EXTENDED SIMPLE COLORED PETRI NETS: A TOOL FOR PLANT SIMULATION

Antonio Camurri and Alessandro Coglio

DIST – Dipartimento di Informatica, Sistemistica e Telematica
Università di Genova
Viale Causa 13, I-16145 Genova, Italy
{music, tokamak}@dist.unige.it

ABSTRACT

Extended Simple Colored Petri Nets (ESCP-nets) are a new class of High-level Petri Nets conceived as a good trade-off between Petri Nets (P-nets) and Colored Petri Nets (CP-nets), to be used in plant simulation. ESCP-nets, while being much more convenient and compact than P-nets, are fairly simpler than CP-nets, thus being more easily implemented and more amenable to formal analysis. Furthermore, they possess some features particularly well-suited to plant simulation. The successful employment of ESCP-nets in an industrial application has confirmed their validity.

1. INTRODUCTION

Petri Nets (P-nets) [7, 6] are very well-suited to applications involving plants, in particular plant simulation. The reason is that P-nets can explicitly and conveniently model flows, resource sharing, concurrency, synchronization, and so on, which are important aspects of the workings of plants and hence of their simulation. By adding some explicit time concept to P-nets (this has been done in many ways in the literature), P-net executors can be implemented to simulate plants in order to experimentally obtain performance measures (e.g. throughput, utilization percentages, etc.) which otherwise would be hardly computed by analytical methods. In addition, there is a host of well-established methods to perform various kinds of formal analysis about properties of P-nets ([6] surveys them).

However, as it is well-known, P-nets tend to rapidly increase their size as the complexity of the systems they model increases. Too large P-nets are inconvenient to create and use, because most of their advantages (such as the visual information they convey) are lost. Furthermore, most methods of formal analysis cannot be applied to large P-nets owing to computational complexity. So, it is impractical to use P-nets to model complex, real-world plants.

To overcome this problem (which not only arises in plant simulations, but also in most other application fields of P-nets as well), Colored Petri Nets (CP-nets) [5] have been introduced. Tokens of CP-nets are data items of arbitrary user-defined data types, which are elaborated through arbitrary user-defined functions. This allows complex systems to be conveniently modeled by relatively compact CP-nets. The price to be paid is that most methods of formal analysis do not generalize to CP-nets, and, even more important from certain points of view, implementing CP-nets requires much greater efforts than implementing P-nets.

For these reasons, we have devised a new class of High-level Petri Nets, called *Simple Colored Petri Nets* (*SCP-nets*) [4]. SCP-nets are similar, in spirit, to CP-nets, but they pose some restrictions upon the types which can be defined by the user. Roughly speaking, they only allow enumerative data types (i.e. types whose items must be explicitly enumerated one by one). Despite this restriction, SCP-nets are extremely useful in many fields, including (but not limited to) plant simulation. An SCP-net can in fact be "exponentially" smaller than a corresponding P-net. The advantage over CP-nets, is their much greater simplicity, which results in highly reduced implementation efforts. Furthermore, SCP-nets seem to be more amenable to formal analysis, and we are doing some research in this direction.

To better employ SCP-nets in plant simulation, we have found it useful to enhance them by three features. First, we have introduced a "built-in" data type consisting of floating point numbers, very useful to model continuous quantities (e.g. tons of pooled materials, sizes of objects, and so on). Second, we have added a simple but powerful explicit time concept, in order to model durations of processes (e.g. the duration of a machine operation). Third, we have provided an "interface" through which an external supervisor can model high-level scheduling strategies (e.g. aimed at optimizing performance and/or avoiding deadlocks) which depend on the overall state of the simulated plant. We have called these enhanced SCP-nets *Extended Simple Colored Petri Nets* (*ESCP-nets*).

In Sec. 2 we describe ESCP-nets. In Sec. 3 we briefly present an industrial application of ESCP-nets. Finally, in Sec. 4 we draw some conclusions and outline future work.

2. DESCRIPTION

We now give an informal description of ESCP-nets; formal definitions can be found in [3]. To ease understanding, as we present concepts we instantiate them to the simple ESCP-net depicted in Fig. 1. Such an ESCP-net models an artificial abstract process where unpainted cubes and spheres (of various sizes) get painted with red, green, or blue paint, by means of suitable subtractive color syntheses of magenta, yellow, and cyan paint. Despite its simplicity, this example exhibits all the key features of ESCP-nets.

Topology

The *topology* of an ESCP-net is defined in the "standard" way: *places*, *transitions*, and *arcs* connecting them. A place and a transition can be connected by more than one arc in the same direction. In Fig. 1, places, transitions, and arcs, are respectively represented as circle, rectangles, and arrows, according to the usual graphical convention (for now, ignore the shading of the transition START).

Tokens

The tokens of an ESCP-net are defined by means of a *token taxonomy*, i.e. a finite directed acyclic graph (DAG) of identifiers, like that in Fig. 1. The terminal nodes of the DAG (e.g. cyan, none, cube) are called *base tokens*, while the non-terminal nodes (e.g. Color, Secondary, Shape) are called *base types*. Given a base token k and a base type y, we say that k *has type* y iff there exists a path in the DAG from y to k (e.g. yellow has both type Color and Primary, but not Shape or Secondary; sphere has type Shape but not Color). Given two base types y and y', we say that y is a *super-type* of y' and that y' is a *sub-type* of y iff there exists a non-empty path in the DAG from y to y' (e.g. Primary is a sub-type of Color and Color is a super-type of Primary; Shape has no sub-types or super-types; Color has two sub-types but no super-types).

In addition, for any ESCP-net all the real numbers are also called *base tokens*, the distinguished identifier **R** is also called *base type*, and we say that each real number *has type* **R**. In other words, **R** is a sort of implicit and predefined base type for any ESCP-net, whose base tokens are the real numbers. However, **R** has no sub-types or super-types.

A *token* is a finite concatenation $k_1;\ldots;k_n$ of $n \geq 1$ base tokens (e.g. magenta;80, cube;2.5;green). A *type* is a finite concatenation $y_1;\ldots;y_m$ of $m \geq 1$ base types (e.g. Color;**R**, Shape;**R**;Color). (So, as expected, each base token is also a token, and each base type is also a type.) We say that $k_1;\ldots;k_n$ *has type* $y_1;\ldots;y_m$ iff $n = m$ and each k_i has type y_i (e.g. magenta;80 has both type Color;**R** and Primary;**R**, but not Secondary;**R** or **R**;Color; cube;2.5;green has type Shape;**R**;Color). Given two types $y = y_1;\ldots;y_m$ and $y' = y_1';\ldots;y_{m'}'$, we say that y is a *super-type* of y' and that y' is a *sub-type* of y iff $y \neq y'$, $m = m'$, and each y_i is either equal to y_i' or is a super-type of y_i' (e.g. Color;**R** is a super-type of Primary;**R**).

Each place of an ESCP-net is labeled by a type, and the place can only be *marked* by (i.e. "contain" occurrences of) tokens of that type. Each place can be marked by multiple occurrences of a same token, in fact the marking of a place is a multiset of tokens. For instance, UNPAINTED is labeled by Shape;**R**;Color, and is in fact marked by cube;2;none and sphere;3;none, which respectively represent an unpainted cube of edge 2 (in some unspecified units), and an unpainted sphere of radius 3. As another example, TANKS1 is labeled by Primary;**R**, and is in fact marked by magenta;80, yellow;76, and cyan;41, which represent three tanks of paint in the indicated quantities (in some unspecified units). (Note that in Fig. 1 we use abbreviations for the base token identifiers inside places.)

Expressions

The arcs of an ESCP-net are labeled by *expressions*, which are built out of tokens, variables, and functions.

Each *variable* has an associated type, and only tokens of that type can be assigned to the variable. For instance, in Fig. 1 the variable *pc1* has Primary as associated type. While yellow can be assigned to *pc1*, red or cube;4.55 cannot.

Each *function* has two associated types (respectively domain and range), and maps tokens of its domain to tokens of its range. For instance, in Fig. 1 the function *subsyn* maps tokens of type Primary;Primary to tokens of type Color, as follows:

subsyn (magenta;yellow) = red ,
subsyn (yellow;magenta) = red ,
subsyn (yellow;cyan) = green ,
subsyn (cyan;yellow) = green ,
subsyn (cyan;magenta) = blue ,
subsyn (magenta;cyan) = blue ,
subsyn (magenta;magenta) = magenta ,
subsyn (yellow;yellow) = yellow ,
subsyn (cyan;cyan) = cyan .

In other words, *subsyn* returns the color obtained by subtractive synthesis of the two (primary) colors given as arguments. The function *area* is instead defined as

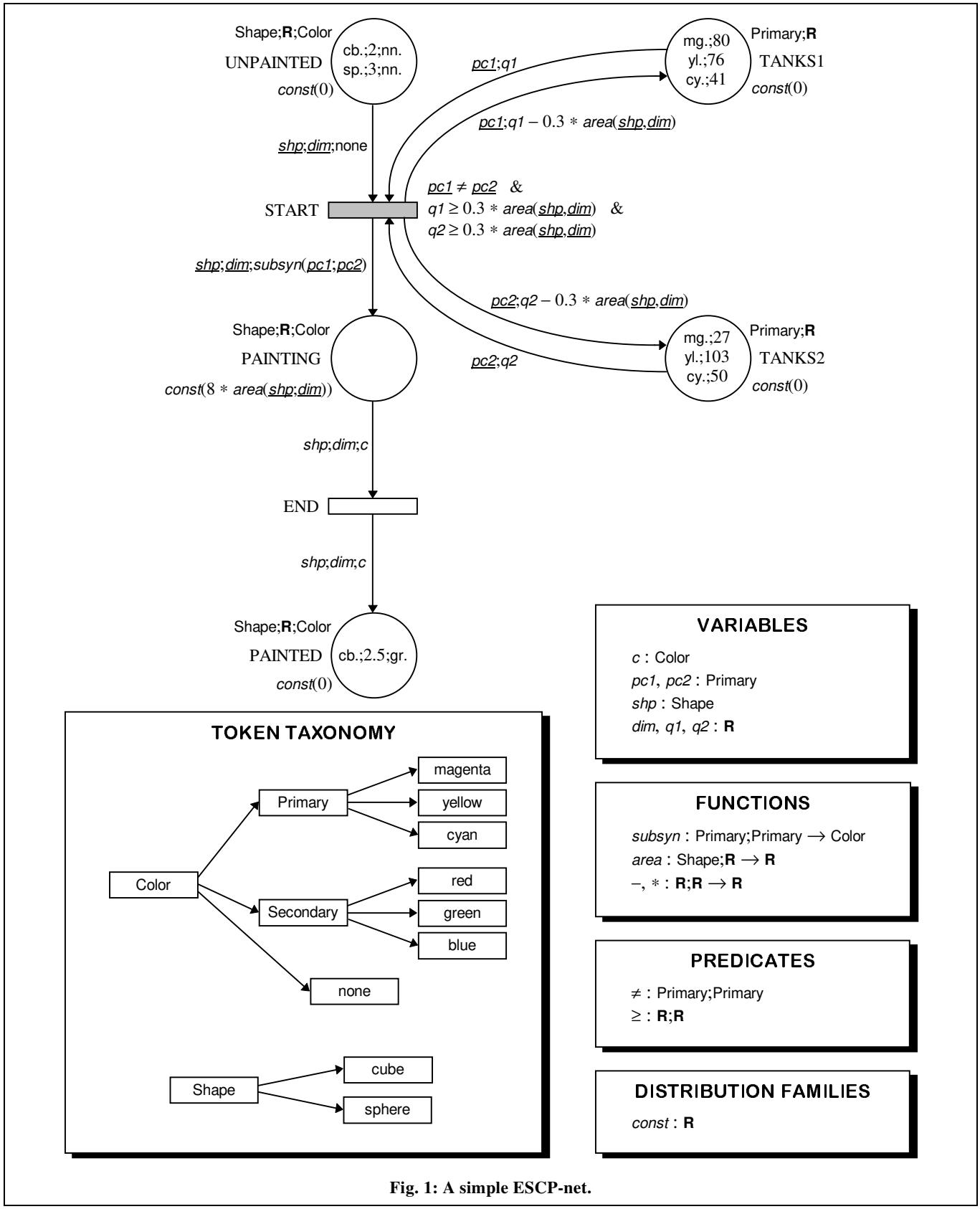

Fig. 1: A simple ESCP-net.

follows:

$\forall\, x$ real number : $\mathit{area}(\text{cube};x) = 6 \cdot x^2$,

$\forall\, x$ real number : $\mathit{area}(\text{sphere};x) = 4 \cdot \mathit{pi} \cdot x^2$.

In other words, *area* returns the surface area of a cube or sphere whose edge or radius (respectively) measures x (*pi* denotes the ratio of a circumference measure to its

diameter). Finally, the functions − and ∗ have the obvious semantics of subtraction and multiplication of real numbers.

In Fig. 1 the arc from UNPAINTED to START is labeled by the expression *shp*;*dim*;none (for now, ignore the underlining of *shp* and *dim*), while the arc from START to TANKS1 is labeled by the expression *pc1*;*q1*–0.3∗*area*(*shp*;*dim*) (note that we use an infix notation for − and ∗). Similarly to a token, each expression has one or more types (e.g. *shp*;*dim*;green has type Shape;**R**;Color and also Shape;**R**;Secondary). It is required that the expression labeling an arc has the type labeling the place which the arc is attached to.

A *binding* is an assignment of tokens to some variables, where of course each token must have the type associated to the variable to which it is assigned. Given a binding for all the variables present in an expression, the expression evaluates to a token. For instance, if we assign cube to *shp*, 2 to *dim*, magenta to *pc1*, and 80 to *q1*, the expression *pc1*;*q1*–0.3∗*area*(*shp*;*dim*) evaluates to the token magenta;72.8.

Guards

The transitions of an ESCP-net are optionally labeled by *guards*, which are built out of expressions, predicates, and logical connectives.

Each *predicate* has an associated type, and maps each token of that type to **T** (true) or **F** (false). For instance, in Fig. 1 ≠ maps, as expected, tokens of type Primary;Primary to **T** if the two components base tokens are different, to **F** otherwise. The predicate ≥ has the obvious semantics of the greater-than-or-equal-to relation over real numbers.

In Fig. 1 START is labeled by the guard (*pc1* ≠ *pc2* & *q1* ≥ 0.3∗*area*(*shp*;*dim*) & *q2* ≥ 0.3∗*area*(*shp*;*dim*)), where of course & is the logical connective for conjunction (note that we use an infix notation for both ≠ and ≥).

Given a binding for all the variables present in a guard, the guard evaluates to **T** or **F**. For instance, if we assign cube to *shp*, 2 to *dim*, magenta to *pc1*, cyan to *pc2*, 80 to *q1*, and 50 to *q1*, the guard labeling START evaluates to **T**.

Firings

Given a transition and a binding for all the variables present in the expressions labeling all the arcs surrounding the transition, the *firing* of the transition with such a binding amounts to first removing tokens from the input places of the transition, then adding tokens to the output places, as indicated by the tokens which the corresponding arc expressions evaluate to. However, in order for a transition to be *enabled* with such a binding, all the tokens to be removed must be present in the input places, and furthermore, if a guard labels the transition, it must evaluate to **T**.

For instance, consider START in Fig. 1, and a binding assigning cube to *shp*, 2 to *dim*, magenta to *pc1*, 80 to *q1*, cyan to *pc2*, and 50 to *q2*. It is easily seen that START is enabled with this binding. Its firing has the following effects: first cube;2;none is removed from UNPAINTED, magenta;80 from TANKS1, and cyan;50 from TANKS2; then, cube;2;blue is added to PAINTING, magenta;72.8 to TANKS1, and cyan;42.8 to TANKS2. This firing represent the fact that an unpainted cube of edge 2 starts to be painted with blue paint, obtained by mixing together magenta and cyan paint in equal proportions, consuming quantities of these paints proportional (according to a factor 0.3 in some unspecified units) to the surface area of the cube.

Times

In ESCP-nets, each token is required to stay in a place a certain amount of time before being allowed to leave the place. This is achieved by attaching a natural number, called *waiting time*, to each (occurrence of) token marking each place. The waiting time indicates the residual time (in some discrete units) to wait, and is meant to be decremented as time passes. A token may leave a place only if its waiting time is 0. (Note that in Fig. 1 we have omitted waiting times.) The waiting time of a token entering a place is initialized according to a *time probability distribution*, i.e. an infinite sequence $(\pi_0, \pi_1, \ldots)$ of real numbers between 0 and 1 (inclusive), such that $\pi_0 + \pi_1 + \ldots = 1$. Each π_i represents the probability that the initial waiting time of the token is i.

Instead of simply labeling each place by a time probability distribution (according to which the waiting time of each token entering the place would be initialized), we adopt a more flexible approach. Each ESCP-net contains, besides variables, functions, and predicates, also *distribution families*, each of which has an associated type. Each distribution family maps tokens of the associated type to time probability distributions. For instance, in Fig. 1 the distribution family *const* maps each real number (which is a token of type **R**) x to the time probability distribution such that $\pi_i = 0$ for all i except the result j of rounding x, for which we have $\pi_j = 1$ (in case x is negative, we can totalize *const* by just taking $j = 0$, however this should never happen unless the ESCP-net is ill-designed). Each place is labeled by a *stochastic time*, which is a syntactical entity (like expressions and guards) of the form *df*(*e*), where *df* is a distribution family and *e* is an expression having the

type associated to *df*. For instance, in Fig. 1 PAINTING is labeled by the stochastic time *const*(8∗*area*(*shp*;*dim*)). So, when a transition fires with a binding, the stochastic times labeling the output places of the transition are evaluated, according to the binding, to time probability distributions, and the waiting times of the tokens entering the places are initialized according to such time probability distributions. For instance, the waiting time of each token entering PAINTING is initialized to a value proportional, according to a factor 8 (in some unspecified units), to the surface area of the cube or sphere to be painted. Only after such number of time units has elapsed, the token is allowed to leave a place and "move" to PAINTED, through the firing of END. This models the fact that the actual painting process takes a time proportional to the surface area of the object being painted.

The advantage of the above approach is that for each place we just specify the family *df* of possible time probability distributions, and the expression *e* according to which one distribution of the family is chosen. In this way, different tokens can have different waiting time statistics. Note that if places were just labeled by fixed time probability distributions, there would be no way to have tokens wait in PAINTING amounts of time proportional to the surface areas. Of course, fixed time probability distributions can be obtained just by means of stochastic times containing constant expression (e.g. all the places in Fig. 1 other than PAINTING are labeled by *const*(0)).

In the ESCP-net in Fig. 1, if we wanted tokens to wait in PAINTING amounts of time with gaussian distributions, with mean values and standard deviations proportional (according to factors 8 and 2, respectively) to the surface areas, it would be sufficient to do the following. First, we would replace *const* with a distribution family *gauss*, with associated type **R**;**R**, mapping each token $x_1;x_2$ of type **R**;**R** to a gaussian-shaped time probability distribution with mean and standard deviation x_1 and x_2 respectively. Then, we would re-label PAINTING with the stochastic time *gauss*(8∗*area*(*shp*;*dim*);2∗*area*(*shp*;*dim*)).

Control Interface

An ESCP-net has a *control interface* through which it can be externally supervised. A control interface consists of a distinguished set of transitions, called *controlled transitions*, and, for each of them, a distinguished subset of the variables surrounding it, called *controlled variables*. The "semantics" of a control interface is that an external supervisor can explicitly enable or disable controlled transitions, and, for each controlled transition it enables, it can specify the tokens to be assigned to the controlled variables.

For instance, in Fig. 1 START is a controlled transition, as indicated by the shading, and *shp*, *dim*, *pc1*, and *pc2* are its controlled variables, as indicated by the underlining. This means that an external supervisor can decide when to enable START (i.e. when the painting of a new cube or sphere starts), and, by assigning tokens to the four controlled variables, can decide which unpainted object (among those represented by the tokens which mark UNPAINTED) is painted and by mixing which primary colors.

3. AN INDUSTRIAL APPLICATION

We have successfully employed ESCP-nets in an industrial application for Demag Italiampianti (the largest Italian industry producing plants), consisting in implementing an architecture for plant simulation that we have designed [2]. The core of the architecture is an executor of ESCP-nets. An expert system externally supervises the executor, through the ESCP-net control interface, by means of rules about the marking of the ESCP-net. Details of the architecture and of its implementation, can be found in [2].

As part of this industrial application, we have also realized a visual editor to create and edit ESCP-nets, through an intuitive graphical interface. ESCP-nets can be saved as ASCII files (in a special-purpose format that we have defined), and thus used in possibly many different applications. Our visual editor offers a lot of facilities to the user, such as recursively hiding ESCP-net fragments, called *ESCP-subnets*, inside so-called *macro places*, so that a same ESCP-subnet can be used inside larger ESCP-(sub)nets for modular development.

The success of this industrial application provides evidence of the validity of ESCP-nets as a tool for plant simulation. In fact, the use of our architecture to realize plant simulators results in dramatic reductions of development times and errors, with respect to the use of general-purpose programming languages like C or C++. For instance, through our implementation of the architecture we have realized a simulator of a real-world plant (namely, the raw material handling area of the steel-plant at Servola-Trieste, in Italy) in a few days, while another simulator of the same plant and at the same level of detail had been developed in Fortran by Demag Italiampianti in several weeks. Furthermore, simulators realized through our architecture are much more readable than source code listings, and hence easier to understand and to maintain.

4. CONCLUSIONS AND FUTURE WORK

ESCP-nets constitute an extremely valuable tool for

plant simulation. Even if the features added to SCP-nets also add some complexity, ESCP-nets are still, like SCP-nets, a good trade-off between the simplicity of P-nets and the convenience of CP-nets. Their implementation in fact requires much less effort than CP-nets. Furthermore, there are formal results which relate the behavior of ESCP-nets to that of SCP-nets (see [3]), thus easing formal analysis.

Although ESCP-nets have been designed for plant simulation, we have planned to investigate other applications as well. In particular, we are working about multimedia systems for novel interactive applications in music, art, entertainment, and museums [1]. ESCP-nets are a promising conceptual tool to support the design of agents capable of establishing creative, multimodal user interaction, by exhibiting real-time adaptive behavior.

We have also planned to enhance our ESCP-net visual editor by allowing the development of parameterized ESCP-subnets, which can be instantiated with actual parameters and used inside larger ESCP-(sub)nets. For instance, a "buffer" ESCP-subnet might be defined, parameterized on the dimension, type of tokens to be stored, and buffering policy (e.g. FIFO, LIFO). Such a feature would greatly increase modularity of development.

5. REFERENCES

[1] A. Camurri, "Network Models for Motor Control and Music", in P. Morasso, V. Sanguineti (eds.), *Self-Organization, Computational Maps and Motor Control*, Elsevier Science B.V., 1997.

[2] A. Camurri, A. Coglio, "A Petri Net-based Architecture for Plant Simulation," to appear in *Proceedings of the 6th IEEE Conference on Emerging Technologies and Factory Automation*, UCLA, Los Angeles, USA, September 9–12, 1997.

[3] A. Camurri, A. Coglio, *Extended Simple Colored Petri Nets*, DIST Technical Report, in preparation.

[4] A. Camurri, A. Coglio, *Simple Colored Petri Nets*, DIST Technical Report, in preparation.

[5] K. Jensen, "Coloured Petri Nets: A High Level Language for System Design and Analysis", in G. Rozenberg (ed.), *Advances in Petri nets 1990*, Lecture Notes in Computer Science, vol. 483, Springer-Verlag, 1990, pp. 342–416.

[6] T. Murata, "Petri Nets: Properties, Analysis and Applications", in *Proceedings of the IEEE*, 77(4), pp. 541-580, April 1989.

[7] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*, Englewood Cliffs, NJ: Prentice Hall, 1981.