Proceedings of the International Workshop, Automata for Cellular and Molecular Computing, MTA SZTAKI, Budapest, pages 76 - 89, 2007.

Principles of Transforming Communicating X-Machines to Population P Systems

Petros Kefalas[1], Ioanna Stamatopoulou[2], and Marian Gheorghe[3]

[1]CITY College, Dept. of Computer Science
Tsimiski 13, Thessaloniki 54624, Greece
kefalas@city.academic.gr

[2]South-East European Research Centre
Mitropoleos 17, Thessaloniki 54624, Greece
istamatopoulou@seerc.org

[3]University of Sheffield, Dept. of Computer Science
Regent Court, 211 Portobello Str., Sheffield S1 4DP, UK
m.gheorghe@dcs.shef.ac.uk

Abstract

Population P Systems is a class of P Systems in which cells are arranged in a graph rather than a hierarchical structure. On the other hand, Communicating X-machines are state-based machines, extended with a memory structure and transition functions instead of simple inputs, which communicate via message passing. One could use Communicating X-machines to create system models built out of components in a rather intuitive way. It is worth investigating how existing Communication X-machine models can be transformed to Population P system models so that we could take advantage of the dynamic features of the latter. In this paper, we attempt to define the principles of transforming Communicating X-machines to Population P Systems. We describe the rules that govern such transformation and we present an example in order to demonstrate the feasibility of the transformation and discuss its advantages and shortcomings.

1 Introduction

In the last years, attempts have been made to devise computational models in the form of generative devices, such as P systems [13] and its variants. These new computational paradigms have been used to solve well-known hard problems. Occasionally, some attempts also have been made to use P Systems towards modelling of swarm-based multi-agent systems [14], in order to take advantage of the reconfiguration features of P systems, such as cell death, cell division, reconfiguration of structure etc. The main problem which appears in such modelling activity is that the model resulting for the object interaction within a cell is not always easy to develop. On the other hand, state-based models provide the necessary "intuitiveness"

to model the behaviour of system components or agents. For instance, communicating X-machines have been used as a suitable paradigm of modelling agent based specification [11].

As a natural consequence of the above complementary features is to either try to combine both formalisms [17, 18] or to transform one formalism to another. The current trend in P system community research shows more interest in connecting this model with other computational approaches - Petri nets [12], process algebra [3], cellular automata [4] etc. In the past relationships between some classes of P systems and communicating X-machines have been investigated. Especially transformations of P systems into communicating stream X-machines have been particularly considered [9]. Most of these studies have been interested in translations between these models in order to make use of various strengths offered by different formalisms - model checking, for process algebra, invariants, for Petri nets, or testing methods, for X-machines.

This paper presents some principles for transforming Communicating X-machines to Population P Systems. Section 2 provides the basic background on X-machine modelling and Communicating X-Machines accompanied by an example. The definition and advantages of Population P Systems are presented in Section 3. In Section 4, we demonstrate how a transformation from one model to another is feasible and apply the guidelines to the particular example. We then discuss how the resulting model could be enhanced further to take advantage of the dynamic features of Population P Systems. We conclude by discussing the ideas behind the transformation and the issues that need further consideration.

2 State-Based Modelling with X-Machines

2.1 X-machines

X-machines (XM), a state-based formal method introduced by Eilenberg [5], are considered suitable for the formal specification of a system's components. Stream X-machines, in particular, were found to be well-suited for the modelling of reactive systems. Since then, valuable findings using the X-machines as a formal notation for specification, communication, verification and testing purposes have been reported [6, 7, 10]. An X-machine model consists of a number of states and also has a memory, which accommodates mathematically defined data structures. The transitions between states are labelled by functions. More formally, a steam X-machine is defined as the 8-tuple $(\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ where:

- Σ and Γ are the input and output alphabets respectively;
- Q is the finite set of states;
- M is the (possibly) infinite set called memory;
- Φ is a set of partial functions φ that map an input and a memory state to an output and a possibly different memory state, $\varphi : \Sigma \times M \rightarrow \Gamma \times M$;

- F is the next state partial function, $F : Q \times \Phi \rightarrow Q$, which given a state and a function from the type Φ determines the next state. F is often described as a state transition diagram;
- q_0 and m_0 are the initial state and initial memory respectively.

For the purposes of this work we consider that the memory M is of the form $M = (m_1, \ldots, m_n)$, where each m_i is a label that refers to any arbitrary value from a domain set D_i.

2.2 Example of XM

Assume a system that consists of two XM models, i.e. one traffic light and one car. The traffic light XM has a number of states $Q = \{green, yellow, red, off\}$ and a set of inputs $\Sigma = \{tick, power\_on, power\_off\}$ representing a clock tick and the availability of electricity respectively. The output $\Gamma = \{green, red, yellow, black\}$ is the colour that the traffic light displays. The memory structure of this XM holds the display duration (in clock ticks) of each colour and a timer that counts down the ticks on each colour display. Therefore, $M = (time\_left\_to\_change, duration\_green, duration\_yellow, duration\_red)$, where $time\_left\_to\_change \in \mathbb{N}_0$, and $duration\_green, duration\_yellow, duration\_red \in \mathbb{N}$. An instance of the above model, e.g. TL_1, may have $m_0 = (20, 20, 3, 10)$ and $q_0 = green$. The state transition diagram F is depicted in Fig. 1. The set Φ consists of a number of functions, as for example:

$keep\_green(tick, (time\_left, dg, dy, dr)) = (green, (time\_left - 1, dg, dy, dr))$,
if $time\_left > 0$

$change\_yellow(tick, (0, dg, dy, dr)) = (yellow, (dy, dg, dy, dr))$

$switch\_off(power\_off, (tl, dg, dy, dr)) = (black, (tl, dg, dy, dr))$

Similarly the car model (Fig. 1) is defined as follows:

$Q = \{stopped, accelerating, cruising, breaking\}$

$M = (speed, decrease\_rate, position)$ with $speed \in \mathbb{N}_0, decrease\_rate \in \mathbb{N}$ and $position \in \{free\_road, approaching\_light(TL)\}$, where TL is the identifier of the specific traffic light the car is approaching.

$\Sigma = \{traffic\_light(TL), passed\_traffic\_light(TL), push\_break\_to\_stop, push\_break, push\_accpedal, leave\_break, leave\_accpedal\}$

and $\Gamma = \mathbb{N}_0$ having each function output the current speed of the car.

An instance of car, e.g. CAR_1, may have $m_0 = (100, 2, free\_road)$ and $q_0 = cruising$.

Indicatively some of the functions in Φ are:

$approaching\_tl(traffic\_light(TL), (speed, decrease\_rate, pos)) =$
$(speed, (speed, decrease\_rate, approaching\_light(TL)))$

$start\_breaking(push\_break, (speed, decrease\_rate, pos)) =$
$(speed/decrease\_rate, (speed/decrease\_rate, decrease\_rate, pos))$

$decrease\_speed(push\_break, (speed, dr, pos)) = (speed/dr, (speed/dr, dr, pos))$

$stop(push\_break\_to\_stop, (speed, dr, pos)) = (0, (0, dr, pos))$

$start(push\_accpedal, (0, dr, pos)) = (10, (10, dr, pos))$

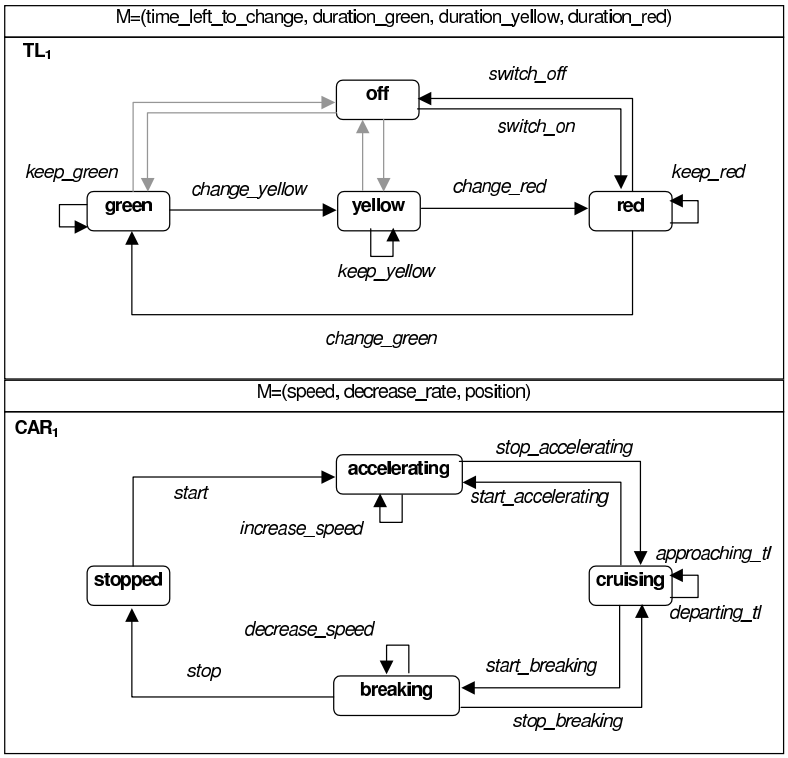

Figure 1: State Transition Diagrams for two XMs: a traffic light and a car.

2.3 Communicating X-machines

In addition to having stand-alone X-Machine models, communication is feasible by redirecting the output of one machine's function to become input to a function of another machine. The system structure of *Communicating X-machines* is defined as the graph whose nodes are the components and edges are the communication channels among them. A *Communicating X-machine System Z* is a tuple:

$$Z = ((C_i^x)_{i=1,\ldots,n},\ R)$$

where:

- C_i^x is the i-th Communicating X-machine Component, and
- R is a relation defining the communication among the components, $R \subseteq C^x \times C^x$ and $C^x = \{C_1^x, \ldots, C_n^x\}$. A tuple $(C_i^x, C_k^x) \in R$ denotes that the X-machine component C_i^x can output a message to a corresponding input stream of X-machine component C_k^x for any $i, k \in \{1, \ldots, n\}$, $i \neq k$.

A Communicating X-machine Component (CXM for short) can be derived by incorporating into an X-machine information about how it is to communicate with other X-machines that participate in the system. Exchange of messages among the components is achieved by redirecting one component's function output to be received as input by a function of another machine. In order to define the communication interface of an X-machine two things have to be stated: (a) which of its

functions receive their inputs from which machines, and (b) which of its functions send their outputs to other machines.

Graphically on the state transition diagram we denote the acceptance of input from another component by a solid circle along with the name C_i^x of the CXM that sends it. Similarly, a solid diamond with the name C_k^x denotes that output is sent to the C_k^x CXM. An abstract example of the communication between two CXMs is depicted in Fig. 2. It has to be noted that though a function φ may only read from one component at a time, it is possible that it sends its output to more than one components. A complete formal definition of Communicating X-Machines can be found in [16].

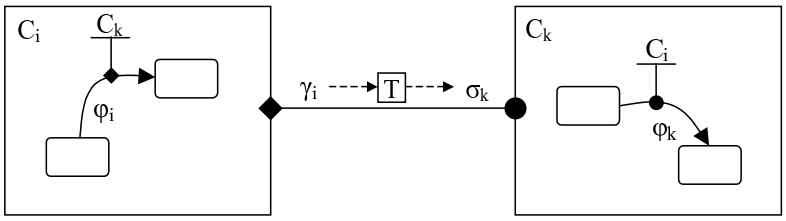

Figure 2: Abstract example of the communication between two Communicating X-machine Components.

2.4 Example of CXM

The two instances TL_1 and CAR_1 may form a communicating system as illustrated in Fig. 3. Functions of TL_1 send messages to CAR_1, through the transformation function T:
$T(change\_yellow) = push\_break$
$T(keep\_yellow) = push\_break$
$T(change\_red) = push\_break\_to\_stop$
$T(change\_green) = push\_accpedal$

Functions in CAR_1 accept those messages as inputs. The rest of the functions not annotated with receive (read) or send (write) obtain their input from the environment and send their output to the environment as normal.

3 Population P-Systems with Active Cells

A *Population P System* (PPS) [2] is a collection of different types of cells evolving according to specific rules and capable of exchanging biological / chemical substances with their neighbouring cells (Fig. 4). More formally, a PPS with active cells [2] is defined as a construct $\mathcal{P} = (V, K, \gamma, \alpha, w_E, C_1, C_2, \ldots, C_n, R)$ where:

- V is a finite alphabet of symbols called objects;
- K is a finite alphabet of symbols, which define different types of cells;

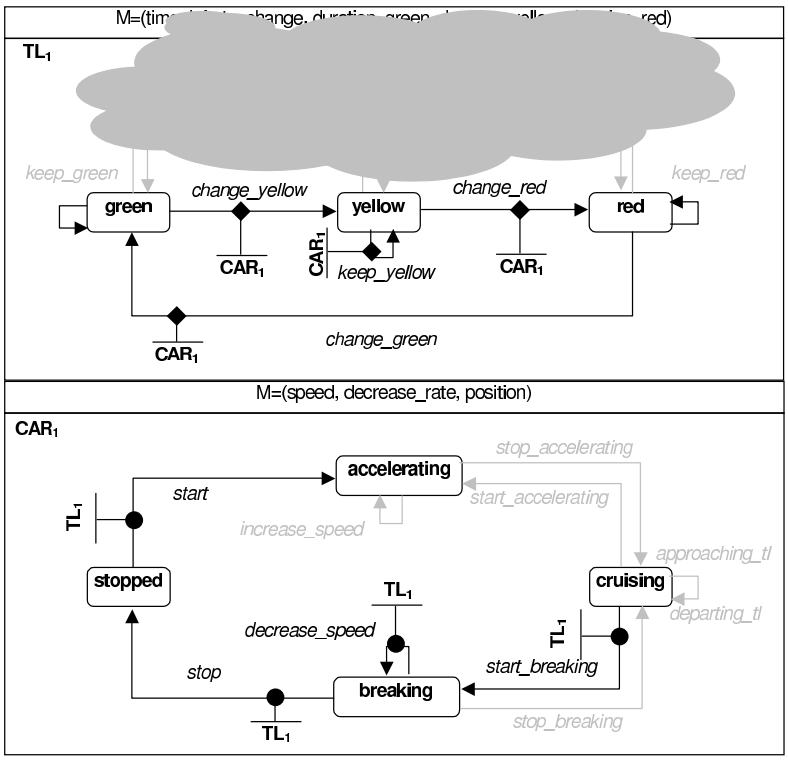

Figure 3: A CXM system consisting of one traffic light and a car.

- $\gamma = (\{1, 2, \ldots n\}, A)$, with $A \subseteq \{\{i, j\} \,|\, 1 \leq i \neq j \leq n\}$, is a finite undirected graph;
- α is a finite set of bond-making rules of the form $(t, x_1; x_2, p)$, with $x_1, x_2 \in V^*$ (multi-sets of objects represented as strings), and $t, p \in K$ meaning that in the presence of x_1 and x_2 inside two cells of type t and p respectively, a bond is created between the two cells;
- $w_E \in V^*$ is a finite multi-set of objects initially assigned to the environment;
- $C_i = (w_i, t_i)$, for each $1 \leq i \leq n$, with $w_i \in V^*$ a finite multi-set of objects, and $t_i \in K$ the type of cell i;
- R is a finite set of rules dealing with communication, object transformation, cell differentiation, cell division and cell death.

All rules present in the PPS are identified by a unique identifier, r. More particularly:

Communication rules are of the form $r : (a; b, in)_t$, $r : (a; b, enter)_t$, $r : (b, exit)_t$, for $a \in V \cup \{\lambda\}$, $b \in V$, $t \in K$, where λ is the empty string, and allow the moving of objects between neighbouring cells or a cell and the environment according to the cell type and the existing bonds among the cells. The first rule means that in the presence of an object a inside a cell of type t an object b can be obtained by

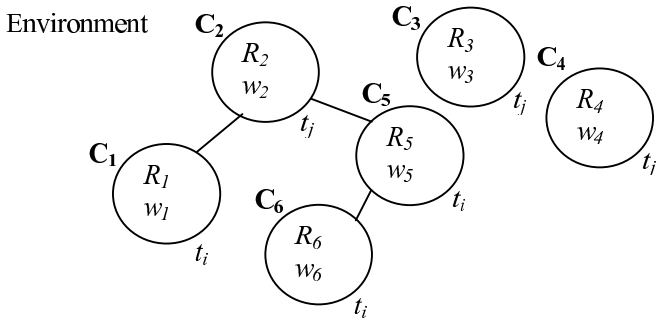

Figure 4: An abstract example of a Population P System; C_i: cells, R_i: sets of rules related to cells; w_i: multi-sets of objects associated to the cells.

a neighbouring cell non-deterministically chosen. The second rule is similar to the first with the exception that object b is not obtained by a neighbouring cell but by the environment. Lastly, the third rule denotes that if object b is present it can be expelled out to the environment.

Transformation rules are of the form $r : (\, a \rightarrow b\,)_t$, for $a \in V$, $b \in V^+$, $t \in K$, where V^+ is the set of non-empty strings over V, meaning that an object a is consumed and replaced by an object b within a cell of type t.

Cell differentiation rules are of the form $r : (\, a\,)_t \rightarrow (\, b\,)_p$, with $a, b \in V$, $t, p \in K$ meaning that consumption of an object a inside a cell of type t changes the cell, making it become of type p. All existing objects remain the same besides a which is replaced by b.

Cell division rules are of the form $r : (\, a\,)_t \rightarrow (\, b\,)_t\, (\, c\,)_t$, with $a, b, c \in V$, $t \in K$. A cell of type t containing an object a is divided into two cells of the same type. One of the new cell has a replaced by b while the other by c. All other objects of the originating cell appear in both new cells.

Cell death rules are of the form $r : (\, a\,)_t \rightarrow \dagger$, with $a \in V$, $t \in K$ meaning that an object a inside a cell of type t causes the removal of the cell from the system.

4 Transformation Principles

The question under investigation is whether some generic guidelines or principles for transforming Communicating X-machines to Population P Systems exist. We are dealing with two different methods that possess different characteristics. CXMs provide a straightforward and rather intuitive way for dealing with a component's behaviour, however, the structure of a communicating system should be known in advance and fixed throughout the computation. Additionally, CXM computation is asynchronous. On the other hand, PPS provide a straightforward way for dealing with the change of a system's structure, however, the rules specifying the behaviour of the individual cells in a PPS are of the simple form of rewrite rules which are not that intuitive to model. Finally, PPS computation is synchronous.

The rationale behind such transformation is to automatically or semi-automatically produce PPS models that can be later on enhanced with dynamic behaviour

features. This will have the advantage of using existing CXM models whose components have been thoroughly verified and tested. The resulting PPS model will have cells, objects, transformation and communication rules. The model can then be enriched with cell differentiation, death, birth and bond making rules (see next section).

4.1 Cells and types

Every CXM component will form a cell with objects, transformation and communication rules. We will refer to these cells as cells of a communicating type.

4.2 Objects in cells

We consider that all objects in the PPS are of the form $(tag : value)$, tag being a description that allows us to identify what each object represents. At least the following objects must be present in a cell to represent:

- the states in Q: objects of the form $(state : q)$, where $q \in Q$
- the memory $M = (m_1, \ldots, m_n)$: objects of the form $(m_1 : d_1) \ldots (m_n : d_n)$ where $d_1, \ldots, d_n \in D_1, \ldots, D_n$, with $D_1, \ldots, D_n$ being the *finite* domains of each memory item
- the inputs in Σ: objects of the form $(input : i)$, where $i \in \Sigma$
- the outputs in Γ: objects of the form $(output : o)$, where $o \in \Gamma$
- the messages sent/received: objects of the form $(message : (m, sender, receiver))$ where m is the actual message being sent.

4.3 Transformation rules

For every function $\varphi : \Sigma \times M \to \Gamma \times M$ such that $\varphi(\sigma, (d_1, ...d_n)) = (\gamma, (d'_1, ...d'_n))$, where $d_i, d'_i \in D_i, \sigma \in \Sigma, \gamma \in \Gamma$, for every $q, q' \in Q$ such that $q' \in F(\varphi, q)$ and for $(m_i : d_i), (m_i : d'_i)$ representing old and new values of the memory used by φ, a rule

$$\begin{aligned} \varphi \quad : \quad & ((state : q)\ (input : \sigma)\ (m_1 : d_1) \ldots (m_n : d_n) \\ \to \quad & (state : q')\ (output : \gamma)\ (m_1 : d'_1) \ldots (m_n : d'_n))_t \end{aligned}$$

is constructed.

4.4 Communication rules

For a function with communication annotations there are three different variations of conformation rules (read only, write only, both read and write). For the latter, it being the most general one, we may consider that for every function $\varphi : \Sigma \times M \to \Gamma \times M$ such that $\varphi(\sigma, (d_1, ...d_n)) = (\gamma, (d'_1, ...d'_n))$, where $d_i, d'_i \in D_i, \sigma \in \Sigma, \gamma \in \Gamma$, for every $q, q' \in Q$ such that $q' \in F(\varphi, q)$, for $(m_i : d_i), (m_i : d'_i)$ representing old and new values of the memory used by φ and for $incoming \in \Sigma$, $T(\varphi) = outgoing$ a rule

$$\begin{array}{rcl} \varphi & : & ((state : q)\ (m_i : d_i) \ldots (m_j : d_j) \\ & & (message : (incoming, sender, *this)) \\ & \rightarrow & (state : q')\ (output : \gamma)\ (m_i : d'_i) \ldots (m_j : d'_j) \\ & & (message : (outgoing', *this, receiver)))_t \end{array}$$

is constructed where $*this$ denotes the identity of the cell containing the rule. *incoming* is a message received from another cell (*sender*) therefore it is of type Σ. *outgoing* is a message to be received by another cell (*receiver*) and thus must be of the input type Σ of the receiver (this being accomplished by $T(\varphi) = outgoing$ which transforms the standard output of the function φ into something understandable for the receiver).

Every transformation rule for a function with communication annotations must also be accompanied by a communication rule that will be responsible for importing the message from a neighbouring cell (receiver) when a bond exists between them. The communication rule resides always on the receiver side and it is of the form: $cr : (\lambda; (message : (incoming, sender, *this)), in)_t$

4.5 Main result

The constructions described by the previous subsections lead to the following:

Theorem 4.1 *For any communicating X-machine working in a synchronous mode, with all sets D_i finite and a given input multi-set there is an equivalent Population P System (produces the same output as the communicating X-machine).*

4.6 Example transformation

The above example of a CXM system consisting of a traffic light TL_1 and a car CAR_1 can be transformed according to the above principles as follows:

So far, we need two types of cells, therefore $K = \{cTL, cCAR\}$ ('c' standing for 'communicating'). There will be two cells, namely $C_{TL_1} = (w_{TL_1}, cTL_{TL_1})$ and $C_{CAR_1} = (w_{CAR_1}, cCAR_{CAR_1})$.

The objects which appear during computation in cell C_{TL_1} will be:

- $(state : q)$, where $q \in \{green, yellow, red, off\}$
- $(time\_left\_to\_change : d_1), (duration\_green : d_2), (duration\_yellow : d_3)$, $(duration\_red : d_4)$ where $d_1 \in \mathbb{N}_0$, and $d_2, d_3, d_4 \in \mathbb{N}$
- $(input : i)$, where $i \in \{tick, power\_on, power\_off\}$
- $(output : o)$, where $o \in \{green, red, yellow, black\}$
- $(message : (push\_break, TL_1, CAR_1))$, $(message : (push\_break\_to\_stop, TL_1, CAR_1))$ and $(message : (push\_accpedal, TL_1, CAR_1))$.

Initially the objects w_{TL_1} are: $(state : green), (time\_left\_to\_change : 20)$, $(duration\_green : 20), (duration\_yellow : 3), (duration\_red : 10)$, which correspond to the initial state and memory values.

The transformation rules for non-communicating functions are indicatively as follows:

$$\begin{array}{lll} keep\_green & : & ((state : green)\ (input : tick)\ (time\_left\_to\_change : tl) \\ & \rightarrow & (state : green)\ (time\_left\_to\_change : tl - 1) \\ & & (output : green))_{cTL}, \text{ if } tl > 0 \\ switch\_off & : & ((state : X)\ (input : power\_off) \\ & \rightarrow & (state : off)\ (output : black))_{cTL} \end{array}$$

The objects which appear during computation in cell C_{CAR_1} will be:

- $(state : q)$, where $q \in \{stopped, accelerating, cruising, breaking\}$
- $(speed : d_1)$, $(decrease\_rate : d_2)$, $(position : d_3)$ where $d_1 \in \mathbb{N}_0$, $d_2 \in \mathbb{N}$ and $d_3 \in \{free\_road, approaching\_light(TL)\}$
- $(input : i)$, where $i \in \{traffic\_light(TL),\ passed\_traffic\_light(TL),\ push\_break,\ push\_break\_to\_stop,\ push\_accpedal,\ leave\_break,\ leave\_accpedal\}$
- $(output : o)$, where $o \in \mathbb{N}_0$ (speed).
- $(message : (push\_break, TL_1, CAR_1))$, $(message : (push\_break\_to\_stop,\ TL_1,\ CAR_1))$ and $(message : (push\_accpedal, TL_1, CAR_1))$.

Initially the objects w_{CAR_1} are: $(state : cruising), (speed : 100), (decrease\_rate : 2), (position : free\_road)$.

Indicatively a transformation rule for the corresponding non-communicating function is:

$$\begin{array}{lll} approaching\_tl & : & ((state : cruising)\ (input : traffic\_light(TL_1)) \\ & & (speed : sp)\ (position : pos) \\ & \rightarrow & (state : cruising)\ (output : sp) \\ & & (speed : sp)\ (position : traffic\_light(TL_1)))_{cCAR} \end{array}$$

As far as communication is concerned, in the cells C_{TL_1} and C_{CAR_1} there will be some transformation rules that correspond to the communicating functions. For example:

$$\begin{array}{lll} change\_yellow & : & ((state : green)\ (input : tick) \\ & & (time\_left\_to\_change : 0)\ (duration\_yellow : dy) \\ & \rightarrow & (state : yellow)\ (message : (push\_break, TL_1, CAR_1)) \\ & & (output : yellow)\ (time\_left\_to\_change : dy))_{cTL} \\ start\_breaking & : & ((state : cruising)\ (decrease\_rate : dr) \\ & & (speed : sp)\ (message : (push\_break, TL_1, CAR_1)) \\ & \rightarrow & (state : breaking)\ (output : sp/dr) \\ & & (decrease\_rate : dr))_{cCAR} \end{array}$$

In addition, cell C_{CAR_1} will have a the communication rules:

$cr_1 : (\lambda; (message : (push\_break, TL_1, CAR_1)), in)_{cCAR}$

$cr_2 : (\lambda; (message : (push\_break\_to\_stop, TL_1, CAR_1)), in)_{cCAR}$

$cr_3 : (\lambda; (message : (push\_accpedal, TL_1, CAR_1)), in)_{cCAR}$

in order to receive messages that appear in C_{TL_1}.

5 Enhancing the model

So far, a set of guidelines have been presented to transform a (static) CXM model to a (static) PPS model. One could enhance the PPS model with features that deal with a potential dynamic structure of the system. For instance:

- if the traffic light malfunctions then it should be removed from the PPS model,
- if the car leaves the traffic light, the bond between the two cells ceases to exist,
- if another car arrives, a new cell should be generated,
- if the new car approaches the traffic light, a bond should be generated, etc.

All the above issues can be dealt with by features of PPS, such as cell death, bond making rules, cell division etc. For the first example, a cell death rule such as $r : ((state : off))_{cTL} \rightarrow \dagger$ will do.

For the rest of the examples, we need to introduce another type of cell which corresponds to the non-communicating counterparts (XMs). This is because two cells that are not connected with a bond should not really have communication rules or transformation rules that correspond to communicating functions. Therefore, it is necessary to introduce two new types in K, namely $genericTL$ and $genericCAR$, which are basically equivalent to the corresponding non-communicating XMs.

So, for example, after a car passes a traffic light, a differentiation rule should change a cell from $cCAR$ type to $genericCAR$ type:

$$\begin{array}{lcl} diffrule_1 & : & ((input : passed\_traffic\_light(TL)))_{cCAR} \\ & \rightarrow & ((input : passed\_traffic\_light(TL)))_{genericCAR} \\ diffrule_2 & : & ((state : green))_{cTL} \\ & \rightarrow & ((state : green))_{genericTL} \end{array}$$

The opposite is also feasible:

$$\begin{array}{lcl} diffrule_3 & : & ((input : traffic\_light(TL)))_{genericCAR} \\ & \rightarrow & ((input : traffic\_light(TL)))_{cCAR} \\ diffrule_2 & : & ((state : yellow))_{genericTL} \\ & \rightarrow & ((state : yellow))_{cTL} \end{array}$$

A bond making rule such as:

$$(cTL, (state : yellow); (input : traffic\_light(TL)), cCAR)$$

will produce a bond between a traffic light and an approaching car.

6 Discussion and Conclusion

We presented a set of principles that guide the transformation of CXM models into PPS models. One of the motives behind this attempt lies in the fact that the resulting PPS model can be further enriched with PPS features that deal with the dynamic nature of the system's structure. There are a few more issues for discussion and further consideration.

Firstly, the objects in the environment w_E in the PPS have not been modelled. In X-machines an environment model per se does not exist. The "environment" provides the inputs in a steam and they are consumed by the functions of the machines

in a timely fashion. In a PPS, we need to consider an equivalent environment. More particularly:

- either the input objects appear in the environment during the computation, or
- input objects are generated by a generator device in an appropriate order.

In both cases, an input object is not as simple as presented in the previous sections. Instead, input objects should be of the form $(input : (\sigma, cell\_identity))$ where *cell_identity* is the cell that the input is for. An additional communication rule in both generic and communicating types of cells is required $r : (\lambda; (input : (\sigma, *this)), enter)_t$ in order for the cells to import the input from the environment. Outputs may be treated in a similar way, i.e. exported to the environment.

Secondly, the direct sending of messages between cells has not been addressed. In a CXM model, a CXM component function sends a message to another CXM component function. In a PPS, a cell cannot directly send a message but instead import a message from another cell as long as they are connected with a bond (due to the bond making rule). For two cells, as presented in the example, this does not appear to be a problem. However, if more than two cells are neighbours, then the transformation rule responsible for producing a messages should be able to produce multiple copies of it. In turn, this would mean that each cell should be aware of the identities of each of its neighbours and therefore it is implied that the identity needs to be communicated once a bond is established.

Finally, we did not deal with the different types of computation, which in CXM is asynchronous whereas in PPS is synchronous. For the specific example, this did not matter much, because the clock ticks and the connectivity of the machines imposes some kind of synchronisation in the CXM model. However, the consequences of the different types of computation in other cases should be further investigated. We anticipate that future work will deal with all these issues.

The current work will facilitate the development of algorithms to automatically translate from a specification to another one. That implies that the tools that have been developed for both methods [1, 8, 15] and their animators, could be linked together to form an integrated environment where transformations are made easy from one model to another and vice-versa.

References

[1] J. Auld, F. Romero-Campero, and M. Gheorghe. P system modelling framework. http://www.dcs.shef.ac.uk/~marian/PSimulatorWeb/P_Systems_applications.htm, November 2006.

[2] F. Bernandini and M. Gheorghe. Population P Systems. *Journal of Universal Computer Science*, 10(5):509–539, 2004.

[3] G. Ciobanu and B. Aman. On the relationship between membranes and ambients. *BioSystems*, 2007. To appear.

[4] D. Corne and P. Frisco. Dynamics of HIV infection studied with cellular automata and conformon-P systems. *BioSystems*, 2007. To appear.

[5] S. Eilenberg. *Automata, Languages and Machines*. Academic Press, 1974.

[6] G. Eleftherakis. *Formal Verification of X-machine Models: Towards Formal Development of Computer-based Systems*. PhD thesis, Department of Computer Science, University of Sheffield, 2003.

[7] M. Holcombe and F. Ipate. *Correct Systems: Building a Business Process Solution*. Springer-Verlag, London, 1998.

[8] E. Kapeti and P. Kefalas. A design language and tool for X-machines specification. In D. I. Fotiadis and S. D. Spyropoulos, editors, *Advances in Informatics*, pages 134–145. World Scientific Publishing Company, 2000.

[9] P. Kefalas, G. Eleftherakis, M. Holcombe, and M. Gheorghe. Simulation and verification of P systems through communicating X-machines. *BioSystems*, 70(2):135–148, 2003.

[10] P. Kefalas, G. Eleftherakis, and E. Kehris. Communicating X-machines: A practical approach for formal and modular specification of large systems. *Journal of Information and Software Technology*, 45(5):269–280, 2003.

[11] P. Kefalas, M. Holcombe, G. Eleftherakis, and M. Gheorghe. A formal method for the development of agent-based systems. In V. Plekhanova, editor, *Intelligent Agent Software Engineering*, pages 68–98. Idea Publishing Group Co., 2003.

[12] J. Klein and M. Koutny. Synchrony and asynchrony in membrane systems. In H. J. Hoogeboom, G. Paun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing, 7th International Workshop, Leiden, Holland*, number 4361 in Lecture Notes in Computer Science, pages 66–85. Springer, 2007.

[13] G. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000. Also circulated as a TUCS report since 1998.

[14] I. Stamatopoulou, M. Gheorghe, and P. Kefalas. Modelling dynamic configuration of biology-inspired multi-agent systems with Communicating X-machines and Population P Systems. In G. Mauri, G. Păun, M. J. Pérez-Jiménez, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing: 5th International Workshop*, volume 3365 of *Lecture Notes in Computer Science*, pages 389–401. Springer-Verlag, Berlin, 2005.

[15] I. Stamatopoulou, P. Kefalas, G. Eleftherakis, and M. Gheorghe. A modelling language and tool for Population P Systems. In *Proceedings of the 10th Panhellenic Conference in Informatics*, Volos, Greece, November 11-13, 2005.

[16] I. Stamatopoulou, P. Kefalas, and M. Gheorghe. Modelling the dynamic structure of biological state-based systems. *BioSystems*, 87(2-3):142–149, February 2007.

[17] I. Stamatopoulou, P. Kefalas, and M. Gheorghe. OPERAS for space: Formal modelling of autonomous spacecrafts. In T. Papatheodorou, D. Christodoulakis, and N. Karanikolas, editors, *Current Trends in Informatics*, volume B of *Proceedings of the 11th Panhellenic Conference in Informatics (PCI'07)*, pages 69–78, Patras, Greece, May 18-20, 2007.

[18] I. Stamatopoulou, P. Kefalas, and M. Gheorghe. OPERAS$_{CC}$: An instance of a formal framework for MAS modelling based on Population P Systems. In *The 8th Workshop on Membrane Computing (WMC'07)*, pages 551–566, 2007.