HYBRIDGE

Distributed Control and Stochastic Analysis of Hybrid Systems
Supporting Safety Critical Real-Time Systems Design

WP4: Compositional Specification of Stochastic Hybrid Systems

Semantics, bisimulation and interaction-structures for the CPDP model

S.N. Strubbe and A.J. van der Schaft[1]

December 13, 2004

Version: *1.0*
Task number: *4.3*
Deliverable number: *D4.3*
Contract: *IST-2001-32460 of European Commission*

[1] University of Twente

DOCUMENT CONTROL SHEET

Title of document: *Semantics, bisimulation and interactionstructures for the CPDP model*

Authors of document: *S.N. Strubbe and A.J. van der Schaft*

Deliverable number: *D4.3*

Contract: *IST-2001-32460 of European Comission*

Project: *Distributed Control and Stochastic Analysis of Hybrid Systems Supporting Safety Critical Real-Time Systems Design (HYBRIDGE)*

DOCUMENT CHANGE LOG

Version #	Issue Date	Sections affected	Relevant information
0.1	13-5-2004		First draft
0.2	3-11-2004	Ch. 6, Bibliography	After comments by Henk Blom
…			
1.0	3-11-2004	Chapters 1,2,3,4	After comments from Aquila

Version 1.0		Organisation	Signature/Date
Authors	S.N. Strubbe	Twente university	3-11-2004
	A.J. van der Schaft	Twente university	3-11-2004
Internal reviewers	Henk Blom	NLR	
	Marika di Benedetto, Allessandro d'Innocenzo and Giordano Pola	University of A'quila	
External reviewers			

Semantics, bisimulation and interaction-structures for the CPDP-model[1]

S.N. Strubbe and A.J. van der Schaft

November 2004

[1]Public deliverable D16 for the EU project HYBRIDGE (IST-2001-32460)

Contents

Chapter 1

Introduction

The main aim of WP4 is the development of a framework for compositional specification and analysis of PDP systems. The PDP model (Piecewise Deterministic Markov Process) forms a class of stochastic hybrid processes that contains virtually every process that does not have diffusion terms (see [1] and [2]). In the report [13], we developed the automaton framework CPDP. A CPDP is an automaton that represents a component of a complex PDP. A complex PDP consists of multiple components, which can all be modelled by CPDPs. In [13] we also defined a composition operator ||, which expresses and formalizes the interaction between the component CPDPs.

In this report we take a close look at the CPDP-syntax, the CPDP-semantics and their interrelation. (CPDPs are defined in [13]). The CPDP-model is designed mainly for compositional specification and analysis of PDP systems. A syntax for CPDPs should therefore be suitable for specification, but also for analysis. Thus, for the analysis part, the syntax should be such, that it can be exploited for analysis. Till now, in the definition of the CPDP (the syntactic object) most emphasis was put on the convenience of modelling. First we defined the CPDP model. Then we realized that the communication mechanism was somewhat restricted and we defined an extended version of the CPDP-model. In this report we will define an even more powerful composition operator, but in order to incorporate that operator in the CPDP-model, we might be forced to change the syntax again (although it need not be drastically). Finally, there is also the intriguing idea of compositional analysis. So far we did not pay attention to this in a structural way, but if we do this in the future, we should be open for the fact that this might need another change in the CPDP-syntax. Thus, there are enough reasons not to be dogmatic about the CPDP-syntax at this point of research.

In Chapter 2, we approach the question which semantics is suitable for CPDPs. Different answers could be given, but from a PDP point of view, the answer seems to be straightforward. However, before we answer this question we first look at how the CPDP-model originated from the PDP-model. One of the main points is that CPDPs allow multiple jumping-mechanisms at the same time, i.e. they allow that there is a race of multiple stochastic variables that can generate a jump (the one who jumps first, wins the race). For example, at a specific location two events may happen, event 1 and event 2, and both events have exponentially distributed variables (as in a Markov chains). If event 1 happens, a jump to location L happens and if event 2 happens, a jump to location L' happens. In the PDP-model, this can be modelled,

but there, the two stochastic variables should be combined into one stochastic variable. In the CPDP-model we chose to explicitly allow this in the syntax: In the CPDP-model we can define a (spontaneous) transition to L for event 1 and we can define another (spontaneous) transition to L' for event 2. Both transitions correspond with a separate stochastic variable and the semantics of CPDP is defined such that there is a race between these two stochastic variables. Thus, for convenience of modelling we think it is good to allow explicit modelling of different stochastic events at the same time (in the sense of a race). Of course, the question remains whether such a syntax can easily be exploited for (compositional) analysis. Although we use (intentionally) a different syntax than the PDP-syntax, in Chapter 2 we prove that the corresponding stochastic process of a CPDP is indeed a process that also corresponds to some PDP.

Bisimulation for PDPs and CPDPs is treated in Chapter 3. In our current research, we did not yet define a fixed or final version of bisimulation for (C)PDP. Therefore, the material of this chapter should be seen as some first steps towards a theory of bisimulation for (C)PDPs. Also, here we make our first steps when it concerns compositional analysis of (C)PDPs. Bisimulation can be seen as a technique for analysis in the context of a composite system: with bisimulation-techniques, we can reduce the state-space of certain components and therewith the state-space of the composite system. It will turn out that, because of the stochastic aspects of (C)PDPs, much attention must be paid to measurability aspects of equivalence relations. We give a definition of bisimulation for PDPs and we make some comments about how this definition could be adapted for the CPDP case.

In Chapter 4 we define the composition operator $|_A^P|$, which is more powerful than the CPDP composition operator $||$ (see [13]). Although the operator is embedded in the active/passive framework (a distinction which was made for communication between CPDPs), it is defined in a very general way such that it can potentially be applied to CPDPs, but also to any other transition-based model for communicating processes. In this chapter we also give a potential application field for this operator, supervisory control (see [11]), and we explain why we think that this operator can be used in a natural way for specification of (modular) supervisory control systems. This chapter can be seen as a stand-alone chapter. Incorporating the defined operator in the CPDP-model is something we plan to work out in the near future. This will then enrich the interaction-structures (where the title of this report refers to) for CPDPs.

Within the Hybridge project, another model is used for compositional specification of PDPs, which is the Petri-net model DCPN (Dynamically Colored Petri-Nets). In Chapter 5 we compare the DCPN model with the CPDP model. For the comparison we distinguish two aspects. First, we compare the (compositional) modelling power of the two models. Second, we compare the syntactic structures from a compositionality point of view and we ask whether these structures might be suitable for exploitation by compositional analysis techniques.

In the final chapter we give an overview of the research done in WP4 of the Hybridge project and we point out directions for future research within WP4 of the Hybridge project.

Chapter 2

CPDP Semantics

We will describe how the CPDP model originated from the PDP model. This will give insight in the relation between PDPs and CPDPs. This insight might make clear which semantics, or which solution concept, is suitable for CPDPs.

First we look at the most evident similarities between PDPs and CPDPs. The state-space of PDPs and CPDPs are hybrid and have identical structures: There are countably many locations and each location has its own continuous state space which is an open subset of the euclidean space $\mathbb{R}^n$, where n may depend on the location. Furthermore, for both the PDP and the CPDP model, the piecewise continuous dynamics is specified by means of a vector field: To each hybrid state a direction vector is assigned which determines the continuous flow. So far, the PDP model and the CPDP model are identical. The difference lies in the jump-process and in the stochastic reset-maps.

We could say that the jump-process and the reset-maps of the PDP model are split out over the different locations in the CPDP model. Suppose we have a (C)PDP with two locations, l_1 and l_2, and the initial state is some state in location l_1. For the PDP there is one stochastic variable (with some survivor function) that determines when the jump from location l_1 takes place. When this jump takes place, there is one probability measure that determines the target state of the jump. This target state might be in location l_1 as well as in location l_2. For the CPDP, both potential target locations l_1 and l_2 have their own stochastic variables that determine potential jump-times. The jump-time is therefore determined by a race of these two stochastic variables: We draw samples for both stochastic variables and the one with the smallest value wins and determines the target location. Then the winning stochastic variable has its own reset-map (probability measure) that determines the continuous state within the target location. When none of the two stochastic variables decides to jump before the boundary of the invariant is hit, we get a boundary-hit jump. This means that an additional stochastic variable (the choice function) chooses which location, l_1 or l_2, will be the target location of the jump and then this location has its own stochastic reset map that determines the continuous state after the jump.

This specific CPDP process is specified in the automaton model as follows. Between two locations (lets say from l_1 to l_2) there can be a spontaneous transition. Lets call this transition s. This transition s assigns a jump-rate $\lambda_s(\xi)$ and a probability measure $R_s(\xi)$ on the

invariant (i.e. continuous state-space) of l_2 to each continuous state ξ within l_1. The jump-rate λ_s determines a specific survivor function (or stochastic variable) for this transition. If in a specific run of the system, this transition wins the race and generates a jump from state ξ, then $R_s(\xi)$ is the probability measure that determines the target continuous state in l_2.

Between two CPDP locations (say l_1 and l_2) there can also be a boundary-hit transition (say b). This transition b assigns to each state ξ in l_1 a probability measure $R_b(\xi)$ on the invariant of l_2. If a run of the CPDP process hits the boundary of l_1 at state ξ, then the choice function may choose transition b, and then $R_b(\xi)$ determines the target continuous state in l_2.

Because we have split out the PDP-jump mechanism over different transitions in CPDP, we have created a clear distinction (in the sense of different transitions) between jumping to, say, location 1 and jumping to, say, location 2. In the PDP both jumps are combined in one jump-rate and one reset-map. This distinction in CPDP can be exploited in communication between components. A component can now communicate to the other components to which location it jumps.

We will prove that for every CPDP, all spontaneous and boundary-hit transitions with their corresponding stochastic reset maps can be combined into one jump-rate function λ and one reset map Q and then the PDP with characteristics λ and Q will have exactly the same stochastic executions (sample paths) as the CPDP. This means that the CPDP and this corresponding PDP have the same stochastic process. The stochastic process of a CPDP can therefore be realized as the realization of the stochastic process of the corresponding PDP. Proving that every PDP can be written as a CPDP can be done in the same lines. The proof is based on the *transition equivalence* notion from [13], but is more complete and sound than the proof in [13].

We first give the exact definition of CPDP and the definition of its stochastic executions. Then we prove the equivalence with PDP. After that we discuss the semantics or solution concepts for CPDPs.

2.1 Definition CPDP

In this section we introduce the CPDP formalism. First we will formally define its structure and after that we will explain how the execution of a CPDP takes place.

A CPDP $\mathcal{A}$ is a 9-tuple $(L, d, Inv,\ A, C, B, M, P,\ G)$, where

- L is a countable set of locations.
- $d : L \to \mathbb{N}$ is a mapping, which maps each location to the dimension of the continuous state space in that location.
- Inv maps each location to its invariant set. This means that for each $l \in L$, $Inv(l)$ is an open subset of $\mathbb{R}^{d(l)}$. We also define the boundary set $\partial Inv(l) := \overline{Inv(l)} \backslash Inv(l)$ of $Inv(l)$, where $\overline{Inv(l)}$ denotes the closure of $Inv(l)$.

- A is the set of labels. These labels are the names of the active discrete events that happen when boundary-hit or spontaneous transitions are executed.

- C is the transition-choice function. $C(b, l, \xi) \in [0, 1]$ is the probability of executing the boundary-hit transition b (see next item) from the boundary state (l, ξ). $C(b, l, \xi)$ is defined on all $l \in L$ and all $\xi \in \partial Inv(l)$ and all $b \in B$ (see next item) that are outgoing transitions of l. Furthermore $\sum_{b \in B_{l\rightarrow}} C(b, l, \xi) = 1$, where $B_{l\rightarrow}$ is the set of all elements of B that are outgoing transitions of l.

- B is the set of boundary-hit transitions. Each element b of B is a quadruple (l, a, l', R), where l is the origin location, a is the label of the jump, l' is the target location, and R is the reset map of the jump. $R(A, \xi) \in [0, 1]$ is the probability of jumping into the set A when the transition b is taken from state $\xi \in \partial Inv(l)$. $R(A, \xi)$ is defined for all $\xi \in \partial Inv(l)$ with $C(b, l, \xi) > 0$ and for all Borel subsets A of $Inv(l')$.

- M is the set of spontaneous transitions (the equivalent of Markovian transitions in Interactive Markov Chains in [4]). We also refer to these spontaneous transitions as Poisson transitions. Each element m of M is a pentuple (l, a, l', R, λ), where l is the origin location, a is the label of the jump, l' is the target location, R is the reset map of the jump, and λ is the jump rate. $R(A, \xi) \in [0, 1]$ is the probability of jumping into the set A when the transition m is taken from state ξ. $R(A, \xi)$ is defined for all $\xi \in Inv(l)$ and for all Borel subsets A of $Inv(l')$. $\lambda : Inv(l) \rightarrow \mathbb{R}_+$ is a bounded Borel measurable function, and determines the rate of jumping of the process.

- P is the set of passive transitions. Each element p of P is a quadruple (l, a, l', R), where l is the origin location, a is the label of the jump, l' is the target location, and R is the reset map of the jump. $R(A, \xi) \in [0, 1]$ is the probability of jumping into the set A when the transition p is taken from state ξ. $R(A, \xi)$ is defined for all $\xi \in Inv(l)$ and for all Borel subsets A of $Inv(l')$.

- G determines the flow of the continuous states within the locations. For each $l \in L$, $G(l)$ is a locally Lipschitz continuous function from $\mathbb{R}^{d(l)}$ to $\mathbb{R}^{d(l)}$, and is the vector field for the continuous flow in location l.

In order to restrict the behavior of CPDPs to PDP-behavior, we need to impose the standard PDP conditions on a CPDP. They can be found on page 62 of [2]. Also it is not necessary that choice-functions and reset-maps of boundary-hit transitions are defined on all boundary-states. Some boundary-states are non-reachable and there these functions and reset-maps need not be defined.

A CPDP $\mathcal{A}$ as defined above, generates a stochastic process. To give insight to the execution of a CPDP process, we will now describe how sample paths of this stochastic process can be generated. As we will see later, the passive transitions play a role in communication with the outside world. For the generation of a sample path, we assume that the outside world is silent, in other words, no communication takes place and therefore, the passive transitions play no role in this sample path generation.

Generating sample-paths for a CPDP

We assume that an initial hybrid state $x_0 = (l_0, \xi_0)$ is given. The flow function $\phi_{l_0}(t, \xi_0)$ of the continuous state ξ in location l_0 is for $t \geq 0$ determined by the differential equation

$$\frac{\mathrm{d}}{\mathrm{d}t}\xi = G_{l_0}(\xi),$$

i.e.

$$\frac{\mathrm{d}\phi_{l_0}(t, \xi_0)}{\mathrm{d}t} = G_{l_0}(\phi_{l_0}(t, \xi_0)).$$

$G(l)$ is written here as G_l. Suppose that $\xi(t)$ reaches the boundary $\partial Inv(l_0)$ at time τ_b (otherwise, τ_b equals ∞) and suppose that location l_0 has n outgoing spontaneous transitions. During the continuous flow, for every outgoing spontaneous transition m we define a survivor function

$$F_m(t, \xi_0) = I_{(t<\tau_b)} \exp\left(-\int_0^t \lambda_m(\phi_{l_0}(t, \xi_0))\mathrm{d}t\right),$$

which can be interpreted as follows: The probability that either m generated a jump before time instant t or the process reached the boundary before time instant t equals $1 - F_m(t, \xi_0)$. Note that I_A is the indicator function: $I_A x = 1$ if $x \in A$ and $I_A(x) = 0$ if $x \notin A$.

For each outgoing spontaneous transition m_i, we draw a sample τ_i from its survivor function (i.e. a sample from a random variable with distribution function $1 - F_m$). This means that m_i would cause a jump at time τ_i if $\tau_i < \tau_b$ and no other Poisson-jump occurred before τ_i. Therefore the most relevant spontaneous transition is the one corresponding to

$$\tau_M := \min_{i=1..n} \tau_i,$$

assumed that $\tau_M < \tau_b$. This transition causes a jump at time τ_M. If $\tau_M = \tau_b$, then we have a boundary-hit jump at time τ_b. Note that the latter does not mean that a Poisson point is generated exactly at the boundary, but that no Poisson points are generated before the boundary is hit.

Boundary-hit transition

A boundary-hit transition at time τ_b from the boundary state $\xi(\tau_b) \in \partial Inv(l_0)$ is executed as follows. It could be that multiple boundary-hit transitions are active in state $\xi(\tau_b)$, therefore we use the choice function C to choose one of the active transitions. We draw a sample $b \in B_{l_0\rightarrow}$ from the probability measure determined by $C(\cdot, l_0, \xi(\tau_b))$. Now, $b = (l_0, a_b, l'_b, R_b)$ is the transition that will be executed. The target location is l'_b and the target state ξ' in $Inv(l'_b)$ is drawn from the probability measure $R_b(\cdot, \xi(\tau_b))$. With the new hybrid state (l'_b, ξ') at time τ_b, we can repeat the algorithm above to continue the sample path.

Spontaneous transition

A spontaneous transition $m = (l_0, a_m, l'_m, R_m, \lambda_m)$ at time τ_M from the state $\xi(\tau_M) \in Inv(l_0)$ is executed as follows. The target location is l'_m and the target state $\xi' \in Inv(l'_m)$ is drawn from the probability measure $R_m(\cdot, \xi(\tau_M))$. With the new hybrid state (l'_m, ξ') at time τ_M,

we can repeat the algorithm above to continue the sample path.

With the sample path description above, we informally described the stochastic processes that correspond to the CPDPs.

2.2 Equivalence CPDPs and PDPs

To show that closed CPDPs (i.e. CPDPs that do not have passive transitions) and PDPs generate the same class of stochastic processes, we introduce the following notions for closed CPDPs.

Suppose we have a CPDP with initial state $x = (l, \xi)$. With τ_b we denote the first potential boundary-hit time (i.e. the boundary-hit time assumed that no spontaneous jumps happen before the boundary is hit). With $\mathcal{H}$ we denote the hybrid state space, i.e. $\mathcal{H} = \{(l, \xi) | l \in L, \xi \in Inv(l)\}$, and we write $\partial\mathcal{H} = \{(l, \xi) | l \in L, \xi \in \partial Inv(l)\}$. Suppose that T_m is a random variable with distribution-function $1 - F_m$ for any spontaneous transition m. Then we define

$$T = \min_{m \in M_{l\rightarrow}} T_m.$$

$P(T < t)$ equals the probability that either one of the Poisson-transitions generated a jump before t or the boundary was hit before t. It can easily be seen that T has distribution function

$$1 - I_{(t<\tau_b)} \exp\left(- \int_0^t \sum_{m \in M_{l\rightarrow}} \lambda_m(\phi_l(t, \xi)) \mathrm{d}t \right).$$

Therefore, the total survivor function $F(t, x)$ of the CPDP process equals

$$F(t, x) = I_{(t<\tau_b)} \exp\left(- \int_0^t \sum_{m \in M_{l\rightarrow}} \lambda_m(\phi_l(t, \xi)) \mathrm{d}t \right).$$

As we described before, generating a CPDP sample path, starting from hybrid state $x = (l, \xi)$, is done by first drawing samples from all Poisson transitions that are outgoing from location l, followed by drawing a sample from the reset map that corresponds to the winning Poisson transition or, in case of a boundary point, from the choice function followed by drawing a sample from the reset map of the winning boundary-hit transition. To find an equivalent PDP, we ask the following: Can we find a jump-rate function $\lambda(x)$ on the hybrid state space of the CPDP and a reset map $Q(\cdot, x)$ (also depending on the hybrid state) such that $\lambda(x)$ generates (as it does in PDPs) exactly the survivor function F of the CPDP and such that drawing a sample $\hat{t}$ from this survivor function followed by drawing a sample from $Q(\cdot, \phi(\hat{t}, x))$ is equivalent with the sample-path generation procedure as described above?

The part concerning $\lambda(x)$ is clear. We can see from the survivor function $F(t, x)$ of the CPDP that this $\lambda(x)$ should be equal to $\sum_{m \in M_{l\rightarrow}} \lambda_m(\xi)$ (remember that $x = (l, \xi)$).

For the second part, we first have to formalize the notion of equivalent sample-path generation procedures. We do that as follows. From a starting state, both procedures determine

two things: first, the time of jumping and second, the target state where to jump to. The time of jumping is an element of $\mathbb{R}$ and the target-jump-state is an element of $\mathcal{H}$. Formally we say that both procedures generate the same jump time and target state (which are of course interrelated) when they define the same probability measure P on the space $(\mathbb{R}\times\mathcal{H}, \mathcal{B}(\mathbb{R}\times\mathcal{H}))$, where $\mathcal{B}(X)$ means the Borel sets of the topological space X, where $P(A)$, with $A \in \mathcal{B}(\mathbb{R}\times\mathcal{H})$, equals the probability that (t, x), with t the jump-time and x the target state, lies in the set A. Before we show this, we first look at the structure of the sets in $\mathcal{B}(\mathbb{R} \times \mathcal{H})$.

Any Borel subset of $\mathcal{H}$ (according to the topology defined on $\mathcal{H}$ at p.58 of [2]) can be written as a countable union of disjoint subsets of the form (l, B) (see [2] p.57). This splits a Borel subset of $\mathcal{H}$ into Borel subsets of the invariants of the different locations. If for example $\hat{B} \in \mathcal{B}(\mathcal{H})$ is split into (l_1, B_1),$\cdots$,(l_n, B_n), then $Q(\hat{B}, (l, \xi))$ is equal to $Q((l_1, B_1), (l, \xi)) + \cdots + Q((l_n, B_n), (l, \xi))$.

For arbitrary $t > 0$ and for arbitrary Borel subset $B \in \mathcal{B}(Inv(l'))$ for some location l', we consider the Borel subset $A = (-\infty, t] \times (l', B) \in \mathcal{B}(\mathbb{R} \times \mathcal{H})$. For $x = (l, \xi)$ we now determine $P_x(A)$ as induced by the sample-generation procedure of the CPDP, where x denotes the initial hybrid state of the PDP.

Suppose location l has n outgoing stochastic transitions, and suppose that k of them are directed towards location l'. With $F_1, \cdots, F_k$ we denote the survivor functions of the k transitions from l to l' and with $F_{k+1}, \cdots, F_n$ we denote the survivor functions of the transitions from l not to l'. Let T_i be a random variable with survivor function F_i. In general, with μ_X we denote the probability measure induced by the random variable X. For the sake of notational simplicity we write $F(\tau)$, $R_i(B, \tau)$, $\lambda_i(\tau)$,$Q(A, \tau)$, $C(b, \tau)$ for $F(\tau, x)$, $R_i(B, \phi_l(\tau, \xi))$, $\lambda_i(\phi_l(\tau, \xi))$, $Q(A, (l, \phi_l(\tau, \xi)))$ and $C(b, l, \phi_l(\tau, \xi))$ respectively. We define $f(\tau) := \frac{\mathrm{d}F}{\mathrm{d}t}(\tau)$, which is well-defined for $t < t_*$, where t_* denotes the boundary-hit time. If $A = (-\infty, t] \times (l', B)$ is such that $t < t_*$, then we get

$$P_x(A) = \sum_{i=1}^{k} \int_0^t \left(\prod_{j \neq i} P(T_j > \tau) \right) R_i(B, \tau) \mathrm{d}\mu_{T_i}(\tau).$$

The above integrals are well-defined because $P(T_j > \tau) = F_j(\tau)$ is bounded and measurable (and therefore integrable) and $R_i(B, \tau)$ is bounded and is measurable because $R_i(B, \tau) = R_i(B, \phi_l(\tau, \xi))$ where $R_i(B, \xi)$ is measurable for fixed B and ϕ_l is continuous and therefore measurable. Then we can write

$$P_x(A) = \sum_{i=1}^{k} \int_0^t \frac{\mathrm{d}F_i}{\mathrm{d}t}(\tau) \left(\prod_{j \neq i} F_j(\tau) \right) R_i(B, \tau) \mathrm{d}\tau = \int_0^t \left(\sum_{i=1}^{k} \lambda_i(\tau) R_i(B, \tau) \right) F(\tau) \mathrm{d}\tau,$$

where $\mathrm{d}\tau$ means integrating over the Lebesgue measure and $F(\tau)$ is the total survivor function. Considering the possibility that $t \geq t_*$ we get

$$P_x(A) = \int_{[0, t_*[} \left(\sum_{i=1}^{k} \lambda_i(\tau) R_i(B, \tau) \right) F(\tau) \mathrm{d}\tau + \psi(t_*) \sum_{b \in B_{l \to l'}} C(b, t_*) R_b(B, t_*),$$

where $\psi(t_*) = I_{(-\infty,t]}(t_*)\left(\lim_{s\uparrow t_*} F(s)\right)$. Then we can rewrite the formula as follows.

$$P_x(A) = \int_{[0,t_*[} \left(\sum_{i=1}^{k} \lambda_i(\tau) R_i(B,\tau)\right) \frac{f(\tau)}{\sum_{n\in M_{l\rightarrow}} \lambda_n(\tau)} \mathrm{d}\tau + \psi(t_*) \sum_{b\in B_{l\rightarrow l'}} C(b,t_*) R_b(B,t_*),$$

where the integrand is defined to be zero when $\sum_{n\in M_{l\rightarrow}} \lambda_n(\tau) = 0$. Then, using the fact that $\lambda_i(\tau) = 0$ for all i if $\tau \geq t_*$ and using the measure μ_T induced by T, we get

$$P_x(A) = \int_{[0,t]} \left(\frac{\sum_{i=1}^{k} \lambda_i(\tau) R_i(B,\tau)}{\sum_{n\in M_{l\rightarrow}} \lambda_n(\tau)} + I_{t_*}(\tau) \sum_{b\in B_{l\rightarrow l'}} C(b,t_*) R_b(B,t_*) \right) \mathrm{d}\mu_T(\tau),$$

which we can write as

$$= \int_{[0,t]} Q(B,\tau) \mathrm{d}\mu_T(\tau),$$

if we define

$$Q(B,\tau) := \begin{cases} \sum_{m\in M_{l\rightarrow l'}} \left(\frac{\lambda_m(\tau)}{\sum_{n\in M_{l\rightarrow}} \lambda_n(\tau)} R_m(B,\tau) \right), & \text{if } \phi(\tau,x) \in Inv(l) \\ \sum_{b\in B_{l\rightarrow l'}} C(b,\tau) R_b(B,\tau), & \text{if } \phi(\tau,x) \in \partial Inv(l) \end{cases}$$

The latter expression exactly expresses $P_x(A)$ for the procedure when samples are created by drawing a sample t from T followed by drawing a sample x' from $Q(\cdot,t)$. This means that generating jump-time and target state via T and Q induces the same probability on sets of the form $(-\infty,t]\times(l',B)$. If B' is a Borel subset of $\mathcal{H}$, then it can be split into (l_i, B'_i) for the different locations l_i and then $P_x((-\infty,t]\times B')$ equals $\sum_i P_x((-\infty,t]\times(l_i,B'_i))$. Therefore, the two methods determine the same probabilities on sets of the form $(-\infty,t]\times B$, with B a Borel subset of $\mathcal{H}$. This collection of sets forms a π-system (i.e. the collection is closed under finite intersections). Two probability measures that agree on a π-system, also agree on the σ-algebra generated by the π-system. Therefore the two procedures determine the same probability measure on $\sigma\{(-\infty,t]\times B | t\in\mathbb{R}, B\in\mathcal{B}(\mathcal{H})\}$, which is the Borel σ-algebra on $\mathbb{R}\times\mathcal{H}$.

In [2] the E-valued stochastic process $\{x_t(\omega), t>0\}$ corresponding to a PDP with state-space E is formally defined on the Hilbert cube $(\Omega,\mathcal{A},P)$, which is the product space of $(\Omega_i,\mathcal{A}_i,P_i)$ $(i\in\mathbb{N})$, where $\Omega_i=[0,1]$, $\mathcal{A}_i$ is the set of Lebesgue measurable sets and P_i is the Lebesgue measure. A measurable function $\psi:\Omega\rightarrow D_E[0,\infty[$, with $D_E[0,\infty[$ the set of right-continuous E-valued functions on $\mathbb{R}_+$ with left-hand limits, is determined. This ψ assigns to each $\omega\in\Omega$ a sample function of the PDP process. Then the stochastic process of the PDP is defined as $x_t(\omega)=\psi(\omega)(t)$. Realizing the stochastic process of the CPDP on the Hilbert cube can now be done as follows: A U[0,1] stochastic variable Y_1 (defined as $Y_1(\omega)=\omega_1$, where $\omega=(\omega_1,\omega_2,\cdots)$) is used for determining the first jump-time. This is done by using the survivor function F of the CPDP. $Y_2(\omega)=\omega_2$ is used for determining the target state of the first jump. This is done by using the Q of the CPDP (as defined above). $Y_3(\omega)=\omega_3$ is used for determining the second jump-time, etc. In this way, the realization of the CPDP-process is exactly the same as the realization of the PDP that has the CPDP characteristics λ and Q. Therefore we can say that the realization of the CPDP-process is defined as the realization of the PDP that has the CPDP characteristics λ and Q. We proved

above that this realization indeed agrees with the CPDP sample-path generation procedure. This means that a CPDP-process realization is indistinguishable from the realization of its corresponding PDP (i.e. the PDP with characteristics λ and Q).

2.3 CPDP semantics

The CPDP model is a hybrid model. On the one hand there is a continuous dynamics, on the other hand there are discrete events. In CPDP, some discrete events are labelled. Jumps that correspond to active (i.e. boundary-hit) and passive transitions are labelled with elements from the set of events A. Spontaneous jumps however, are not labelled with events, because they are not used for communication.

Choosing a semantics for CPDP, means also choosing which aspects of the process are relevant for example analysis. From a PDP-point-of-view, the relevant aspects are the continuous dynamics and the stochastics, with other words the stochastic process of the CPDP (because the stochastic process is the object that we want to analyze). Therefore, we define a stochastic-processes-semantics for CPDPs, which is the stochastic process of its corresponding PDP. Note that passive transitions play no role in this semantics because passive transitions cannot influence the stochastic process, unless the CPDP is within a composition context, where the passive transitions can be triggered by other components. But for this stochastic-processes-semantics, we assume that we are not within a composition, but that we are dealing with the total composed system, i.e. with the final object that we want to analyze.

This means that if we want to express within the semantics how the process behaves as a component (i.e. within a composition context), then information regarding the labels and the passive transitions are essential, since they define the interaction behavior between components. We plan to define such a semantics, which combines the stochastic and communicating behaviors, in the near future. This semantics can then be used for compositional analysis techniques like bisimulation.

Chapter 3

Bisimulation for PDPs and CPDPs

In the literature concerning labelled transition systems (i.e. process algebra, automata etc.), bisimulation is one of the most popular equivalence notions. Bisimulation is effectively used for state-space reduction (see e.g. [4] for examples in the context of Interactive Markov Chains), which makes analysis easier. We could say that two bisimular processes are externally equivalent. They may differ when it comes to internal dynamics, but their external (that what we can see or measure and what we can influence) behaviors are the same. In [14], a bisimulation notion is defined for systems with continuous dynamics and in special for non-stochastic hybrid systems. Also algorithms are given through which we can compute maximal bisimulations (which can be used for maximal state reduction). From an analysis point of view, bisimulation can be a powerful technique, because bisimulation algorithms may reduce the state space and therefore can make analysis (which can then be performed on a smaller model) easier.

In this chapter we make the first steps towards a notion of bisimulation for PDPs and for CPDPs. One of our aims is to develop the bisimulation notion along the lines of [14], such that also algorithms can be constructed to compute maximal bisimulations. This is far from trivial, which is mainly due to the role of the stochastic aspects in PDPs and CPDPs.

From a compositional analysis point of view, a bisimulation relation should be congruent with respect to the parallel composition operator, which means that if we replace one of the components of a composite system by a bisimular component, then the resulting composite system should be bisimular with the composite system before substitution.

Concerning PDPs in this chapter, we consider PDPs enhanced with an output function. This output function maps each hybrid state to a real value, the output of the state. The external behavior is then defined as the behavior of the output process. If two PDPs have different state spaces with different dynamics, but the dynamics of the output are the same, then their external behaviors are the same. This will be the leading idea for our notion of bisimulation for PDPs. In the next section this is found back in formal language in Conjecture 1.

In the next section we give a formal definition of bisimulation for PDPs. Although the idea is rather simple, the formalization gets rather technical because of the stochastic reset-maps.

If two states (of two PDPs for example) are bisimular and a jump might take place from this state, then the reset-maps of these two states should be equivalent in some sense. This means that these reset-maps, which are probability measures, should assign the same probabilities to equivalent measurable sets. In the next section we define in Definition 1 which measurable sets are equivalent and which are not. This definition is rather technical because we have to make sure that equivalence classes of hybrid states are totally contained in a measurable set or are totally not contained. Once this technicality has been worked out, the definition of bisimular PDPs can be stated straightforwardly without any non-intuitive parts.

In the final section of this chapter we make some comments about bisimulation for CPDPs, although we do not give a formal definition because research has not been developed so far.

3.1 Bisimulation for PDPs

In this section we define an equivalence or bisimulation notion for weighted PDPs (i.e. PDPs enhanced with an output function). This notion is such that when two PDPs are bisimilar (i.e. their initial states are bisimilar), then the stochastic processes of their outputs are equivalent in the sense that we can find realizations of these processes on the Hilbert cube such that the stochastic processes are indistinguishable. That we define the realizations of stochastic processes on the Hilbert cube is mainly because we follow the lines of [2], where the stochastic process x_t of a PDP is defined on the Hilbert cube. Before we can define bisimulation for PDPs, we need to introduce some notation.

We call the measurable space $(E, \mathcal{E})$ a *standard Borel space* if E is a Borel subset of a complete separable metric space and $\mathcal{E}$ is the corresponding sub-σ-algebra. A standard Borel space $(E, \mathcal{E})$ is called *separable* if $\{x\} \in \mathcal{E}$ for all $x \in E$. The space $(E, \mathcal{E})$ of a PDP as defined in [2] is a separable standard Borel space.

Let X and Y be some spaces. We define the equivalence relation on X that is induced by the relation $\mathcal{R} \subset X \times Y$, as the transitive closure of $\{(x, x') | \exists y \text{ s.t. } (x, y) \in \mathcal{R} \text{ and } (x', y) \in \mathcal{R}\}$. We write $X/_{\mathcal{R}}$ and $Y/_{\mathcal{R}}$ for the sets of equivalence classes of X and Y induced by $\mathcal{R}$. We denote the equivalence class of $x \in X$ by $[x]$. We will now define the notion of *measurable relations* and of *equivalent measures*. We will see that in that definition, two probability measures on two spaces X and Y are equivalent with respect to some relation $\mathcal{R} \subset X \times Y$ if both measures induce the same probability measure on the set of equivalence classes of X and Y.

We will now define the notion of *measurable relation*. A relation partitions the hybrid state-space of a PDP. Suppose that two states in the same equivalence class are indistinguishable in some sense (which sense does not matter now and will become clear later). For a reset-map we are now only interested in the probability to jump to measurable sets of equivalence classes. Therefore the relevant measurable sets are those in which every equivalence class is either totally contained or totally not contained. For the definition of bisimulation we need that for such a relevant set, the corresponding set (induced by this relation) in the bisimular PDP is also measurable.

Definition 1 *Let $(X, \mathcal{X})$ and $(Y, \mathcal{Y})$ be standard Borel spaces and let $\mathcal{R} \subset X \times Y$ be a relation. Let*

$$\mathcal{X}^* = \mathcal{X} \cap \{A \subset X | \text{ if } x \in A \text{ and } [x] = [x'] \text{ then } x' \in A\}$$

be the collection of all Borel sets of X in which any equivalence class of X is either totally contained or totally not contained. It can be checked that $\mathcal{X}^*$ is a σ-algebra. Let $X/_{\mathcal{R}}$ be the set of equivalence classes of X, let $f_X : X \to X/_{\mathcal{R}}$ be the mapping that maps each $x \in X$ to its equivalence class and let

$$\mathcal{X}/_{\mathcal{R}} = \{A \subset X/_{\mathcal{R}} | f_X^{-1}(A) \in \mathcal{X}^*\}.$$

Then $(X/_{\mathcal{R}}, \mathcal{X}/_{\mathcal{R}})$, which is a measurable space, is called the quotient space *of X with respect to $\mathcal{R}$. We define a bijective mapping $f : X/_{\mathcal{R}} \to Y/_{\mathcal{R}}$ as: $f([x]) = [y]$ if $(x, y) \in \mathcal{R}$ for some $x \in [x]$ and some $y \in [y]$ (this is uniquely defined). We say that the relation $\mathcal{R}$ is* measurable *if X and Y define the same quotient space, i.e. if $f(\mathcal{X}/_{\mathcal{R}}) = \mathcal{Y}/_{\mathcal{R}}$.*

Definition 2 *Suppose we have measures P_X and P_Y on standard Borel spaces $(X, \mathcal{X})$ and $(Y, \mathcal{Y})$ respectively. Suppose that we have a measurable relation $\mathcal{R} \subset X \times Y$. The measures P_X and P_Y are called equivalent with respect to $\mathcal{R}$ if they define the same probability measure on the quotient space $(Z, \mathcal{Z})$ of X and Y, i.e. if we have $P_X(f_X^{-1}(A)) = P_Y(f_Y^{-1}(A))$ for all $A \in \mathcal{Z}$.*

Suppose we have a PDP with state-space X and f_X is a real-valued measurable function on X. Then we call the pair (X, f_X) a *weighted PDP*. (The function f_X can easily be generalized to a mapping to $\mathbb{R}^n$ or to any PDP hybrid state space). Thus, with (X, f_X) we refer to the PDP with state-space X. As long as we do not define different PDPs on one state-space, (X, f_X) will unambiguously denote one PDP with function f_X defined on its state-space. For $x \in X$ we also write $\lambda(x)$, $\phi(t, x)$, $Q(x)$, $t_*(x)$ to denote the λ, ϕ, Q and t_* functions of the PDP that corresponds with the state-space X (again assuming that there is exactly one). Thus, if $x = (\nu, \zeta)$ then $\phi(t, x)$ denotes $\phi_\nu(t, \zeta)$ for the PDP which has x as state, etc.

The function f_X can be seen as a weight function on the state-space. It can also be seen as the continuous output or the observable component of the system. We call f_X the weight-function or the output-function. We will now define an equivalence notion for weighted PDPs.

Definition 3 *Suppose we have two weighted PDPs with state-spaces X and Y and weight-functions f_X and f_Y. A measurable relation $\mathcal{R} \subset X \times Y$ is a* bisimulation *iff $(x, y) \in \mathcal{R}$ implies that*

- *$f_X(x) = f_Y(y)$, $t_*(x) = t_*(y)$ and $\lambda(x) = \lambda(y)$.*
- *$(\phi(t, x), \phi(t, y)) \in \mathcal{R}$ for all $t \in [0, t_*(x)[$.*
- *$Q(x)$ and $Q(y)$ are equivalent probability measures with respect to $\mathcal{R}$. Also $Q(\phi(t_*(x), x))$ and $Q(\phi(t_*(y), y))$ are equivalent probability measures with respect to $\mathcal{R}$.*

Two states x and y are bisimilar if they are contained in some bisimulation.

In words, we could say that Definition 3 means that two bisimilar states x and y are bisimilar if: Bullet 1: there outputs are equal, the times until the boundary is reached are equal, the jump-rates are equal (the latter two mean that they have equal stochastic jump times). Bullet 2: At all times before a jump, after the processes started in x and y, the states are bisimilar. Bullet 3: The reset maps of two bisimilar states (and of the two boundary states where they flow to) are equivalent.

Conjecture 1 *If initial states x and y of two weighted PDPs (X, f_X) and (Y, f_Y) are contained in some bisimulation $\mathcal{R}$, then, assumed that the quotient space is a separable standard Borel space, we can construct the stochastic processes x_t and y_t on the Hilbert cube $(\Omega, \mathcal{A}, P)$ in such a way that for each $\omega \in \Omega$ we have $f_X(x_t(\omega)) = f_Y(y_t(\omega))$.*

The proof of this conjecture is under construction. Note that this conjecture says that two weighted PDPs that are bisimular have indistinguishable output-processes. This means that if we are interested only in the output behavior of a PDP (which is often the case when we want to calculate distributions or expectations of PDPs), then if we can find a bisimular PDP that has a simpler structure (like a smaller state-space), then we can as well analyze that bisimular PDP.

3.2 Bisimulation for CPDPs

A bisimulation for CPDPs should concern two things: First, it should encompass the bisimulation notion for PDPs, that is, the stochastic output behavior of two bisimular CPDPs should be equivalent (Note that this needs the introduction of an output function for CPDPs). Second, two bisimular CPDPs should have equivalent behavior when it concerns communication. This means that two bisimular CPDPs interact in the same way with their environment and consequently, substituting a component for another bisimular component will not change the external behavior of the composite system.

A definition of bisimulation can be given in terms of the process semantics or in terms of the process syntax. In the definition of bisimulation for PDPs in this chapter we find a mix of syntactical terms (like the jump-rate function) and semantical terms (like the flow function). A bisimulation that is defined in terms of the syntax is also called a structural bisimulation. In order to design algorithms that can find automatically (maximal) bisimulations for CPDPs, it is probably needed that the bisimulation for CPDPs is defined in a structural way.

The question whether the CPDP is syntactically defined in a convenient way, can also be regarded from a bisimulation point of view. Therefore we will also look at the idea of a total split of the PDP-mechanism and the communication mechanism (see also Chapter 5 for this split-up idea) with the bisimulation idea in mind. With such a split-up (which is also used in Interactive Markov Chains for example), it might be possible to also split up the bisimulation in two parts: The stochastic process part and the communication part.

Chapter 4

The composition operator $|_A^P|$

4.1 Introduction

A complex system typically consists of multiple components which are running simultaneously and which are interacting with each other. Modelling these complex systems in a compositional way can be done by using a composition operator which combines the different components of the system. If we denote the composition operator by $|$, then $A|B$ is the system that is composed out of subsystems A and B. The interaction between the two subsystems is regulated by the composition rules of the operator $|$. Because the systems A and B run in parallel (or concurrently), we call $|$ the parallel composition operator.

The goal of this chapter is to make clear that for the modelling of many types of systems, it is advantageous and natural to use two types of interaction: blocking-interaction and non-blocking-interaction. We show that this idea can be formalized naturally by distinguishing active and passive actions. For this chapter it does not matter which framework is used for the modelling of systems, as long as it is a transition-based framework (such as CPDPs, automata or process algebra expressions).

In Section 4.2, which is the main part of this chapter, we introduce the active/passive framework and we develop a composition operator that can establish several types of interaction between systems (that are built out of active and passive transitions). While we develop the operator step by step, we motivate each step by means of simple and clear examples. Some of these examples are about supervisory control systems because we think that supervisory control systems provide natural examples for our modelling framework. After the framework has been introduced (including a structural operational semantics for the composition operator), we give a more extensive example in Section 4.2.5 which shows all features of the framework and the composition operator. In Section 4.2.6 we take a closer look at supervisory control system. We explain why we think that our framework has certain advantages over other frameworks in modelling supervisory control systems. Section 4.2 ends with a technical result on the operator.

In [12] we used solid arrows for active transitions and we used dashed arrows for passive transitions. If both an active and a passive transition have the same label, then they can synchronize. In this chapter we have chosen for a different notation for the passive transitions.

Here, passive transitions are distinguished from active transitions because the label has a little bar above it. Now an active transition with label a can synchronize with a passive transition with label $\bar{a}$. We chose for this notation because it is a common notation in process algebra literature (as in [9]) and the contents of this chapter is, besides for example CPDPs, in fact applicable for all kinds of process algebra models.

4.2 The active/passive framework

We can distinguish two types of interactions between processes. First, *blocking*-interaction: both partners (for example the controller and the process) need to be able to do the action, otherwise the action will not take place. This is the type of interaction that we see in many process algebra models (e.g. [9]). Secondly, *non-blocking*-interaction: one of the partners (the passive one) is not able to block the other (the active one). For example if a person (the passive partner) observes that a light (the active partner) is switched on. The person observes, but is not able to block the switching, i.e. the light could also be switched on without the person observing it.

Non-blocking-interaction can already be found in the literature, e.g. in broadcast systems ([10]) where several listening processes can receive (but not block) a signal, or in I/O automata ([7, 8]), where processes should be input-enabled, i.e. ready to receive an input in any state of the process, such that an output will never be blocked.

We see that most formalisms support only blocking interaction and that some formalisms (broadcasting systems, I/O automata) support only non-blocking interaction. However, there exists no formalism that supports both blocking and non-blocking interaction. We will motivate that for our purposes, it is desirable to have a formalism that supports both types of interaction.

The context of processes that we want to specify is the following:

1. We want to specify *supervisory control systems*. We will see that we need both blocking and non-blocking interaction for this.
2. We are also interested in *modular supervisory control*, where a controller may consist of several modules. We will motivate that in addition to blocking and non-blocking interaction, we need to control the scope of non-blocking interaction.
3. We want to specify *complex processes that consist of interacting subprocesses*. Here we will argue that we need the possibility to have multi-way synchronizations.

Now we will introduce our framework, which is based on active and passive actions and which supports both blocking and non-blocking interaction. We introduce the framework in three steps, corresponding to the three points in the above context. In each step we give motivating examples. In the first step we explain how active and passive actions can be used to establish blocking and non-blocking interaction. In the second step we explain how to deal with observations in systems that consist of more than two components. In the third step we treat the problem of multi-way synchronization, i.e. the problem whether an active action can be synchronized with multiple passive actions or only with one. After that, we give a small summary of how the composition operator works.

4.2.1 Step 1: establishing blocking and non-blocking interaction

We consider two types of actions: *active* actions (denoted as a, b etc.) and *passive* actions (which are observing actions and are denoted as $\bar{a}$, $\bar{b}$, etc.). Because we want to have both blocking and non-blocking actions, we have to make clear which actions are blocking and which actions are non-blocking. Therefore, we introduce the set A which contains all actions that are blocking. The composition operator will now be denoted by $|A|$. We still use $|$ to denote the composition operator in cases where A is unspecified or where the set A is not used. Blocking-interaction is now expressed by the following operational rule:

$$r1.\frac{L_1 \xrightarrow{a} L_1', L_2 \xrightarrow{a} L_2'}{L_1|A|L_2 \xrightarrow{a} L_1'|A|L_2'}(a \in A),$$

which says that a blocking-synchronization (or active-active synchronization) on a from joint location $L_1|A|L_2$ to $L_1'|A|L_2'$, can only happen when both partners have the action a available from locations L_1 and L_2 to locations L_1' and L_2' respectively. (In order to comply with the terminology used in timed and hybrid automata, we use the term *locations* to designate the *states* in an automaton). For a graphical example of a product automaton that is the result of composition operators that have composition rules in the form of $r1$, we refer to [13]. In Figure 4.1, where $a \in A$, we see that both the process P and the controller C have transitions labelled with a from their initial locations P_1 and C_1. This results in the (synchronized) a-transition in the composite system $P|C$. In location P_3, P can do an a-action, but because $a \in A$ and C does not have an active a-transition in location C_3, this transition is blocked by C and therefore the transition is not present in $P|C$.

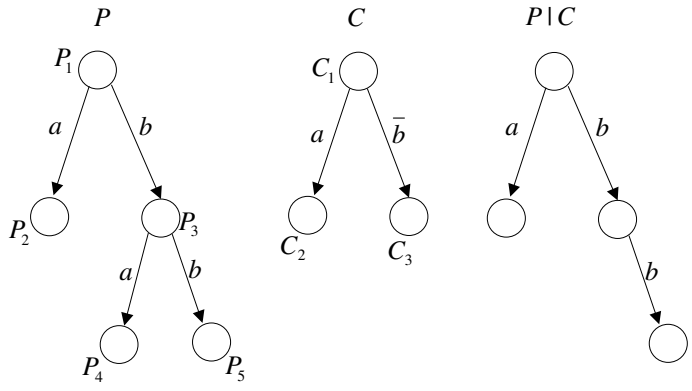

Figure 4.1: Process P controlled by C

Blocking as expressed in rule r1 can be used in supervisory control where a controller C blocks certain actions of the process P. Another situation where active actions must synchronize (i.e. are in A) is where two partners cooperate on a certain task. If two persons are chatting with each other, then they are cooperating on the chat-task so to say. In other words, both are chatting or both are not chatting, one cannot chat without the other (this situation will be described in the example in Section 4.2.5). For a situation where two persons P_1 and P_2 can both do a specific action (like hanging up the phone as described in Section 4.2.5) independently of the other person, then the action can be independently performed

(i.e. should not be in A).

For non-blocking-interaction, we need an active a with $a \notin A$ in one partner and a passive $\bar{a}$ in the other partner. Non-blocking interaction is now expressed by

$$r2.\frac{L_1 \xrightarrow{a} L_1', L_2 \xrightarrow{\bar{a}} L_2'}{L_1|A|L_2 \xrightarrow{a} L_1'|A|L_2'}(a \notin A),$$

which means that L_1 executes an a, which is observed by a $\bar{a}$-transition outgoing from L_2. In Figure 4.1 we see for example that the b-transition from location P_1 in P, is observed by the $\bar{b}$-transition in C. We also need rule r2', which is the mirrored rule of r2 (i.e. $L_1 \xrightarrow{\bar{a}} L_1', L_2 \xrightarrow{a} L_2'$ instead of $L_1 \xrightarrow{a} L_1', L_2 \xrightarrow{\bar{a}} L_2'$ etc.).

If L_2 can not observe a-actions (i.e. there is no outgoing $\bar{a}$-transition), L_1 should still be able to execute a, since this execution does not depend on whether or not some other process is observing the action. This is expressed by

$$r3.\frac{L_1 \xrightarrow{a} L_1', L_2 \not\xrightarrow{\bar{a}}}{L_1|A|L_2 \xrightarrow{a} L_1'|A|L_2}(a \notin A)$$

and its mirror rule r3'. In Figure 4.1 we see for example that P has a b-transition from location P_3, which cannot be observed by C, but which is still present in $P|C$.

Note that rule r3 says that if L_2 *can* observe a, then it also *will* observe a, i.e. if L_2 has a $\bar{a}$-transition, then it can not choose not to observe an a executed by the other component. This makes sense because when some system broadcasts a radio signal a and a receiver is able to receive the signal (i.e. it has a passive $\bar{a}$-transition), then we should not allow the possibility that the signal is broadcast while the receiver does not receive (i.e. does not synchronize its passive $\bar{a}$-action with the active a-action).

In the supervisory control context, observing as active/passive-synchronization means that the controller observes actions of the process. Outside the supervisory control context, this mechanism can be used for other kinds of observation (e.g. a person that observes an alarm signal as described in Section 4.2.5).

4.2.2 Step 2: Controlling the scope of the observations

Passive transitions are intended to observe active transitions. This means that outside an open-systems context (i.e. when the system is closed and will not interact with other systems), passive transitions are supposed not to be present, since there is nothing to observe. Suppose we have a system that consists of a process P and a controller C, where the controller observes the actions of P. One could reason that $|A|$ should be defined such that after composition, there are no passive transitions anymore (which is the case if we only consider rules r1,r2 and r3). But that would not be suitable for a broader composition context like modular supervisory control: Suppose we have a system with one process P and two controllers C_1 and C_2, where both C_1 and C_2 are concurrently observing P. If we define $C := C_1|C_2$ as the composed controller, then C should still observe P, with other words, C should still contain the passive transitions from C_1 and C_2 to observe P. This means that the

'real' observations happen when we compose C with P and not when we compose C_1 with C_2.

One solution one could think of is to indicate within the composition operator where the 'real' observations take place. We could use an observation set O and then $|_O$ (or together with A, $|_A^O|$) means that observations take place in this composition for the events in O. Then in the compound $(C_1|C_2)|_O P$ there are no passive transitions with labels from O anymore.

For the double-controller situation, $|_O$ seems to be a good solution. However, it seems that for modelling the following situation we need a different solution: Suppose three persons P_1, P_2 and P_3 are working on a problem. All persons start working independently on this problem. Once one of the persons found the solution, all three persons stop with the problem. This can be modelled as in Figure 4.2, where the signal *ready* is 'broadcast' by one of the persons as soon as this person solved the problem, and is then received by the others. In this situation, every P_i should be able to hear every P_j $(i \neq j)$. With $|_O$ we can not express this. (In $(P_1|P_2)|_O P_3$, P_1 does not observe P_2. In $(P_1|_O P_2)|P_3$, P_3 does not observe P_1 and P_2 etc.).

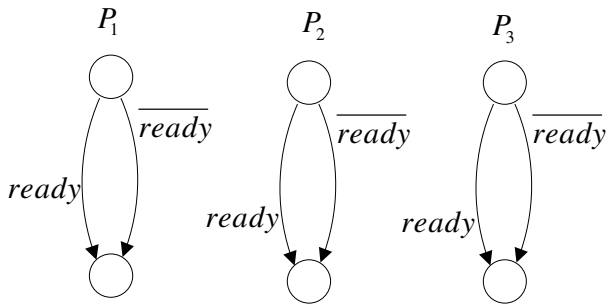

Figure 4.2: Situation where all persons can observe all others

Instead of using $|_O$, we use the closing operator $[\cdot]_C$. $[X]_C$ discards all passive transitions in X with labels from C. With this closing operator, we can solve both the double-controller and the 'three persons' problem. The composition rules for this operator are

$$r7.\frac{L \xrightarrow{a} L'}{[L]_C \xrightarrow{a} [L']_C}$$

$$r8.\frac{L \xrightarrow{\bar{a}} L'}{[L]_C \xrightarrow{\bar{a}} [L']_C}(\bar{a} \notin C).$$

With $[\cdot]_C$ we can control the scope of the observations. Now, $|$ should be defined such that in $X|Y$, X can observe Y (and vice versa), but also $X|Y$ can still observe Z in $(X|Y)|Z$. This is expressed by

$$r4.\frac{L_1 \xrightarrow{\bar{a}} L_1'}{L_1|A|L_2 \xrightarrow{\bar{a}} L_1'|A|L_2}$$

and mirror rule r4' which say that after the composition, every passive transition 'remains', such that the component to which this transition belongs, is still able to observe a new

component which might be added to the composite system in a new composition operation. If, for example, we know that X and Y only observe each other and will (or can) not observe Z or other components, then we could specify this as $[X|Y]|Z$. ($[\cdot]$ is shorthand for $[\cdot]_{\bar{\Sigma}}$ with $\bar{\Sigma}$ the set of all passive actions).

4.2.3 Step 3: Multi-way synchronization

Consider the modular supervisory control situation again where we have two controllers C_1 and C_2 and a process P. Now suppose that P has an action a which can be observed by both C_1 and C_2 (i.e. both controllers have $\bar{a}$-transitions). If we allow that both controllers can observe a concurrently (which is most natural), then we need to express the possibility of a multi-way synchronization (in this case: two passive actions and one active action). Another example (outside the supervisory control context) where we need a multi-way synchronization is the situation where an alarm signal in an office is heard (observed) by two different employees working in that office (this example is also described in Section 4.2.5).

The question now is whether there are situations we want to model that do not allow multi-way synchronizations. One such example (also considered in Section 4.2.5) is the telephone situation: a telephone in an office rings and only one of the employees may answer the call. Although both employees hear the telephone, only one may answer it (i.e. may synchronize its passive action with the active telephone signal).

We see that it is desirable to distinguish two types of passive actions: passive actions for which multi-way synchronization is allowed and passive actions for which this is not allowed. Therefore we introduce the set P, which contains all passive actions for which multi-way synchronization is allowed. The composition operator will now be denoted by $|^P_A|$. Multi-way synchronization is expressed by

$$r5.\frac{L_1 \xrightarrow{\bar{a}} L_1', L_2 \xrightarrow{\bar{a}} L_2'}{L_1|^P_A|L_2 \xrightarrow{\bar{a}} L_1'|^P_A|L_2'}(\bar{a} \in P),$$

which means that two passive $\bar{a}$-transitions, one in C_1 and one in C_2, synchronize, which results in a $\bar{a}$-transition for the composite system $C_1|C_2$. This synchronized passive transition can observe an a in P, which results in a new (multi-way) synchronized transition in $(C_1|C_2)|P$ which then expresses that C_1 and C_2 concurrently observe P.

If $a \in P$ and C_1 can observe a, but C_2 cannot observe a, then C_1 in $(C_1|C_2)|P$ should synchronize its passive $\bar{a}$ with the active a of P, while C_2 idles. This situation is expressed by

$$r6.\frac{L_1 \xrightarrow{\bar{a}} L_1', L_2 \not\xrightarrow{\bar{a}}}{L_1|^P_A|L_2 \xrightarrow{\bar{a}} L_1'|^P_A|L_2}(\bar{a} \in P)$$

and its mirror rule

$$r6'.\frac{L_1 \not\xrightarrow{\bar{a}}, L_2 \xrightarrow{\bar{a}} L_2'}{L_1|^P_A|L_2 \xrightarrow{\bar{a}} L_1|^P_A|L_2'}(\bar{a} \in P).$$

In the new situation (where we have introduced the set P), we can see that rule r4 (which expresses that passive transitions should interleave) only applies for passive actions not in P. Therefore r4 should be changed to

$$r4.\frac{L_1 \xrightarrow{\bar{a}} L_1'}{L_1|_A^P|L_2 \xrightarrow{\bar{a}} L_1'|_A^P|L_2}(\bar{a} \notin P).$$

With the set P as part of the operator, rules r1,r2 and r3 should be changed to

$$r1.\frac{L_1 \xrightarrow{a} L_1', L_2 \xrightarrow{a} L_2'}{L_1|_A^P|L_2 \xrightarrow{a} L_1'|_A^P|L_2'}(a \in A),$$

$$r2.\frac{L_1 \xrightarrow{a} L_1', L_2 \xrightarrow{\bar{a}} L_2'}{L_1|_A^P|L_2 \xrightarrow{a} L_1'|_A^P|L_2'}(a \notin A),$$

$$r3.\frac{L_1 \xrightarrow{a} L_1', L_2 \not\xrightarrow{\bar{a}}}{L_1|_A^P|L_2 \xrightarrow{a} L_1'|_A^P|L_2}(a \notin A).$$

Now rules r1 till r6 (and the mirror rules r2',r3',r4' and r6') form the structural operational semantics for $|_A^P|$ and rule r7 and r8 form the structural operational semantics for $[\cdot]_C$.

Note that the synchronization-mechanisms for active transitions and passive transitions are different. If an active action a is synchronizing (i.e. $a \in A$), then a component can execute the action only when the other component in the composition can execute the action (and vice versa). If a passive action $\bar{a}$ is synchronizing (i.e. $\bar{a} \in P$), then if both components can execute the action, they have to synchronize. However, if only one component can execute the action, the action can still be executed without the other component synchronizing with it. This is expressed in rule r6. For active-active synchronization we do not have an equivalent rule as r6. Because of this difference between the synchronization-mechanisms, we need to use different composition rules for both the active and the passive actions (i.e. we can not combine active and passive synchronization in the same rules).

4.2.4 Summary concerning $|_A^P|$

The operator $|_A^P|$ expresses the following interaction: Actions that are elements of the set A must synchronize. Suppose we have the context $P_1|_A^P|P_2$ and $a \in A$. Then P_1 can execute an a only if P_2 can execute an a at the same time (and vice versa). Execution of a in P_1 and P_2 will than happen synchronously. If $a \notin A$, then if P_1 can execute an a, it can be executed independently of P_2. Also, if $a \notin A$, passive $\bar{a}$-transitions in P_2 will be triggered by active a transitions in P_1 (i.e. they synchronize), and vice versa. If $\bar{a} \in P$, then two passive actions in P_1 and P_2 with this label must synchronize. This expresses multi-way synchronization which means that multiple passive transitions (one transition in each of the multiple components) can synchronize together with one active transition in another component. If $\bar{a} \notin P$, then passive transitions with the same label interleave, which expresses that only one passive transition may synchronize with an active transition. This means that when multiple components are ready to synchronize their $\bar{a}$-transition on an a-transition in another component, a (non-deterministic) choice should be made on which one is the one that is allowed to synchronize.

4.2.5 Example

In Figure 4.3 we see an example where we find back all interaction structures we described before: non-synchronizing active transitions, synchronizing active transitions, non-synchronizing passive transitions, synchronizing passive transitions and active-passive synchronization.

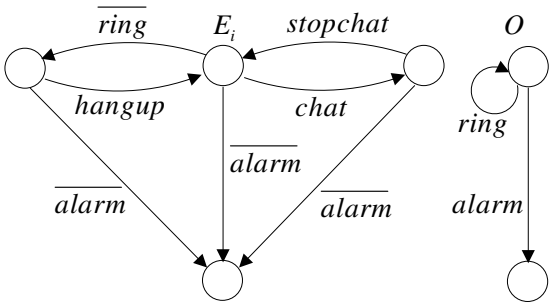

Figure 4.3: Example with different kinds of interaction

The example of Figure 4.3 concerns an office O with two employees E_1 and E_2. In the office there are two sources which can produce a signal: A telephone and an alarm. The telephone rings when somebody calls the office, the alarm fires when there is danger, which means that the employees should leave the office when they hear the alarm. Both the telephone and the alarm execute their signals independently from the employees. Therefore they are modelled as active transitions labelled *ring* and *alarm* respectively (see Figure 4.3). If the telephone rings, O makes a self-loop which means that O stays in the same location. This location has the meaning of 'normal-working-conditions'. If the alarm goes, O jumps to a second location which means 'dangerous-working-conditions'.

E_1 and E_2 have the same automaton-structure. From E_1 (or E_2), the employee can exhibit three different actions: He can chat with his fellow employee, he can pick up the phone and he can leave the office. Leaving the office only happens when the alarm goes off. This action is modelled as a passive transition which synchronizes with the independent active alarm-transition in O. We see that from every location, the employee can react on the alarm. When the phone rings, the employee can pick up the phone by synchronizing its passive *ring* event with the active *ring* from O. The employee hangs up with an active *hangup* event. The employee can start a chat with his office-mate via the active *chat* event. This should synchronize with an active *chat* event of the office-mate (both should be willing or able to chat). The chat will be ended with an active-active synchronization of *stopchat*.

From the model we see that if the employees are chatting, they first have to end the chat before one of them can pick up the phone. This means that they will probably miss the first *ring* signal, but can maybe interact on the second *ring* signal (the phone might give multiple *ring* signals when somebody calls). Also, if one employee is phoning, he first has to hang up before a chat can be started.

Where do we see the different kinds of interaction? The passive *ring* transitions of E_1 and

E_2 should be non-synchronizing since only one passive transition is allowed to synchronize with the active *ring* transition in O. The passive *alarm* transitions in E_1 and E_2 should synchronize because both employees react synchronously on the alarm. The active *hangup* transitions in E_1 and E_2 should be non-synchronizing (only one employee hangs up). The active *chat* and *stopchat* transitions in E_1 and E_2 should synchronize (both employees chat).

From the above follows that for the composition $E_1|_A^P|E_2$ we get $A = \{chat, stopchat\}$ and $P = \{\overline{alarm}\}$. For the composition $(E_1|^{\overline{alarm}}_{chat,stopchat}|E_2)|_P^A|O$ we get $A = \emptyset$ and $P = \bar{\Sigma}$. The total specification is then

$$(E_1|^{\overline{alarm}}_{chat,stopchat}|E_2)||O. \tag{4.1}$$

If (4.1) is the system that we want to analyze, then we could close down all observation channels (i.e. the passive transitions) with the $[\cdot]$ operator. Now suppose that (4.1) is only one chamber of a bigger office consisting of multiple chambers. Then this chamber is only one unit and lets call it U_1. The other units then are $U_2 = (E_1'|^{\overline{alarm}}_{chat,stopchat}|E_2')||O'$, $U_3 = (E_1''|^{\overline{alarm}}_{chat,stopchat}|E_2'')||O''$ etc., where $E_i = E_i' = E_i''$, $O = O' = O''$ etc. The telephones in each office are local. With other words, if a telephone rings in one office, only the employees in that office hear the phone and can answer it. The alarm however is global. If there is danger in the building, the alarm goes synchronously in all offices. To specify that the phones are local, we use $[\cdot]$ and get $[U_i]_{ring}$. To specify that the alarm is global, we use *alarm* as a synchronization event in the composition of the units. The total composition of the whole building is then

$$U_1'|_{alarm}|U_2'|_{alarm}|U_3'|_{alarm}|\cdots,$$

where $U_i' = [U_i]_{ring}$.

4.2.6 Supervisory control with $|_A^P|$

In this section we want to take a closer look at supervisory control. We will shortly describe the main concepts of supervisory control and thereafter we compare how specification of control systems can be done in our active/passive framework to how it can be done in a framework where there is only blocking-interaction.

In the supervisory control paradigm ([11]), the actions of a process can be observable or unobservable and they can be controllable or uncontrollable. Observable actions can be observed by the controller (and unobservable actions can not). Controllable actions can be controlled by the controller (and uncontrollable actions can not). This means that the controller can block these actions, i.e. can prevent them from happening.

If we specify the control system within an automata framework, then the process and the controller are modelled as two separate automata which can interact. The plant executing an action is then modelled as a transition (labelled with this action) from one process-location to another process-location. The controller observing an action is then modelled as a transition in the controller that synchronizes with the to-be-observed transition in the process.

Consider the process P in Figure 4.4 with a controllable/observable and b uncontrollable/observable. We want to control this process such that the behavior of P is restricted to

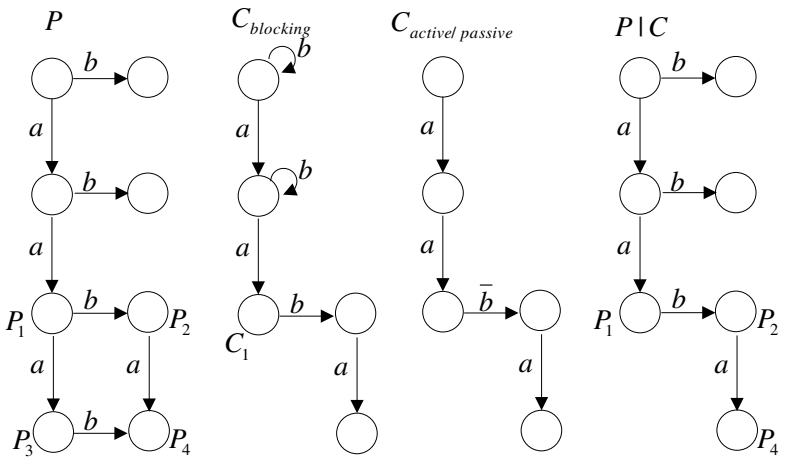

Figure 4.4: Control in only-blocking framework and in active/passive framework

$P|C$ in Figure 4.4.

First lets see how this can be done in the only-blocking-interaction framework. Since a and b are both observable, we mark them as synchronization actions (i.e. in the composition operation these actions must synchronize while non-synchronization actions will be interleaved). The controller that does the job is $C_{blocking}$ in Figure 4.4. We see that we need two self-loops on action b, because otherwise the two upper b-transitions of P are blocked and that is not what we want according to $P|C$. We see that, according to the specification $P|C$, transition $P_1 \xrightarrow{a} P_3$ of P is blocked because of the absence of an a-transition in $C_{blocking}$ at location C_1.

For the active/passive framework, $C_{active/passive}$ in Figure 4.4 does the job, because a is controllable (i.e. blockable), $a \in A$ and because b is not controllable, $b \notin A$. Blocking the a transition from P_1 to P_3 is done in the same way as in the only-blocking framework. The difference however is, that here we do not need the self-loops on b.

We could say that in the active/passive framework, the controller will observe only (by means of a passive transition) when it needs the information. In Figure 4.4, $C_{active/passive}$ observes $P_1 \xrightarrow{b} P_2$, but does not observe the upper b-transitions of P, because P may execute them without the controller 'knowing' it. In the only-blocking framework, the controller must also synchronize on the upper b-transitions, because otherwise they will be blocked. We think that this is an important advantage of using active/passive transitions: for uncontrollable/observable actions, transitions are only needed where observations are needed. If there is only blocking-interaction, transitions in the controller are needed everywhere the process executes an uncontrollable/observable action (otherwise they will be blocked and that is not allowed).

In fact we can say that in the supervisory control context, A (from $P|_A^P|C$) contains the controllable actions of the process P. Then, we resume: uncontrollable actions are observed by

passive transitions and controllable actions are observed and controlled by active transitions. Note that an active transition of C labelled with a controllable event of P is both controlling (it allows P to execute the action) and observing (the controller synchronizes this transition with the one from P).

4.2.7 Commutativity and associativity of $|_A^P|$

In this section we present the more technical result that $|_A^P|$ is commutative and associative.

Suppose that X, Y and Z are processes (automata). Suppose that L_1 and L_1' are locations of X, L_2 and L_2' are locations of Y and L_3 and L_3' are locations of Z. For the composite system $X|Y$ we then have locations $L_i|L_j$, with L_i, L_j locations of X and Y respectively. For the composite system $(X|Y)|Z$ we have locations $(L_i|L_j)|L_k$, with L_i, L_j, L_k locations of X, Y and Z respectively. etc. *Commutativity* of $|$ means that for all processes X and Y and for every transition $L_1|L_2 \xrightarrow{\alpha} L_1'|L_2'$ of $X|Y$, we have that $L_2|L_1 \xrightarrow{\alpha} L_2'|L_1'$ is a transition of $Y|X$ and vice versa (α may be active or passive). *Associativity* of $|$ means that for all processes X,Y and Z and for every transition $(L_1|L_2)|L_3 \xrightarrow{\alpha} (L_1'|L_2')L_3'$ of $(X|Y)|Z$, we have that $L_1|(L_2|L_3) \xrightarrow{\alpha} L_1'|(L_2'|L_3')$ is a transition of $X|(Y|Z)$ and vice versa.

Theorem: *$|_A^P|$ is commutative for all A and P. $|_A^P|$ is associative if and only if we have that $a \in A$ implies $\bar{a} \in P$ for all actions.*

Proof: Because we also have the symmetric rules 1' till 6' of 1 till 6, $|_A^P|$ is clearly commutative. For associativity, we first consider the case where P is full (the set of all labels), we do this by considering 20 cases (which are all cases that can bring forth a transition in the composite system): Independent of A, there are seven cases that bring forth a passive transition in the composition of three components (case 1-7 below). If $a \notin A$, there are 12 cases (case 8-19) that bring forth an active transition in the composition of three components. For the case that $a \in A$, there is only one case (case 20) where three components bring forth an active transition.

In the following table, we see implication arrows. Above these arrows is indicated which rules should be applied to get the result of that specific line in the table, where a zero means that no rule should be applied. For line 1 for example we have that for the left hand situation no rule should be applied for the in-bracket terms (which result in no-passive transitions for this composition) after which rule r6' should be applied which results in the passive transition after the second composition. For the right-hand side of line 1, we should apply rule r6' for the in-bracket term (which results in a passive transition for this composition), after which rule r6' should be applied again, which results in a passive transition after the second composition. We see that for the situation in line 1, we have associativity. This result is now proven in the following table for all possible situations.

1. $(\not\xrightarrow{\bar{a}}|_A^P|\not\xrightarrow{\bar{a}})|_A^P|\xrightarrow{\bar{a}}$ $\overset{0,r6'}{\Longrightarrow}$ $\xrightarrow{\bar{a}}$ $\overset{r6',r6'}{\Longleftarrow}$ $\not\xrightarrow{\bar{a}}|_A^P|(\not\xrightarrow{\bar{a}}|_A^P|\xrightarrow{\bar{a}})$
2. $(\not\xrightarrow{\bar{a}}|_A^P|\xrightarrow{\bar{a}})|_A^P|\not\xrightarrow{\bar{a}}$ $\overset{r6',r6}{\Longrightarrow}$ $\xrightarrow{\bar{a}}$ $\overset{r6,r6'}{\Longleftarrow}$ $\not\xrightarrow{\bar{a}}|_A^P|(\xrightarrow{\bar{a}}|_A^P|\not\xrightarrow{\bar{a}})$
3. $(\xrightarrow{\bar{a}}|_A^P|\not\xrightarrow{\bar{a}})|_A^P|\not\xrightarrow{\bar{a}}$ $\overset{r6,r6}{\Longrightarrow}$ $\xrightarrow{\bar{a}}$ $\overset{0,r6}{\Longleftarrow}$ $\xrightarrow{\bar{a}}|_A^P|(\not\xrightarrow{\bar{a}}|_A^P|\not\xrightarrow{\bar{a}})$
4. $(\not\xrightarrow{\bar{a}}|_A^P|\xrightarrow{\bar{a}})|_A^P|\xrightarrow{\bar{a}}$ $\overset{r6',r5}{\Longrightarrow}$ $\xrightarrow{\bar{a}}$ $\overset{r5,r6'}{\Longleftarrow}$ $\not\xrightarrow{\bar{a}}|_A^P|(\xrightarrow{\bar{a}}|_A^P|\xrightarrow{\bar{a}})$

5. $(\xrightarrow{\bar{a}}|^P_A|\not\xrightarrow{\bar{a}})|^P_A|\xrightarrow{\bar{a}} \overset{\text{r6,r5}}{\Longrightarrow} \xrightarrow{\bar{a}} \overset{\text{r6}'\text{,r5}}{\Longleftarrow} \xrightarrow{\bar{a}}|^P_A|(\not\xrightarrow{\bar{a}}|^P_A|\xrightarrow{\bar{a}})$
6. $(\xrightarrow{\bar{a}}|^P_A|\xrightarrow{\bar{a}})|^P_A|\not\xrightarrow{\bar{a}} \overset{\text{r5,r6}}{\Longrightarrow} \xrightarrow{\bar{a}} \overset{\text{r5,r6}}{\Longleftarrow} \xrightarrow{\bar{a}}|^P_A|(\xrightarrow{\bar{a}}|^P_A|\not\xrightarrow{\bar{a}})$
7. $(\xrightarrow{\bar{a}}|^P_A|\xrightarrow{\bar{a}})|^P_A|\xrightarrow{\bar{a}} \overset{\text{r5,r5}}{\Longrightarrow} \xrightarrow{\bar{a}} \overset{\text{r5,r5}}{\Longleftarrow} \xrightarrow{\bar{a}}|^P_A|(\xrightarrow{\bar{a}}|^P_A|\xrightarrow{\bar{a}})$

$a \notin A$:

8. $(\not\xrightarrow{\bar{a}}|^P_A|\not\xrightarrow{\bar{a}})|^P_A|\xrightarrow{a} \overset{\text{0,r3}'}{\Longrightarrow} \xrightarrow{a} \overset{\text{r3}'\text{,r3}'}{\Longleftarrow} \not\xrightarrow{\bar{a}}|^P_A|(\not\xrightarrow{\bar{a}}|^P_A|\xrightarrow{a})$
9. $(\not\xrightarrow{\bar{a}}|^P_A|\xrightarrow{a})|^P_A|\not\xrightarrow{\bar{a}} \overset{\text{r3}'\text{,r3}}{\Longrightarrow} \xrightarrow{a} \overset{\text{r3,r3}'}{\Longleftarrow} \not\xrightarrow{\bar{a}}|^P_A|(\xrightarrow{a}|^P_A|\not\xrightarrow{\bar{a}})$
10. $(\xrightarrow{a}|^P_A|\not\xrightarrow{\bar{a}})|^P_A|\not\xrightarrow{\bar{a}} \overset{\text{r3,r3}}{\Longrightarrow} \xrightarrow{a} \overset{\text{0,r3}}{\Longleftarrow} \xrightarrow{a}|^P_A|(\not\xrightarrow{\bar{a}}|^P_A|\not\xrightarrow{\bar{a}})$
11. $(\not\xrightarrow{\bar{a}}|^P_A|\xrightarrow{a})|^P_A|\xrightarrow{\bar{a}} \overset{\text{r3}'\text{,r2}}{\Longrightarrow} \xrightarrow{a} \overset{\text{r2,r3}'}{\Longleftarrow} \not\xrightarrow{\bar{a}}|^P_A|(\xrightarrow{a}|^P_A|\xrightarrow{\bar{a}})$
12. $(\not\xrightarrow{\bar{a}}|^P_A|\xrightarrow{\bar{a}})|^P_A|\xrightarrow{a} \overset{\text{r6}'\text{,r2}'}{\Longrightarrow} \xrightarrow{a} \overset{\text{r2}'\text{,r3}'}{\Longleftarrow} \not\xrightarrow{\bar{a}}|^P_A|(\xrightarrow{\bar{a}}|^P_A|\xrightarrow{a})$
13. $(\xrightarrow{a}|^P_A|\not\xrightarrow{\bar{a}})|^P_A|\xrightarrow{\bar{a}} \overset{\text{r3,r2}}{\Longrightarrow} \xrightarrow{a} \overset{\text{r6}'\text{,r2}}{\Longleftarrow} \xrightarrow{a}|^P_A|(\not\xrightarrow{\bar{a}}|^P_A|\xrightarrow{\bar{a}})$
14. $(\xrightarrow{\bar{a}}|^P_A|\not\xrightarrow{\bar{a}})|^P_A|\xrightarrow{a} \overset{\text{r6,r2}'}{\Longrightarrow} \xrightarrow{a} \overset{\text{r3}'\text{,r2}'}{\Longleftarrow} \xrightarrow{\bar{a}}|^P_A|(\not\xrightarrow{\bar{a}}|^P_A|\xrightarrow{a})$
15. $(\xrightarrow{a}|^P_A|\xrightarrow{\bar{a}})|^P_A|\not\xrightarrow{\bar{a}} \overset{\text{r2,r3}}{\Longrightarrow} \xrightarrow{a} \overset{\text{r6,r2}}{\Longleftarrow} \xrightarrow{a}|^P_A|(\xrightarrow{\bar{a}}|^P_A|\not\xrightarrow{\bar{a}})$
16. $(\xrightarrow{\bar{a}}|^P_A|\xrightarrow{a})|^P_A|\not\xrightarrow{\bar{a}} \overset{\text{r2}'\text{,r3}}{\Longrightarrow} \xrightarrow{a} \overset{\text{r3,r2}'}{\Longleftarrow} \xrightarrow{\bar{a}}|^P_A|(\xrightarrow{a}|^P_A|\not\xrightarrow{\bar{a}})$
17. $(\xrightarrow{\bar{a}}|^P_A|\xrightarrow{\bar{a}})|^P_A|\xrightarrow{a} \overset{\text{r5,r2}'}{\Longrightarrow} \xrightarrow{a} \overset{\text{r2}'\text{,r2}'}{\Longleftarrow} \xrightarrow{\bar{a}}|^P_A|(\xrightarrow{\bar{a}}|^P_A|\xrightarrow{a})$
18. $(\xrightarrow{\bar{a}}|^P_A|\xrightarrow{a})|^P_A|\xrightarrow{\bar{a}} \overset{\text{r2}'\text{,r2}}{\Longrightarrow} \xrightarrow{a} \overset{\text{r2,r2}'}{\Longleftarrow} \xrightarrow{\bar{a}}|^P_A|(\xrightarrow{a}|^P_A|\xrightarrow{\bar{a}})$
19. $(\xrightarrow{a}|^P_A|\xrightarrow{\bar{a}})|^P_A|\xrightarrow{\bar{a}} \overset{\text{r2,r2}}{\Longrightarrow} \xrightarrow{a} \overset{\text{r5,r2}}{\Longleftarrow} \xrightarrow{a}|^P_A|(\xrightarrow{\bar{a}}|^P_A|\xrightarrow{\bar{a}})$.

$a \in A$:

20. $(\xrightarrow{a}|^P_A|\xrightarrow{a})|^P_A|\xrightarrow{a} \overset{\text{r1,r1}}{\Longrightarrow} \xrightarrow{a} \overset{\text{r1,r1}}{\Longleftarrow} \xrightarrow{a}|^P_A|(\xrightarrow{a}|^P_A|\xrightarrow{a})$.

Now suppose that an action $a \in A$ and $\bar{a} \notin P$. Then, for the operator $|^P_A|$, only rules r1 and r4 apply. r1 says that active a-transitions must synchronize. It can easily be seen then that for three components all components should have an active a-transition, otherwise the composition result will not have an active a-transition. This is true for both composition orders (left- and right-hand). This means that so far we have associativity for this operator. r4 says that passive $\bar{a}$-transitions are interleaving (i.e. they are independent of active and passive transitions in other components). Here it can also clearly be seen that there can not be a composition case where this rule will violate associativity. Thus, if $a \in A$ and $\bar{a} \notin P$, then $|^P_A|$ is associative.

Now suppose there exists $\bar{a} \notin P$ together with $a \in A$. Then the following is a counterexample for associativity:

$$(L_1 \xrightarrow{a} L_1'|^P_A|L_2 \xrightarrow{\bar{a}} L_2')|^P_A|L_3 \xrightarrow{\bar{a}} L_3' \quad \overset{\text{r2,r2}}{\Longrightarrow} \quad (L_1|^P_A|L_2)|^P_A|L_3 \xrightarrow{a} (L_1'|^P_A|L_2')|^P_A|L_3',$$

$$L_1 \xrightarrow{a} L_1'|^P_A|(L_2 \xrightarrow{\bar{a}} L_2'|^P_A|L_3 \xrightarrow{\bar{a}} L_3') \quad \Longrightarrow \quad (L_1|^P_A|L_2)|^P_A|L_3 \not\xrightarrow{a} (L_1'|^P_A|L_2')|^P_A|L_3'.$$

We conclude that $|^P_A|$ is associative if and only if $a \in A$ implies $\bar{a} \in P$ for all actions. $\square$

If, according to the above theorem, $|^P_{A_1}|$ and $|^P_{A_2}|$ are associative operators, we have $(X|^P_{A_i}|Y)|^P_{A_i}|Z = X|^P_{A_i}|(Y|^P_{A_i}|Z)$ $(i = 1, 2)$ and therefore we could write $X|^P_{A_i}|Y|^P_{A_i}|Z$ instead. Note, however, that in general $(X|^P_{A_1}|Y)|^P_{A_2}|Z \neq X|^P_{A_1}|(Y|^P_{A_2}|Z)$ just as in general we do not have $(X|A_1|Y)|A_2|Z \neq X|A_1|(Y|A_2|Z)$ in for example CSP (see [5]) or LOTOS (see [6]).

4.3 Conclusions

In this chapter we used active and passive transitions to model both one-way and two-way interaction between processes. We motivated that the use of active/passive transitions is particularly interesting for modelling supervisory control systems. Supervisory controllers have two distinct actions: observing and controlling, which can be modelled naturally with passive and active transitions respectively. With the use of active/passive transitions, we are able to avoid the problems that arise in the two-way framework when it comes to modelling uncontrollable process actions. We introduced the composition operator $|^P_A|$ for the active/passive framework by means of structural operational rules. With the active/passive framework and the operators $[\cdot]_C$ and $|^P_A|$, we have given two important tools for the modular specification of control systems. First, with $[\cdot]_C$ we can control the scope of the observations within the composite system. Secondly, with $|^P_A|$ we can establish synchronization of observations (which results in multi-may synchronizations) via the set P. We think that many phenomena, in particular supervisory control systems, can be modelled naturally within the active/passive framework together with $|^P_A|$ and $[\cdot]_C$.

Chapter 5

CPDP and DCPN

Besides CPDP, another model has been developed for compositional specification of PDPs. This model is called DCPN which stands for Dynamically Colored Petri Nets (see [3]). In this chapter we will compare the CPDP-automaton model with the DCPN-Petri-net model.

Both the CPDP and the DCPN models are equivalent with the PDP model in the sense that every PDP can be transformed into a CPDP/DCPN and every CPDP/DCPN can be transformed into a PDP. This transformation is such that the stochastic processes of the PDP and its transformed CPDP/DCPN (and vice versa) are the same.

5.1 Compositional modelling with CPDP and DCPN

Both CPDP and DCPN allow compositional modelling in the sense that a complex system can be built by first specifying its parts (or components) followed by specifying the interrelation between the parts.

5.1.1 DCPN

In the DCPN model, two DCPNs can be composed by connecting them with arcs (and if desirable with extra places and transitions). There are two types of arcs that can be used for this interconnection: Enabling arcs and inhibitor arcs. If a transition in a component has incoming enabling arcs from other components, then this transition is enabled if each of the components has a token in the place from which the enabling arc leaves. If a transition in a component has an incoming inhibitor arc, then the transition is forbidden to be executed when there is a token in the place on the other end of the inhibitor arc.

Interaction in DCPN can be seen as sending and receiving of information about the absence, presence and characteristics of tokens in places. In principle every place (in any component) could send its token-information to any transition in the other components (by drawing the corresponding arcs). This information can then be used in several ways: presence of a token can be used for enabling or disabling the transition, characteristics (i.e. the color) of the token can be used for determining the distribution of tokens (and their values) just after the transition.

Communication consists in fact of a continuous flow of information (i.e. a transition gets input-information from its incoming enabling/inhibitor arcs at any moment).

With some clever use (of maybe extra transitions, places etc.), quite rich interaction structures can be built. Two places in two components can for example enable each others transitions (assumed that both places have one outgoing transition) which gives the effect of two components waiting for each other before they do a synchronized transition. In this way the classical synchronization idea (of most process algebra literature) can be simulated.

5.1.2 CPDP

Communication in CPDP (as it is established in [13]) is done via synchronizing passive and active transitions. The idea is simple: If in one component an active (or boundary-hit) transition is executed, then all other components that are able to execute a passive transition with the same label will execute these passive transitions synchronously with the active one. In this way a component informs the other components that it executed an active (or spontaneous) transition with a specific label.

Why do we put the communication labels on the boundary-hit and the spontaneous transitions? The idea is that the switching of a mode (i.e. jumping to another location via a boundary-hit or a spontaneous transition) is a relevant action for other components. It can be relevant for other components to know when and how the other components switch to other modes/locations. This is exactly what is expressed by the CPDP communication mechanism: Via passive transitions, components can receive information of the active (or spontaneous) transitions of other components.

In Chapter 4 we developed a parallel composition operator which can express active-passive interaction, but which can also express other kinds of communication (like active-active interaction: Two components waiting for each other before they proceed). For this active-active interaction, it does not seem very natural anymore to put the event-labels on the boundary-hit and spontaneous transitions, because boundary-hit and spontaneous transitions 'just happen' (from a PDP-point of view) and, intuitively, they do not wait for other components. One possible new way of establishing communication for CPDPs, which we are thinking of now, is making a total distinction between PDP-phenomena (like boundary-hit jumps and spontaneous jumps) and communication-phenomena (i.e. transitions which are only intended to communicate something). We would then get an orthogonal split-up of PDP-transitions (boundary-hit/spontaneous) and communication-transitions (active/passive). In extended-CPDP, in [13], we enhanced the boundary-hit transitions with extra guards, which also gave the possibility of active-active interaction. However, in this way the transition (with guard) is not exactly a boundary-hit transition anymore. Boundary-hit transitions are then a special case of these guarded transitions, where the guards are equal to the boundaries of the invariants.

Thus, we still have to think about how to exactly define the communication mechanism: Do we want to specify boundary-hit transitions as special cases of guarded transitions, or do we want to split PDP-transitions from communication-transitions? In the first case, the number of different transitions is smaller (although we need to introduce guards), but in the second

case the connection with PDPs can be made easier and more intuitive because the PDP-behavior corresponds exactly with the PDP-transitions (and not with the communication-transitions).

5.2 Comparison CPDP - DCPN

We look at two different aspects for the comparison of the CPDP-model and the DCPN-model: Synchronization and reset-maps.

5.2.1 Synchronization

Comparing the communication mechanisms of CPDP and DCPN is difficult because their communication-mechanisms are rather different. In DCPN, the tokens in some places pre-enable transitions that are connected to other places via ordinary arcs. Once a transition is enabled, it will be executed, alone. In CPDP, communication does not take place via enabling or disabling of transitions, but by the synchronization of transitions. An active transition in one component can force passive transitions in other components to synchronize with it. Still, we could say that both communication mechanisms are equivalent in the sense that transitions in one component influence or force transitions in other components: In DCPN, a transition in one component changes the distribution of its tokens, which may change the pre-enabledness of transitions in other components. That may have as consequence that immediate transitions in these other components are forced to be executed immediately. In some sense this means synchronization, since the transitions are executed at the same time. On the other hand, the transitions are executed in a specific order which is not the case for synchronizing transitions in CPDP.

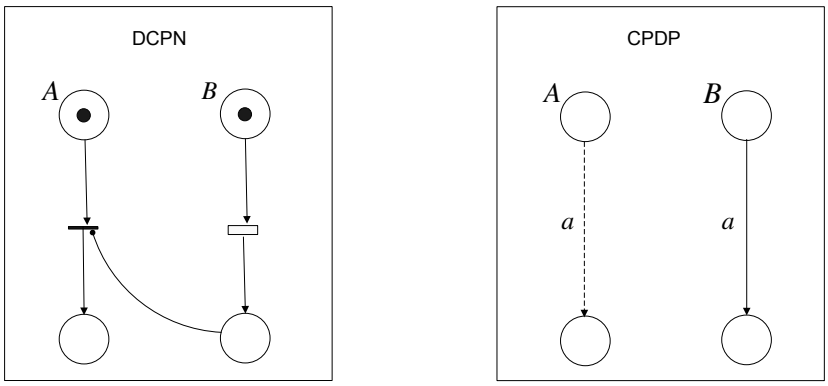

Figure 5.1: Communication for CPDP and DCPN

In Figure 5.1 for example, we see CPDP $A||B$ (i.e. the composition of A and B), where the passive transition of A will synchronize on the active transition of B. The DCPN with components A and B is equivalent: Once the guard-transition (i.e. the analogy of active transition as boundary-hit) in B is executed, the immediate transition in A is enabled and will be executed immediately. Therefore the two transitions are executed at the same time, but still they have a specific order of execution where the transition in B is executed first

while we do not have such an order of execution in the corresponding CPDP.

Note that (in general) changing the pre-enabledness of guard- and delay-transitions in DCPN means, in CPDP-terms, jumping to another mode (location) where new jump-rates and boundaries are active.

5.2.2 Reset-maps

For DCPN, the colors of the tokens from the input-places of a specific transition determine the reset-map (i.e. firing measure in DCPN terminology). This means that different color-values in these place may result in totally different reset-maps. We could say therefore that there is communication of the continuous process data via these reset-maps in DCPN. In the old-CPDP model, described in [13], this kind of continuous-communication was not possible. In the extended-CPDP model, also described in [13], continuous variables were able to communicate (composition in the behavioral sense [15]) and also the reset-maps in a specific component could be influenced by the values of the continuous processes of other components.

Thus, we could say that also here CPDP and DCPN seem to have more or less the same modelling-power (although the communication mechanisms are rather different for CPDP and DCPN), however, the mechanism of the reset-maps in extended-CPDP is not really satisfactory from a compositionality point of view: A reset-map (as it is defined now in CPDP) is part of a specific transition of a specific component. Thus, a specific reset-map is a characteristic of one component. From a compositionality point of view, this should ideally mean that this reset map should not contain information about other components, or about the context of the composition (since we want that a certain component can be plugged-in in many different composition contexts). But as it is defined now in extended-CPDP, the reset-maps need inputs from specific variables. This means that the composition context should be such that these variables should indeed be present in one of the components. Therefore, in this way the reset-map assumes or claims a specific composition context and this results in the need for compatibility conditions: components should be compatible with each other, otherwise the process is not well-defined. If for example a reset-map of one component needs information of the current value of variable V, but there is no component that indeed determines the differential equations for V, then the process is ill-defined or at least underspecified (in the sense that we need another component that will specify the flow of the continuous variable V). We should therefore find a new way which is satisfactory from a compositionality point of view and where it is still possible to communicate data of the continuous variables.

5.3 Syntactical representation of composite systems

Both CPDPs and DCPNs are syntactical objects. Therefore, we have to be clear about the behavior (or semantics) of such syntactical objects. For CPDPs and DCPNs this is done by specifying how stochastic executions can be generated. In Chapter 2 we also made the connection of CPDPs with PDPs, which gives the possibility to define the behavior of a CPDP as the behavior of its corresponding PDP. The same thing is also done for DCPNs in [3].

For CPDPs, we defined a composition operator, denoted by $||$. (For extended-CPDP we defined in fact multiple composition operators). If A and B are CPDPs, then $A||B$ is the

CPDP that expresses the behavior of the composite system with as components A and B. A and B are syntactical objects, but also $A||B$ is a syntactical object. We could say that $A||B$ is a syntactical expression that represents the composite CPDP that consists of components A and B. For $A||B$ (or for $A||B||C||D$) it is clear for the composite system what its components are: A and B (or A,B,C and D). Thus we could say that the compositional structure of the composite system is reflected in the syntactical expression. This makes it possible to do compositional analysis on the syntactical expression.

For DCPN, composite systems can be modelled as component-Petri-Nets (i.e. component-DCPNs) which are then connected by extra arcs (and if desirable extra places). The composite system is then again a DCPN (which is also a syntactical object). The difference with CPDP is that there is no syntactical expression which can reveal the composition structure of the composite system. Although specification of DCPNs can be done in a modular way, with rich interaction structures, once the composite DCPN is specified, we do not have a formal means (like a syntactical expression) which can reveal the composition structure.

Therefore, from a compositional modelling point of view, DCPN seems to be very powerful. From a compositional analysis point of view (e.g. state space reduction by bisimulation), a formal description of the composition structure of a composite system seems to be needed. We think that using composition operators (with a clear operational semantics), which connect components on a syntactical level, is a good choice if one has compositional analysis in mind as one of the main targets.

Chapter 6

Overview and future research

Finally in this report, we will give an overview of the research done in WP4 of the Hybridge project and we will point out directions for future research.

Overview:

- We developed an automaton-model called CPDP. CPDPs can communicate with each other by sending and receiving signals (discrete events) when either a boundary-hit happens or when a spontaneous transition happens. Semantical equivalence of PDPs and CPDPs has now been formally proven (by means of indistinguishable stochastic processes).
- Steps were taken to make an extended model of the CPDP, which includes more powerful discrete-event communication and also interaction between continuous variables. However, from a compositionality point of view some problems arise in this extended model concerning the stochastic reset maps.
- The composition operator $|_A^P|$ has been developed and discussed in this report. This operator allows different kinds of interaction and is potentially more powerful than the one we defined for extended-CPDP. We tried to make clear that this operator might have a natural use in the context of supervisory control systems.
- We defined a bisimulation notion for PDPs that are enhanced with output functions, where bisimular PDPs have equivalent output-processes.

Future research:

- We will explore the idea of redefining the syntax of the CPDP model such that PDP-transitions (boundary-hits, spontaneous jumps) and communication-transitions are split instead of combined as it is now (where boundary-hit transitions carry a communication label, etc.).
- We want to incorporate the $|_A^P|$ operator in the CPDP-model. Here we might profit from the split-up suggested in the previous item. Once this is done, we want to find the conditions under which a composite CPDP (composed with these new operators) behaves as a PDP.

- We need to find a way to bring communication of data of the continuous process in the CPDP-model. This should be done such that this data-communication can be formalized by a composition operator like (an extended version of) $|_A^P|$. The way it is done in extended-CPDP seems to be problematic. To have equivalent modelling power as DCPN, this needs to be done.
- Compositional analysis: We will continue our work on bisimulation for CPDPs. Can we find a structural definition of bisimulation, such that maximal bisimulations can effectively be calculated?

Bibliography

[1] M. H. A. Davis, *Piecewise deterministic markov processes: a general class of non-diffusion stochastic models*, Journal Royal Statistical Soc. (B) **46** (1984), 353–388.

[2] ———, *Markov models and optimization*, Chapman & Hall, London, 1993.

[3] M. H. C. Everdij and H. A. P. Blom, *Modelling hybrid state Markov processes through dynamically and stochastically coloured Petri nets*, Tech. Report D4.2 of Hybridge project IST-2001-32460, 2004.

[4] H. Hermanns, *Interactive markov chains*, Ph.D. thesis, Universität Erlangen Nürnberg, 1998.

[5] C.A.R. Hoare, *Communicating sequential processes*, Prentice-Hall, 1985.

[6] R. Langerak, *Transformations and semantics for lotos*, Ph.D. thesis, Universiteit Twente, 1992.

[7] N. A. Lynch, R. Segala, and F. W. Vaandrager, *Hybrid I/O automata revisited*, Lecture Notes in Computer Science **2034** (2001), 403–414.

[8] N. A. Lynch and M. R. Tuttle, *An introduction to input/output automata*, CWI Quarterly **2** (1988), no. 3, 219–246.

[9] R. Milner, *Communication and concurrency*, Prentice Hall, 1989.

[10] K. Prasad, *A Calculus of Broadcasting Systems*, Proc. 16th Colloquium on Trees in Algebra and Programming, vol. 493, 1991, pp. 338–358.

[11] P.J. Ramadge and W.M. Wonham, *The control of discrete event systems*, Proceedings of the IEEE **77** (1989), no. 1, 81–98.

[12] S. N. Strubbe, A. A. Julius, and A. J. van der Schaft, *Communicating piecewise deterministic markov processes*, Preprints Conference on Analysis and Design of Hybrid Systems ADHS 03, 2003, pp. 349–354.

[13] S.N. Strubbe, *Cpdp: A compositional framework for stochastic hybrid systems of the pdp type*, Tech. report, Twente University, 2003.

[14] A.J. van der Schaft, *Bisimulation of dynamical systems*, HSCC, Lecture Notes in Computer Science, vol. 2993, Springer, 2004, pp. 555–569.

[15] J. C. Willems, *On interconnections, control and feedback*, IEEE Transactions on Automatic Control **42** (1997), 326–339.