Proceedings of the 2006 American Control Conference
Minneapolis, Minnesota, USA, June 14-16, 2006

Value passing for Communicating Piecewise Deterministic Markov Processes

Stefan Strubbe, Arjan van der Schaft and Agung Julius

Abstract— **In this paper we extend the CPDP model, which is used for compositional specification of PDP-type stochastic hybrid systems, to the value passing CPDP model. With value passing we can express communication of values of continuous variables between CPDP components. We show that the class of value passing CPDPs is closed under composition. We illustrate the use of value passing CPDPs by modelling an Air Traffic Management system as a network of interacting value passing CPDPs.**

I. Introduction

In [4], [6], the modelling framework CPDP is introduced. CPDP stands for Communicating Piecewise Deterministic Markov Process. With CPDPs we can model PDP-type systems, which form a broad class of stochastic hybrid systems [1], [2], in a compositional way.

The interaction power of composition of CPDPs as defined in [6], is not strong enough to specify interactions where communication of values of continuous variables is involved. In this paper we extend the CPDP framework with value passing. We show that this idea of value passing can be 'coded' in the active transitions. We show that composition of value passing CPDPs can be formalized in the same way as it is done for CPDPs: by using a composition operator $|_A^P|$. We show that the result of composing two (or more) value passing CPDPs is again a CPDP.

For the biggest part, this paper is written to illustrate how value passing can be used and which types of interaction can be expressed through it. This illustration is done by specifying a rather complex Air Traffic Management system. This example shows how value passing can be used to communicate and store data/signals between components.

The paper is organized as follows. In Section II, we define CPDPs and its composition. In Section III we show how the CPDP model and its composition can be extended to the value passing CPDP model. In Section IV we give a detailed description of an Air Traffic Management system, modelled as a composition of value passing CPDPs. The main aim of this paper is to illustrate the use of value passing. For technical results concerning value passing CPDPs, we refer to [7].

This work was supported by the EU project Hybridge

Stefan Strubbe is with the Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente. Arjan van der Schaft is with the Institute for Mathematics and Computer Science, University of Groningen. Agung Julius is with the University of Pennsylvania. s.n.strubbe@math.utwente.nl, a.j.van.der.schaft@math.rug.nl, agung@seas.upenn.edu

II. CPDPs

We give the formal definition of CPDP as an automaton.

Definition 2.1: A CPDP is a tuple $(L,V,\nu,W,\omega,F,G,\Sigma,\mathscr{A},\mathscr{P},\mathscr{S})$, where: L is a set of locations. V is a set of state variables. With $d(v)$ for $v \in V$ we denote the dimension of variable v. $v \in V$ takes its values in $\mathbb{R}^{d(v)}$. W is a set of output variables. With $d(w)$ for $w \in W$ we denote the dimension of variable w. $w \in W$ takes its values in $\mathbb{R}^{d(w)}$. $\nu : L \to 2^V$ maps each location to a subset of V, which is the set of state variables of the corresponding location. $\omega : L \to 2^W$ maps each location to a subset of W, which is the set of output variables of the corresponding location. F assigns to each location l and each $v \in \nu(l)$ a mapping from $\mathbb{R}^{d(v)}$ to $\mathbb{R}^{d(v)}$, i.e. $F(l,v) : \mathbb{R}^{d(v)} \to \mathbb{R}^{d(v)}$. $F(l,v)$ is the vector field that determines the evolution of v for location l (i.e. $\dot{v} = F(l,v)$ for location l). G assigns to each location l and each $w \in \omega(l)$ a mapping from $\mathbb{R}^{d(v_1)+\cdots+d(v_m)}$ to $\mathbb{R}^{d(w)}$, where v_1 till v_m are the state variables of location l. $G(l,w)$ determines the output equation of w for location l (i.e. $w = G(l,w)$). Σ is the set of communication labels. $\bar{\Sigma}$ denotes the 'passive' mirror of Σ and is defined as $\bar{\Sigma} = \{\bar{a} | a \in \Sigma\}$. $\mathscr{A}$ is a finite set of active transitions and consists of five-tuples (l,a,l',G,R), denoting a transition from location $l \in L$ to location $l' \in L$ with communication label $a \in \Sigma$, guard G and reset map R. G is a closed subset of the state space of l. The reset map R assigns to each point in G for each variable $v \in \nu(l')$ a probability measure on the state space of v for location l'. $\mathscr{P}$ is a finite set of passive transitions of the form $(l,\bar{a},l',R)$. R is defined on the state space of l (as the R of an active transition is defined on the guard space). $\mathscr{S}$ is a finite set of spontaneous transitions and consists of four-tuples (l,λ,l',R), denoting a transition from location $l \in L$ to location $l' \in L$ with jump-rate λ and reset map R. The jump rate λ (i.e. the Poisson rate of the Poisson process of the spontaneous transition) is a mapping from the state space of l to $\mathbb{R}_+$. R is defined on the state space of l as it is done for passive transitions.

We introduce some notation. We call an active transition with event $a \in \Sigma$ an a-transition and we call a passive transition with event $\bar{a} \in \bar{\Sigma}$ a $\bar{a}$-transition. For a CPDP X with $v \in V_X$, where V_X is the set of state variables of X, we call $\mathbb{R}^{d(v)}$ the state space of state variable v. We call $\{(v = r) | r \in \mathbb{R}^{d(v)}\}$ the valuation space of v and each $(v = r)$ for $r \in \mathbb{R}^{d(v)}$ is called a valuation. We call $val(l) := \{(v_1 = r_1, v_2 = r_2, \cdots, v_m = r_m) | r_i \in \mathbb{R}^{d(v_i)}\}$, where v_1 till v_m are the variables from $\nu(l)$, the valuation space or state space

of location l and each $(v_1 = r_1, \cdots, v_m = r_m)$ is called a valuation or state of l. We call $\{(l,x)|l \in L, x \in val(l)\}$ the (hybrid) state space of the CPDP with location set L and valuation spaces $val(l)$. With the output variables (instead of state variables) we define in the same way output valuations, output space of location l and the (hybrid) output space of the CPDP. If a state x lies in the guard G_α of active transition α, then we say that α is enabled at x. We say that a passive transition α is enabled at x if l_x, the location of x, is the origin location of α. We say that a transition leaves location l if l is the origin location of that transition.

A reset map R of a CPDP consists of an indexed set of probability measures (i.e. R assigns to each state x of a location l a probability measure). We call the probability measures of a reset map *reset measures*. A reset map/measure probabilistically resets the state variables of a specific location. We call this specific location the *target location* of the reset map/measure.

To understand how CPDPs 'run', we give a very brief informal semantics for CPDP now. A complete and formal semantics for CPDP can be found in [7]. We assume that a CPDP $X = (L,V,\nu,W,\omega,F,G,\Sigma,\mathscr{A},\mathscr{P},\mathscr{S})$ starts at initial hybrid state (l,x), with $l \in L$ and $x \in val(l)$. Then, the continuous state x (which consists of valuations for the variables in $\omega(l)$), evolves along the vector fields $F(l)(v)$ for all variables $v \in \omega(l)$. Each spontaneous transition α with origin location l, may cause a jump at any time. The jump rate $\lambda(x)$ of the spontaneous transition at continuous state x determines the probability of jumping at state x. Roughly said, the probability that a spontaneous transition with jump rate $\lambda(x)$ jumps within Δt time units after the start at state (l,x) equals approximately $\lambda(x)\Delta t$ for Δt small enough. If a spontaneous transition α jumps at some state (l,x') (i.e. the state has already evolved from x to x'), then the new location equals the target location of α and the new continuous state (in the new location) is determined by the probability measure of $R_\alpha(x')$, where R_α is the reset map of α. Besides a jump caused by a spontaneous transition, a jump can also be caused by an active transition: if after some time the state has evolved to state x' and if x' lies in the guard of active transition α, then α may be executed and the target location and state are determined by the target location of α and the reset map of α. Note that there is non-determinism in the model here: at state x', α may be executed, but α is not forced to be executed. A passive transition with origin location l may be executed at all states in location l. The target location and state are then determined by the target location and the reset map of the passive transition. Passive transitions need to be triggered by active transitions from other CPDPs in a composition context. This interaction mechanism between CPDPs is defined in the following definition where the composition of two CPDPs is determined. After the definition, we give a small explanation of the composition rules. For a full explanation of the composition operator $|^P_A|$ we refer to [5] or [7].

Definition 2.2: Let $X = (L_X,V_X,\nu_X,W_X,\omega_X,F_X,G_X,\Sigma,\mathscr{A}_X,\mathscr{P}_X,\mathscr{S}_X)$ and $Y = (L_Y,V_Y,\nu_Y,W_Y,\omega_Y,F_Y,G_Y,\Sigma,\mathscr{A}_Y,\mathscr{P}_Y,\mathscr{S}_Y)$ be two CPDPs such that $V_X \cap V_Y = W_X \cap W_Y = \emptyset$. Then $X|^P_A|Y$ is defined as the CPDP $(L,V,\nu,W,\omega,F,G,\Sigma,\mathscr{A},\mathscr{P},\mathscr{S})$, where

- $L = \{l_1|^P_A|l_2 \mid l_1 \in L_X, l_2 \in L_Y\}$,
- $V = V_X \cup V_Y$, $W = W_X \cup W_Y$,
- $\nu(l_1|^P_A|l_2) = \nu(l_1) \cup \nu(l_2)$, $\omega(l_1|^P_A|l_2) = \omega(l_1) \cup \omega(l_2)$,
- $F(l_1|^P_A|l_2,v)$ equals $F_X(l_1,v)$ if $v \in \nu_X(l_1)$ and equals $F_Y(l_2,v)$ if $v \in \nu_Y(l_2)$.
- $G(l_1|^P_A|l_2,w)$ equals $G_X(l_1,w)$ if $w \in \omega_X(l_1)$ and equals $G_Y(l_2,w)$ if $w \in \omega_Y(l_2)$.
- $\mathscr{A}$, $\mathscr{P}$ and $\mathscr{S}$ are the least relations satisfying the rules r1,r2,r2',r3,r3',r4,r4',r5,r6,r6',r7 and r7', defined below

$$r1.\frac{l_1 \xrightarrow{a,G_1,R_1} l_1', l_2 \xrightarrow{a,G_2,R_2} l_2'}{l_1|^P_A|l_2 \xrightarrow{a,G_1\times G_2,R_1\times R_2} l_1'|^P_A|l_2'}(a \in A).$$

$$r2.\frac{l_1 \xrightarrow{a,G_1,R_1} l_1', l_2 \xrightarrow{\bar{a},R_2} l_2'}{l_1|^P_A|l_2 \xrightarrow{a,G_1\times vs(l_2),R_1\times R_2} l_1'|^P_A|l_2'}(a \notin A).$$

$$r2'.\frac{l_1 \xrightarrow{\bar{a},R_1} l_1', l_2 \xrightarrow{a,G_2,R_2} l_2'}{l_1|^P_A|l_2 \xrightarrow{a,vs(l_1)\times G_2,R_1\times R_2} l_1'|^P_A|l_2'}(a \notin A).$$

$$r3.\frac{l_1 \xrightarrow{a,G_1,R_1} l_1', l_2 \not\xrightarrow{\bar{a}}}{l_1|^P_A|l_2 \xrightarrow{a,G_1\times vs(l_2),R_1\times Id} l_1'|^P_A|l_2}(a \notin A).$$

$$r3'.\frac{l_1 \not\xrightarrow{\bar{a}}, l_2 \xrightarrow{a,G_2,R_2} l_2'}{l_1|^P_A|l_2 \xrightarrow{a,vs(l_1)\times G_2,Id\times R_2} l_1|^P_A|l_2'}(a \notin A).$$

$$r4.\frac{l_1 \xrightarrow{\bar{a},R_1} l_1'}{l_1|^P_A|l_2 \xrightarrow{\bar{a},R_1\times Id} l_1'|^P_A|l_2}(\bar{a} \notin P),$$

$$r4'.\frac{l_2 \xrightarrow{\bar{a},R_2} l_2'}{l_1|^P_A|l_2 \xrightarrow{\bar{a},Id\times R_2} l_1|^P_A|l_2'}(\bar{a} \notin P)$$

$$r5.\frac{l_1 \xrightarrow{\bar{a},R_1} l_1', l_2 \xrightarrow{\bar{a},R_2} l_2'}{l_1|^P_A|l_2 \xrightarrow{\bar{a},R_1\times R_2} l_1'|^P_A|l_2'}(\bar{a} \in P).$$

$$r6.\frac{l_1 \xrightarrow{\bar{a},R_1} l_1', l_2 \not\xrightarrow{\bar{a}}}{l_1|^P_A|l_2 \xrightarrow{\bar{a},R_1\times Id} l_1'|^P_A|l_2}(\bar{a} \in P),$$

$$r6'.\frac{l_1 \not\xrightarrow{\bar{a}}, l_2 \xrightarrow{\bar{a},R_2} l_2'}{l_1|^P_A|l_2 \xrightarrow{\bar{a},Id\times R_2} l_1|^P_A|l_2'}(\bar{a} \in P)$$

$$r7.\frac{l_1 \xrightarrow{\lambda_1,R_1} l_1'}{l_1|^P_A|l_2 \xrightarrow{\hat{\lambda}_1,R_1\times Id} l_1'|^P_A|l_2}, \quad r7'.\frac{l_2 \xrightarrow{\lambda_2,R_2} l_2'}{l_1|^P_A|l_2 \xrightarrow{\hat{\lambda}_2,Id\times R_2} l_1|^P_A|l_2'},$$

where $\hat{\lambda}_1$ and $\hat{\lambda}_2$ are defined as $\hat{\lambda}_1(\xi_1,\xi_2) := \lambda_1(\xi_1)$ and $\hat{\lambda}_2(\xi_1,\xi_2) := \lambda_2(\xi_2)$.

Interaction between CPDPs is expressed through synchronization of active and passive transitions. The conditions under which transitions synchronize are determined by the

composition rules r1 till r7. A is the set of active synchronization actions. Rule r1 expresses that for an action $a \in A$, a component can execute an a-transition if and only if at the same moment the other component executes an a-transition. Rules r2 and r2' express that if one component executes an a-transition and the other component has a $\bar{a}$-transition enabled at that time, then a $\bar{a}$-transition will synchronize with the a-transition. Thus, this expresses how active transitions trigger passive transitions in other components. Rules $r3$ and $r3'$ express that active transitions can still be executed when the other component has no matching passive transitions enabled. This expresses that active a-transitions (with $a \notin A$) are independent of transitions in other components. P is the set of passive synchronization actions. Rules r4, r4', r5, r6 and r6' concern situations where more than one component has passive transitions. For value-passing and for our ATM example, this situation is not relevant and we refer to [7] for an explanation of these rules. Rules r7 and r7' express that spontaneous transitions of the components are present in the composite CPDP. Spontaneous transitions are independent, i.e., they do not synchronize.

III. The value passing CPDP model

The class of CPDPs is closed under composition (this is proven in [7]). We now give an extension of the CPDP model called *value passing*. With value passing, CPDPs can communicate information about the continuous variables. Technically, value passing is 'coded' into the active transitions. This means that the definition of value passing CPDPs is the same as the definition of CPDPs, except for the active transitions, which get a richer structure. After the definition of value passing CPDPs, we define how value passing CPDPs can be composed such that the result of composition is again a value passing CPDP. After that we briefly explain the value passing composition rules. We illustrate the use of value passing for CPDPs in the next section.

Definition 3.1: A value-passing CPDP is a tuple $(L,V,W,\nu,\omega,F,G,\Sigma,\mathscr{A},\mathscr{P},\mathscr{S})$, where all elements except $\mathscr{A}$ are defined as in Definition 2.1 and where $\mathscr{A}$ is a finite set of active transitions that consists of six-tuples (l,a,l',G,R,vp), denoting a transition from location $l \in L$ to location $l' \in L$ with communication label $a \in \Sigma$, guard G, reset map R and value-passing element vp. G is a subset of the valuation space of l. vp can be equal to either $!Y$, $?U$ or $\emptyset$. For the case $!Y$, Y is an ordered tuple $(w_1,w_2,\cdots,w_m)$ where $w_i \in w(l)$ for $i = 1 \cdots m$, For the case $?U$, we have $U \subset \mathbb{R}^n$ for some $n \in \mathbb{N}$ The reset map R assigns to each point in $G \times U$ (for the case $vp = ?U$) or to each point in G (for the cases $vp = !Y$ and $vp = \emptyset$) for each state variable $v \in \nu(l')$ a probability measure on $\mathbb{R}^{d(v)}$. Active transitions α with $\omega(oloc(\alpha)) = \emptyset$, i.e., whose origin locations have no continuous variables, have value passing element $vp = \emptyset$.

Definition 3.2: Let $X = (L_X,V_X,\nu_X,W_X,\omega_X,F_X,G_X,\Sigma,\mathscr{A}_X,\mathscr{P}_X,\mathscr{S}_X)$ and $Y = (L_Y,V_Y,\nu_Y,W_Y,\omega_Y,F_Y,G_Y,\Sigma,\mathscr{A}_Y,\mathscr{P}_Y,\mathscr{S}_Y)$ be two value passing CPDPs such that $V_X \cap V_Y = W_X \cap W_Y = \emptyset$. Then $X|_A^P|Y$ is defined as the CPDP $(L,V,\nu,W,\omega,F,G,\Sigma,\mathscr{A},\mathscr{P},\mathscr{S})$, where L, V, ν, W, ω, F, G, Σ, $\mathscr{P}$ and $\mathscr{S}$ are defined as in Definition 2.2 and $\mathscr{A}$ is the least relation satisfying the rules $r1$, $r2$, $r2'$, $r3$, $r3'$ from Definition 2.2 and the rules $r1data$, $r2data$ and $r2data'$ defined below.

$$r1data.\frac{l_1 \xrightarrow{a,G_1,R_1,v_1} l_1', l_2 \xrightarrow{a,G_2,R_2,v_2} l_2'}{l_1|_A^P|l_2 \xrightarrow{a,G_1|G_2,R_1\times R_2,v_1|v_2} l_1'|_A^P|l_2'}(a \in A, v_1|v_2 \neq \perp).$$

$$r2data.\frac{l_1 \xrightarrow{a,G_1,R_1,v_1} l_1'}{l_1|_A^P|l_2 \xrightarrow{a,G_1\times val(l_2),R_1\times Id,v_1} l_1'|_A^P|l_2}(a \notin A).$$

$$r2data'.\frac{l_2 \xrightarrow{a,G_2,R_2,v_2} l_2'}{l_1|_A^P|l_2 \xrightarrow{a,val(l_1)\times G_2,Id\times R_2,v_2} l_1|_A^P|l_2'}(a \notin A),$$

where $l_1 \xrightarrow{a,G_1,R_1,v_1} l_1'$ means $(l_1,a,l_1',G_1,R_1,v_1) \in \mathscr{A}_X$ with $v_1 \neq \emptyset$, and $l_1 \xrightarrow{a,G_1,R_1} l_1'$ means $(l_1,a,l_1',G_1,R_1,\emptyset)$ and $v_1|v_2$ is defined as:

- v1: $v_1|v_2 := !Y$ if $v_1 = !Y$ and $v_2 := ?U$ and $\dim(U)=\dim(Y)$ or if $v_2 = !Y$ and $v_1 := ?U$ and $\dim(U)=\dim(Y)$,
- v2: $v_1|v_2 := ?(U_1 \cap U_2)$ if $v_1 = ?U_1$ and $v_2 = ?U_2$ and $\dim(U_1)=\dim(U_2)$,
- v3: $v_1|v_2 := \perp$ otherwise, where $\perp$ means that v_1 and v_2 are not compatible.

Furthermore, $G_1|G_2$ is, only when $v_1|v_2 \neq \perp$, defined as:

- g1: $G_1|G_2 := (G_1 \cap U) \times G_2$ if $v_1 = !Y$ and $v_2 = ?U$,
- g2: $G_1|G_2 := G_1 \times (G_2 \cap U)$ if $v_1 = ?U$ and $v_2 = !Y$,
- g3: $G_1|G_2 := G_1 \times G_2$ if $v_1 = ?U_1$ and $v_2 = ?U_2$.

Here we define $G \cap U$ as the set of all states in G whose output values lie in U.

Rules $r2data$ and $r2data'$ express that if $a \notin A$, then no synchronization happens and the transitions interleave. Value passing is expressed through rule $r1data$. Case $v1$ of $r1data$ expresses that if one component has an a-transition with output $!Y$ and the other component has an a-transition with input $?U$, then they synchronize, which expresses that the output value is passed to the input transition. The input transition can 'use' the output value in its reset map, because the reset map chooses different probability measures for different input values (see Definition 3.1). Case $v2$ expresses that two input transitions with the same dimension, i.e., which can both receive values of the same dimension, synchronize. This expresses that an output transition can pass its value to multiple input transitions (one in each other component, compare with rule r5). Case $v3$ expresses that when two transitions are not compatible, i.e., have different dimensions, then they will not synchronize. This also expresses that a-transitions (with $a \in A$), which do not have a matching partner in the other component (with the same dimension), are blocked.

Value passing is partly 'coded' into the guards of the resulting synchronized transition. Cases $g1$ and $g2$ express that the guard of the output transition is restricted by U of the

input transition. This expresses that if some $u \notin U$, i.e., when the input transition cannot receive value u, then the output transition is not allowed to be executed at states with output u. Case $g3$ expresses that the guard of a synchronized input transition is the intersection of the guards of the individual input transitions. This means that in a composition context with three or more components, an output a-transition can pass value u only when all input a-transitions in the other components allow value u, i.e., have $u \in U$.

IV. Air Traffic Management example

We give an example of an Air Traffic Management system modelled as a composition of value passing CPDPs. This system is a part of the larger system 'free flight', which was originally modelled as a Dynamically Colored Petri Net [3]. The system describes a pilot in an aircraft (the so called 'pilot flying') who has to execute different tasks. There is an audio alert system that can signal to the pilot which tasks need to be done. The system is then decomposed into two parts: 1. Pilot flying: models the pilot executing different tasks during the flight. 2. Audio Alert: models the communication device that communicates to the pilot which tasks need to be executed at which times.

We decompose 'pilot flying' into three subsystems: Current Goal, Task Performance and Memory. These three systems do not correspond to actual systems, but form a way to systematically model the behavior of the pilot. Task Performance is the part of 'pilot flying' that is actually executing a task. This system contains the information and structure needed to execute the different tasks. Current Goal is a control unit. It keeps track of which task is executed and which tasks need to be executed in the future. It forms the interface between Audio Alert, Memory and Task Performance in the following sense: Audio Alert communicates to Current Goal that a task needs to be executed. Then, either Current Goal communicates to Task Performance to execute the task, or, if the pilot is still busy executing a task, stores the task into Memory such that when the pilot is not busy anymore, it can retrieve this information from the Memory and execute the task. Thus, Current Goal models in some sense the awareness of the pilot what he is doing and what he needs to do in the future. Memory models the memory (or a memory aid) of the pilot. We now describe the system and the communication within it, in more detail.

During the flight, there are seven distinguished tasks that might be executed by the pilot, which are as follows. C1: Collision avoidance, C2: Emergency actions, C3: Conflict resolution,C4: Navigation vertical, C5: Navigation horizontal, C6: Preparation route change and C7: Miscellaneous. Task C2, emergency actions, is split up into six distinct tasks. Each C2-task corresponds to a specific kind of emergency action. Task C2.1 is executed in case of an engine failure, task C2.2 is executed in case of a navigation-system failure. Tasks C2.3 till C2.5 correspond to failures of other aircraft systems. Task C2.6 is titled 'other emergency'.

The ordering is an ordering of priority. This means that C1, collision avoidance, has higher priority than C2, emergency action, which has higher priority than C3, conflict resolution, etc. If the pilot is executing task C2 and Audio Alert communicates that task C1 needs to be done, then, because C1 has higher priority, finishing task C2 needs to be postponed and C1 is executed immediately. In this case 'executing task C2' should be put into Memory, such that the pilot 'knows' after executing task C1 that he still has to execute task C2. Vice versa, if the pilot is executing task C1 and Audio Alert communicates that task C2 needs to be executed, then the pilot should put 'execute C2' into memory and continue executing task C1. There is no ordering for the tasks C2.1 till C2.6. If the pilot starts executing task C2 while there are multiple failures, i.e. multiple tasks of the set of tasks C2 need to be executed, then a random choice is made about which C2 task is executed.

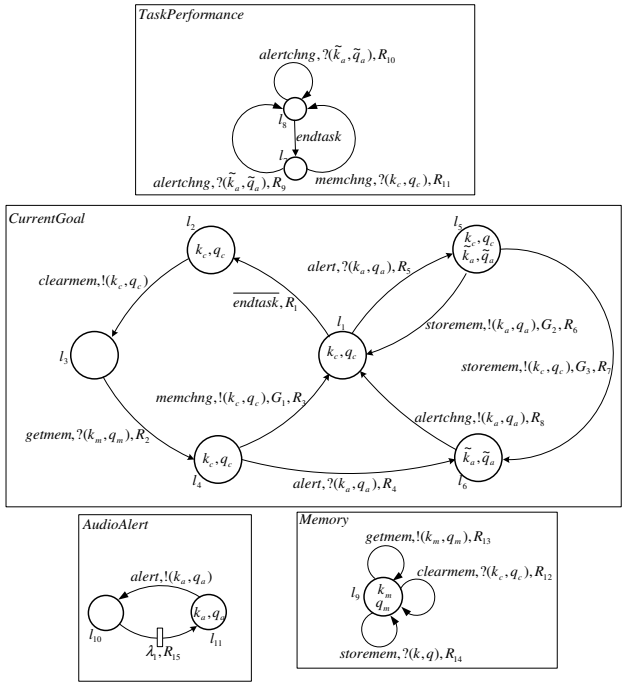

Fig. 1. CPDP 'pilot flying' model

We will model this system in detail as far as it concerns the control unit Current Goal and its Memory. The number of details of the Task Performance system is large and the interactions happening within Task Performance are complex. We do not model these details and we use a very simple unrealistic model for the Task Performance. It is our goal to show that interactions between these subsystems can be modelled through composition of value passing CPDPs and in order to that we do not need to model all the details of the ATM system. Also, the Audio Alert system will be modelled in a simple unrealistic way, where we assume that the time that a task needs to be executed is exponentially distributed for each task.

The total system will be composed as

$$((CurrentGoal|^P_{A_1}|TaskPerformance)|^P_{A_2}|Memory)|^P_{A_3}|AudioAlert,$$

where $A_1 = \{alertchng, memchng\}$, $A_2 = \{getmem, clearmem, storemem\}$ and $A_3 = \{alert\}$. We now describe all CPDP components.

1) CPDP AudioAlert: Location l_{10} of *AudioAlert* models the situation where an alert signal might be 'generated'. l_{10} is an empty location, i.e. there are no continuous dynamics at this location. Constant λ is the parameter of the exponential distributed time indicating the time of an alert. If an alert signal is generated at time t, meaning that a task needs to be executed at time t, then *AudioAlert* switches to location l_{11}. Reset map R_{15} resets the state variables k_a and q_a of location l_{11}. The value of k_a and q_a denote the task that needs to be executed. $k_a = i$ corresponds to task Ci and in case $k_a = 2$, $q_a = j$ corresponds to task C2.j. If $k_a \neq 2$, then q_a is irrelevant and equals zero.

Let the rate of occurrence that task C needs to be executed be equal to λ_C. Then we get

$$\lambda = \sum_{C \in Tasks} \lambda_C,$$

where $Tasks = \{C1, C2.1, C2.2, C2.3, C2.4, C2.5, C2.6, C3, C4, C5, C6\}$ and

$$R_{15} = \sum_{C \in Tasks} \frac{\lambda_C}{\lambda} R_C,$$

where R_{C1} resets $k_a := 1$ and $q_a := 0$, $R_{C2.1}$ resets $k_a := 2$ and $q_a := 1$, $R_{C2.2}$ resets $k_a := 2$ and $q_a := 2$, R_{C3} resets $k_a = 3$ and $q_a = 0$, etc. The *alert*-transition from l_{11} to l_{10} is executed immediately after the spontaneous transition, because there is no guard. (Technically, the guard equals the whole state space of location l_{11}). In this ATM example we do not distinguish between state and output variables. Formally, we model this by having for each state variable x a copy output variable y_x whose value equals the state value everywhere. The value passing part $!(k_a, q_a)$ expresses that the values of variables k_a and q_a are value-passed in a synchronization with component *CurrentGoal*, as we will see later. (Formally only output variables can be value-passed and the value passing part should therefore formally be equal to $!(y_{k_a}, y_{q_a})$.)

We see that location l_{11} is an intermediate location where no time is consumed, which only serves the value passing of the alert signal (k_a, q_a) via channel *alert*.

2) CPDP TaskPerformance: The empty location l_7 of CPDP *TaskPerformance* denotes the situation where the pilot is not executing a task and is waiting for a new task to be executed. If a new task needs to be executed, *TaskPerformance* switches to location l_8 which denotes the situation where a task is executed. The dynamics of 'executing a task' is large and complex in the 'pilot flying' system. We do not model this complexity and model it as one location, l_8. We unrealistically assume that the dynamics of the execution of a task is expressed by a differential equation $\dot{x} = f(x)$, where each value for x denotes a state somewhere in the execution of some task. We assume that for each task $C \in Tasks$ a state x_C exists such that evolution from x_C denotes starting and evolution of task C. We also assume that if x_C enters guard area G_4, then the task of execution is completed. If a task is completed, *TaskPerformance* switches to l_7 via the active *endtask*-transition with guard G_4. The *endtask* signal will be received by CPDP *CurrentGoal*.

There are two ways in which a task execution is started. First, if CPDP *CurrentGoal* executes a *alertchng*-transition with value passing $!(k_a, q_a)$. Execution of this transition by *CurrentGoal* denotes the situation where, as we will see later in detail, *CurrentGoal* has received a (k_a, q_a) signal from *AudioAlert*. This value (k_a, q_a) is received by the $alertchng, ?(k_a, q_a)$-transition from location l_7 of *TaskPerformance*. Then, the reset map R_9 resets state x of l_8 to value x_C, where C is the task that corresponds to the passed value (k_a, q_a). Note that the $alert, ?(k_a, q_a)$ transition is formally specified as $alert, ?U$ with $U = \{(1,0), (2,1), (2,2), (2,3), (2,4), (2,5), (2,6), (3,0), (4,0), (5,0), (6,0), (7,0)\}$, the set of all values corresponding to all tasks. If the pilot is execution a task, thus, if *TaskPerformance* is in location l_8, and a task with a higher priority needs to be executed, then this switching of tasks is expressed by the *alertchng*-transition from l_8 to itself. Reset map R_{10} is equal to R_9.

The second situation where a task execution is started, is the one where *CurrentGoal* executes a *memchng*-transition with value passing $!(k_c, q_c)$. This expresses the situation where after the completion of a task a new to-be-executed-task is retrieved from the memory by *CurrentGoal* and stored in variables k_c and q_c, after which the $memchng, !(k_c, q_c)$-transition is executed. The value (k_c, q_c) is received by the $memchng, ?(k_c, q_c)$-transition from location l_7 of *TaskPerformance*. Reset map R_{11} resets x to x_C, with C the task corresponding to received value (k_c, q_c).

3) CPDP Memory: CPDP *Memory* has one location l_9 with state variables k_m and q_m. There is no continuous dynamics, i.e. $\dot{k}_m = \dot{q}_m = 0$. k_m has seven components, i.e. $k_m = (k_{m,1}, k_{m,2}, \cdots, k_{m,7})$ and takes value in $\mathbb{R}^7$. q_m has six components, i.e. $q_m = (q_{m,1}, q_{m,2}, \cdots, q_{m,6})$ and takes value in $\mathbb{R}^6$. If at some time $k_{m,i} = 1$, then this means that task Ci needs to be executed because, as we will see, it is the consequence of *CurrentGoal* putting the value $k_{m,i} := 1$ to place task Ci on the stack. Similarly, $q_{m,i} = 1$ means that task C2.i is put on the stack. If a task Ci is not on stack, then $k_{m,i}$ equals zero. For example $k_m = (0,1,1,0,0,0,0)$ and $q_m = (1,1,0,0,0,0)$ means that tasks C2.1, C2.2 and C3 are put on stack. This situation can happen in the unlucky situation where the pilot is executing task C1, collision avoidance, which has highest priority, while Audio Alert communicates that the engine and navigation systems switched to failure mode.

There are three transitions corresponding to three memory actions. The three memory actions are: retrieving the memory state, storing a new to-be-executed-task in the memory and clearing a to-be-executed-task from the memory as soon as this task has been completed by the pilot.

Retrieving the memory state is done via the $getmem, !(k_m, q_m)$ transition. The memory value (k_m, q_m) is then passed to CPDP *CurrentGoal*. Reset map R_{13} is the identity reset map, i.e. it does not change the state of the

memory.

Storing a new task in memory is done via the $storemem, ?(k,q)$ transition. The value (k,q) is passed by *CurrentGoal* to *Memory* via this transition. Reset map R_{14} does not change the memory state except that for $i = k$ and $j = q$, $k_{m,i}$ and $q_{m,j}$ are reset to one.

Removing a task from the memory is done via the $clearmem, ?(k,q)$ transition. The value (k,q) is passed by *CurrentGoal* to *Memory* via this transition, indicating that the corresponding task is completed. Reset map R_{12} does not change the memory state except that for $i = k$ and $j = q$, $k_{m,i}$ and $q_{m,j}$ are reset to zero.

4) CPDP CurrentGoal: During the flight situation where the pilot is not working on a task and there are no tasks in memory, CPDP *CurrentGoal* is in location l_4 with $k_c = q_c = 0$. $k_c = 0$ indicates here that no task needs to be executed. If $k_c \neq 0$ in l_4, then, as we will see later, this would indicate that a task needs to be executed and then the *memchng*-transition to l_1 would be taken. Suppose that $k_c = q_c = 0$ in l_4. If an *alert* signal is executed by *AudioAlert*, then *CurrentGoal* switches to l_6 with value passing transition $alert, ?(k_a, q_a)$. Reset map R_4 copies the input values of the transition (k_a, q_a) to the variables $\tilde{k}_a$ and $\tilde{q}_a$. Thus, in l_4 $\tilde{k}_a$ and $\tilde{q}_a$ correspond to the task that needs to be executed according to *AudioAlert*. The guardless transition $alertchng, !(\tilde{k}_a, \tilde{q}_a)$ to l_1 is taken immediately, inducing the $alertchng, ?(\tilde{k}_a, \tilde{q}_a)$-transition in *TaskPerformance*, which means that the task corresponding to $(\tilde{k}_a, \tilde{q}_a)$ will be executed. Reset map R_5 puts the values of $\tilde{k}_a, \tilde{q}_a$ into the variables k_c and q_c at location l_1. Thus, at l_1, k_c and q_c correspond to the task that is currently worked on.

At l_1, two things can happen: 1. An *alert*-signal from *AudioAlert* is received, 2. An *endtask*-signal from *TaskPerformance* is received.

In case 1, *CurrentGoal* executes the $alert, ?(k_a, q_a)$ transition and switches to location l_5. Reset map R_3 copies k_c and q_c at l_1 to k_c and q_c at l_5 and R_3 copies inputs k_a and q_a to $\tilde{k}_a$ and $\tilde{q}_a$ at l_5. At l_3 the current task (k_c, q_c) and the requested task $(\tilde{k}_a, \tilde{q}_a)$ need to be compared in order to decide which task has the highest priority. This 'comparing' is coded in the guards G_2 and G_3. G_2 contains all states of l_5 where task (k_c, q_c) has higher or equal priority, i.e.

$$G_2 = \{(k_c, q_c, \tilde{k}_a, \tilde{q}_a) | (k_c, q_c) > (\tilde{k}_a, \tilde{q}_a)\},$$

where $(k_c, q_c) > (\tilde{k}_a, \tilde{q}_a)$ means that task (k_c, q_c) has higher or equal priority than task $(\tilde{k}_a, \tilde{q}_a)$. Equal priority only happens when both tasks are tasks of $C2$. (The tasks of C2 have no ordering). G_3 is the complement of G_2, i.e. G_3 contains all states where task (k_c, q_c) has lower priority. If at l_5, G_2 is satisfied, then the $storemem, !(\tilde{k}_a, \tilde{q}_a)$ transition to l_1 is executed, where R_6 copies k_c and q_c at l_5 to k_c and q_c at l_1. This transition induces the *storemem* transition of *Memory*. The cycle $l_1 \to l_5 \to l_1$, means that the requested task of *AudioAlert* is stored into memory and the task that is currently worked on at l_1 is not changed. If at l_5, G_3 is satisfied, then the $storemem, !(k_c, q_c)$ transition to l_6 is executed. This means that the task that was worked on at l_1 is now stored into memory. Reset map R_7 copies $(\tilde{k}_a, \tilde{q}_a)$ at l_5 to $(\tilde{k}_a, \tilde{q}_a)$ at l_6. At l_6 the requested task $(\tilde{k}_a, \tilde{q}_a)$ needs to be executed, and this happens via the *alertchng*-transition to l_1.

In case 2 at location l_1, *CurrentGoal* waits until the task that is currently worked on is completed. After completion, *TaskPerformance* sends the *endtask* signal, which induced the $\overline{endtask}$ transition of *CurrentGoal* to l_2. Reset map R_1 copies (k_c, q_c) at l_1 to (k_c, q_c) at l_2. Thus, at l_2 the state (k_c, q_c) corresponds to the task that has just been completed. This means that this task needs to be removed from the memory. This happens via the $clearmem, !(k_c, q_c)$ transition to l_3, which induced the *clearmem* transition of *Memory* which removes task (k_c, q_c) from the memory. At l_3, the pilot needs to check the memory, whether there is a to-be-executed-task stored in the memory. The $getmem, ?(k_m, q_m), R_2$ transition to l_4 checks the memory and stores the new to-be-executed-task, or $(0,0)$ in case there is no new to-be-executed-task, in (k_c, q_c) at l_4. Via this value passing transition, the memory state is passed to *CurrentGoal*. Reset map R_2 should then be defined as:

1) Let $i = \min(\{r | k_{m,r} = 1\} \cup \{0\})$.
2) If $i \neq 2$, then $j := 0$, otherwise, take j randomly from the set $\{r | q_{m,r} = 1\}$.
3) Reset $k_c := i$ and $q_c := j$ at l_4.

Note that R_2 resets $k_c = q_c = 0$ if there are no tasks in the memory, otherwise R_2 takes the task with the highest priority. At l_4 the task from the memory is executed via the *memchng* transition to l_1, or, in case $k_c = q_c = 0$, the pilot 'waits' at l_4 for an alert signal.

V. Conclusions

In this paper we introduced value-passing for CPDPs, which can be used to express the communication of continuous data between components. In the future, we hope to develop model-checking tools for CPDPs, such that system properties can be automatically verified.

References

[1] M. H. A. Davis, "Piecewise Deterministic Markov Processes: a general class of non-diffusion stochastic models," *Journal Royal Statistical Soc. (B)*, vol. 46, pp. 353–388, 1984.

[2] ——, *Markov Models and Optimization*. London: Chapman & Hall, 1993.

[3] M. H. C Everdij and H. A. P. Blom, "Petri-nets and hybrid-state Markov processes in a power-hierarchy of dependability models" in *Preprints Conference on Analysis and Design of Hybrid Systems ADHS 03*, 2003, pp. 355–360.

[4] S. N. Strubbe, A. A. Julius, and A. J. van der Schaft, "Communicating Piecewise Deterministic Markov Processes," in *Preprints Conference on Analysis and Design of Hybrid Systems ADHS 03*, 2003, pp. 349–354.

[5] S. N. Strubbe and R. Langerak, "A composition operator for complex control systems," 2005, Accepted for 25th IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems, Taipei.

[6] S. N. Strubbe and A. J. van der Schaft, "Stochastic semantics for Communicating Piecewise Deterministic Markov Processes," Accepted for Conf. Decision and Control, Seville, 2005.

[7] S. N. Strubbe, "Compositional modelling of stochastic hybrid systems," Twente University, Phd. Thesis, 2005.