested to satisfy condition (c). If condition (a), or (c), is not verified, SYNT1 terminates by correctly concluding that the controller does not exist; otherwise it can conclude the existence of a candidate controller \(C\) identified by the procedure EXP explained below.

To solve the synthesis problem, in the case shown in Fig.3, we consider a modified version of SYNT1, called SYNT2 in the following, that behaves like SYNT1 but handles the events \(c\) belonging to \(G(C)\setminus G\) as events executable by \(P\) as follows:

\[
P - c \to P
\]

that is as events non modifying the state of \(P\). If condition (a), or (c), is not verified, SYNT2 behaves as SYNT1; otherwise it can conclude the existence of a candidate controller \(C\) identified by the procedure EXP explained below.

Now we explain the procedure EXP used by SYNT1 and SYNT2 to identify the expression of a possible controller. Firstly we assume that SYNT1 and SYNT2 have successfully terminated, that is that they have always successfully checked the conditions (a) - (d). Then we note that during this proof SYNT1 and SYNT2 produce a tree \(T\) representing the ordering in time of the events \(g,c\) and \(p\) executed by the plant or by the candidate controller to satisfy conditions (a) - (d). Each node \(t_i\) of \(T\) transforms itself into another node as follows:

\[
\begin{align*}
\text{a) } & t_i \to p \to t_k \\
\text{b) } & t_i \to g \to t_r \\
\text{c) } & t_i \to c \to t_s
\end{align*}
\]

where case a) is due to an event \(p\) executed by the plant, whereas cases b) and c) are due to events executed by the candidate controller respectively in cooperation with the plant or alone.

By definition of parallel composition, from the controller point of view the above node \(t_k\) is not distinguishable from the node \(t_i\); therefore the candidate controller tree can be obtained by modifying \(T\) according to the following procedure:

i) the indistinguishable nodes \(t_k\) and \(t_i\) have to be considered as a single node \(t_{ki}\) such that:

\[
\forall \epsilon \in P(t_i) \\
\text{if } t_i \to \epsilon \rightarrow t_k \text{ then } t_{ki} \to \epsilon \rightarrow t_k
\]
\[ \psi e2 \in F(tk) \]
if \( tk - e2 \rightarrow ts \) then \( tki - e2 \rightarrow ts \)

ii) this node \( tki \) does not exist if:

\[ F(ti) \cap (G(c) - G) \neq F(tk) \cap (G(c) - G) \]

that is if the set of observable events of \( T \) executable at node \( ti \) is not the same of that executable at the indistinguishable node \( tk \).

Note that the nodes \( ti \) and \( t2 \) obtained from two indistinguishable nodes by executing the same event of type \( c \) has to be considered indistinguishable too, that is:

if \( ti, tk \) indistinguishable
and \( ti - c \rightarrow t1, tk - c \rightarrow t2 \)
then \( t1, t2 \) indistinguishable.

These nodes \( ti \) and \( t2 \) have to be considered as a single node as explained before.

Procedure EXP terminates when there are no indistinguishable node pairs to be considered as a single node. Of course EXP unsuccessfully terminates if condition ii) is satisfied. The tree obtained by EXP represents the candidate controller \( C' \) if the input events are changed in output events and viceversa.

The third step is managed by using BIP. Generally we can have two possibilities:

- equation (2) is satisfied when we use the controller expression \( C' \) identified by SYNT1 or SYNT2;

- equation (2) is not satisfied; in this case by using BIP it is possible to know the event that determines the unequivalence between \( (P \mid C') \setminus G \) and \( B \), and consequently we can try to check if the desired result can be obtained by modifying the plant expression. Note that this plant modification could not be in agreement with that expected by \( B \) in some other point. This will be signalled by BIP at the due point. If so the controller does not exist, otherwise the controller exist on condition that the plant be modified.

5. EXAMPLES

This section shows some small examples to illustrate the concepts presented in Section 4.

Let us consider the trees representing the behaviour of the plant \( P \) and of the desired behaviour \( B \):

By applying the procedure EXP described at the end of Section 4 we obtain the following candidate controller tree \( C' \):

![Diagram]

By applying BIP we obtain that:

\[ (P \mid C') \setminus G = B \]

Then \( C' \) is the controller that solves the synthesis problem.

Now we assume \( B \) as the desired behaviour, whereas the plant \( P' \) has the following expression:

![Diagram]

In this case the tree \( T \) produced by SYNT2 is the same of the first case and therefore \( C' \) has also the same expression of the first case. However by BIP we obtain that:

\[ (P' \mid C') \setminus G \neq B \]

By BIP we know also that the unequivalence depends on the event \( g2 \) of \( P' \) arising after \( pl \); consequently to solve the synthesis problem it is necessary to modify the plant by pruning away the
subtree starting with the above event q2. In this way P' is transformed onto P and by BIP we obtain: (P | C')AG = B.

Another more realistic example is a ship lock (as plant) with two doors in which a ship can pass from door1 to door2. It is a black box interacting with the environment through the following gates:

- o1 open door 1
- o2 open door 2
- c1 close door 1
- c2 close door 2
- p1 a ship passes through door 1
- p2 a ship passes through door 2

The plant tree is as follows:

The desired behaviour is the following:

By BIP we deduce that this candidate controller is the desired controller of the ship lock.

6. CONCLUSIONS

This paper presents a prototype tool for the synthesis of discrete controllers for distributed systems. The use of PROLOG programming language made easy the implementation of the tool. ECCS specification language allows us to study discrete and asynchronous systems characterized by non-deterministic behaviours. These systems are widely diffused in computer networks, robotics and process control.

REFERENCES


[CAR2] V. Carchiolo et alii, "ECCS and LIPS: two languages for OSI systems specification and verification", Submitted to ACM Transaction on Programming Language


