A Petri Net-based Architecture for Plant Simulation

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

Abstract – **We propose a hybrid architecture for the simulation of plants. Its core is a Petri Net executor based on a new class of Petri Nets called Extended Simple Colored Petri Nets. The executor is supervised by an expert system. Our architecture is very well-suited for the software-engineering development of plant simulators, as development times and errors can be dramatically reduced. The validity of our approach has been demonstrated by means of an implementation that we have realized as an industrial project for Demag Italimpianti.**

I. INTRODUCTION

In this paper we deal with plant simulations aimed at experimentally obtaining performance measures (e.g. throughput, waiting times, utilization percentages, etc.) of plants, also useful to discover bottlenecks, evaluate the impact of proposed modifications, and so on. While such performance measures can hardly be computed by analytical methods, they can be easily obtained by running simulators for long enough times and having them collect and output all the interesting data.

Building a simulator in any general-purpose programming language (e.g. C, C++) is not an easy task. It is necessary to model flows of materials and parts, concurrency of operations, synchronization of processes, use of shared resources, and so on. Furthermore, high-level scheduling strategies, which depend on the overall state of the plant, must also be modeled.

In this paper we propose a hybrid architecture for plant simulators. Its core is a Petri Net executor based on *Extended Simple Colored Petri Nets* (*ESCP-nets*) [1, 2], which models flows, concurrency, synchronization, resource sharing, etc. The executor is supervised by an expert system, which models the high-level scheduling strategies by means of rules concerning the marking of the ESCP-net. This architecture is very well-suited for the software-engineering development of plant simulators: its capability to conveniently model the most crucial aspects of plant simulation, allows dramatic reductions of development times and errors.

The validity of our approach has been demonstrated by means of an implementation of the architecture that we have realized as an industrial project for Demag Italimpianti, the largest Italian industry producing plants.

In Section II we describe a real-world plant, namely the raw material handling area of the steel-plant at Servola-Trieste (Italy). In Section III we give an overview of the architecture through the description of a simulator, which we have realized using our implementation, of the Servola-Trieste plant. Finally, in Section IV we draw some conclusions and describe future work.

II. A REAL-WORLD PLANT

Fig. 1 sketches the raw material handling area at Servola-Trieste. The fossil pool consists of four (heaps of) raw materials whose codes are CF1, CF2, CF3, CF4. The mineral pool consists of eight (heaps of) raw materials whose codes are PEH, PEG, P1, P2, F1, F2, F3, F4. While the blast-furnace is working, these twelve materials flow at constant rates from the pools to the blast-furnace. Raw materials are brought to the pools by means of eight kinds of ships, each of which carries one, two, or three fixed materials in fixed quantities. As soon as the quantity of a raw material in the pools goes below a critical threshold, a fixed ship (carrying that material) is "called" (unless a ship carrying it is already coming). Arriving ships are queued in the roadstead, as only one ship may occupy the berth. When the berth is free, the first ship in the roadstead is moored, its materials are unloaded to the pools, and then it is unmoored. If, owing to ship delays, some raw material runs out, the blast-furnace is turned off; it is turned on again as soon as the needed material arrives.

Our simulator of this plant, described in the next section, is as detailed as an existing simulator of the same plant developed in Fortran by Demag Italimpianti.

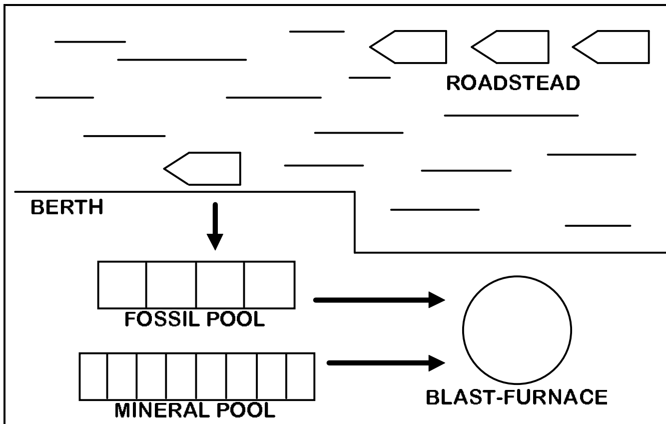

Fig. 1: A sketch of the Servola-Trieste plant.

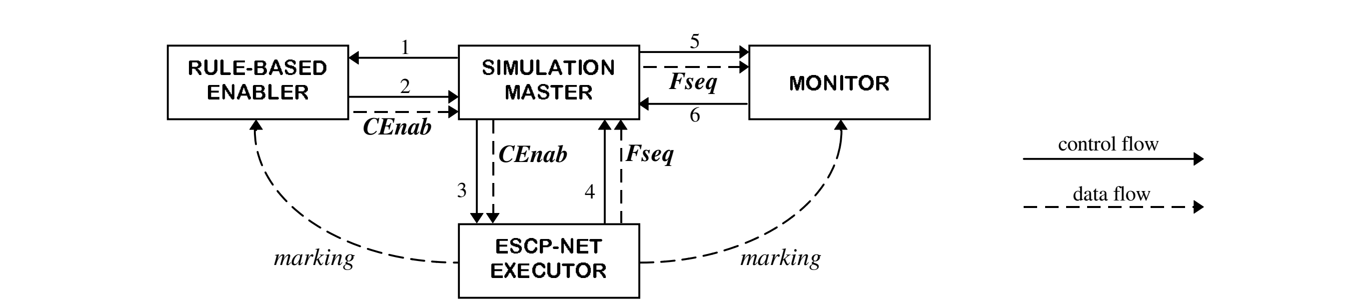

Fig. 2: The organization of our architecture.

III. OVERVIEW OF THE ARCHITECTURE

Fig. 2 shows the overall organization of our architecture. The simulation is run by the *master*, which cyclically calls the other three modules, cycles being delimited by evenly spaced instants of the simulated time axis. Each cycle consists in first calling the *rule-based enabler* which returns *CEnab*, then the *ESCP-net executor* with *CEnab* which returns *Fseq*, and finally the *monitor* with *Fseq*. In the remainder of this section, we describe ESCP-nets, the executor, the enabler, and the monitor (and we also precise what *CEnab* and *Fseq* are), and we conclude with a brief description of our implementation of the architecture.

A. Extended Simple Colored Petri Nets

ESCP-nets [1, 2] are a new class of Petri Nets. As implied by their name, they constitute an extension of *Simple Colored Petri Nets* (*SCP-nets*) [3]. SCP-nets are conceived as a good trade-off between "classical" Petri Nets (P-nets) [6] and Colored Petri Nets (CP-nets) [5]: while being much more convenient and compact than P-nets, they are fairly simpler than CP-nets, thus being more easily implemented and more amenable to formal analysis. ESCP-nets add some features to SCP-nets to make them well-suited to be used in plant simulations, possibly being externally supervised (e.g. by an expert system, as in our architecture).

The tokens of an ESCP-net are defined by a *token taxonomy*, i.e. a finite directed acyclic graph (DAG) of identifiers (like the one in Fig. 3). The terminal nodes (e.g. cf2, ready, none) are *base tokens*, the non-terminal nodes (e.g. Material, Fossil, State) are *base types*. Given a base token k and a base type y, k *has type* y iff there exists a path from y to k in the DAG (e.g. s2 has type Ship, peg has both type Mineral and Material, off has not type Free). In addition, for any ESCP-net, real numbers are also base tokens, the distinguished identifier **R** is also a base type, and each real number has type **R**. A *token* is a finite sequence $k_1;\ldots;k_n$ of $n \geq 1$ base tokens (e.g. peg;1000.5, free), and a *type* is a finite sequence $y_1;\ldots;y_m$ of $m \geq 1$ base types (e.g. State, Material;**R**). $k_1;\ldots;k_n$ *has type* $y_1;\ldots;y_m$ iff $n = m$ and each k_i has type y_i (e.g. cf3;4000 has both type Fossil;**R** and

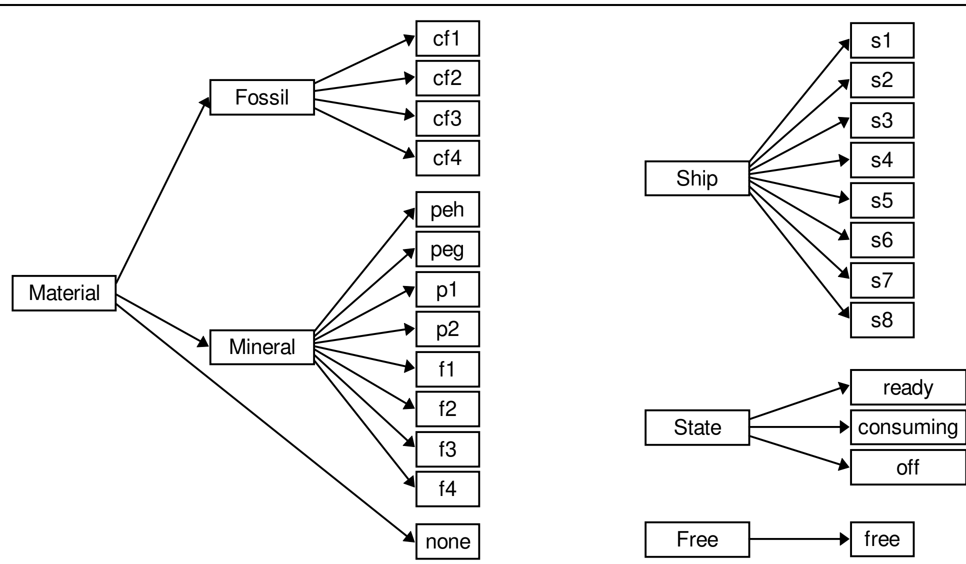

Fig. 3: The token taxonomy for the Servola-Trieste plant.

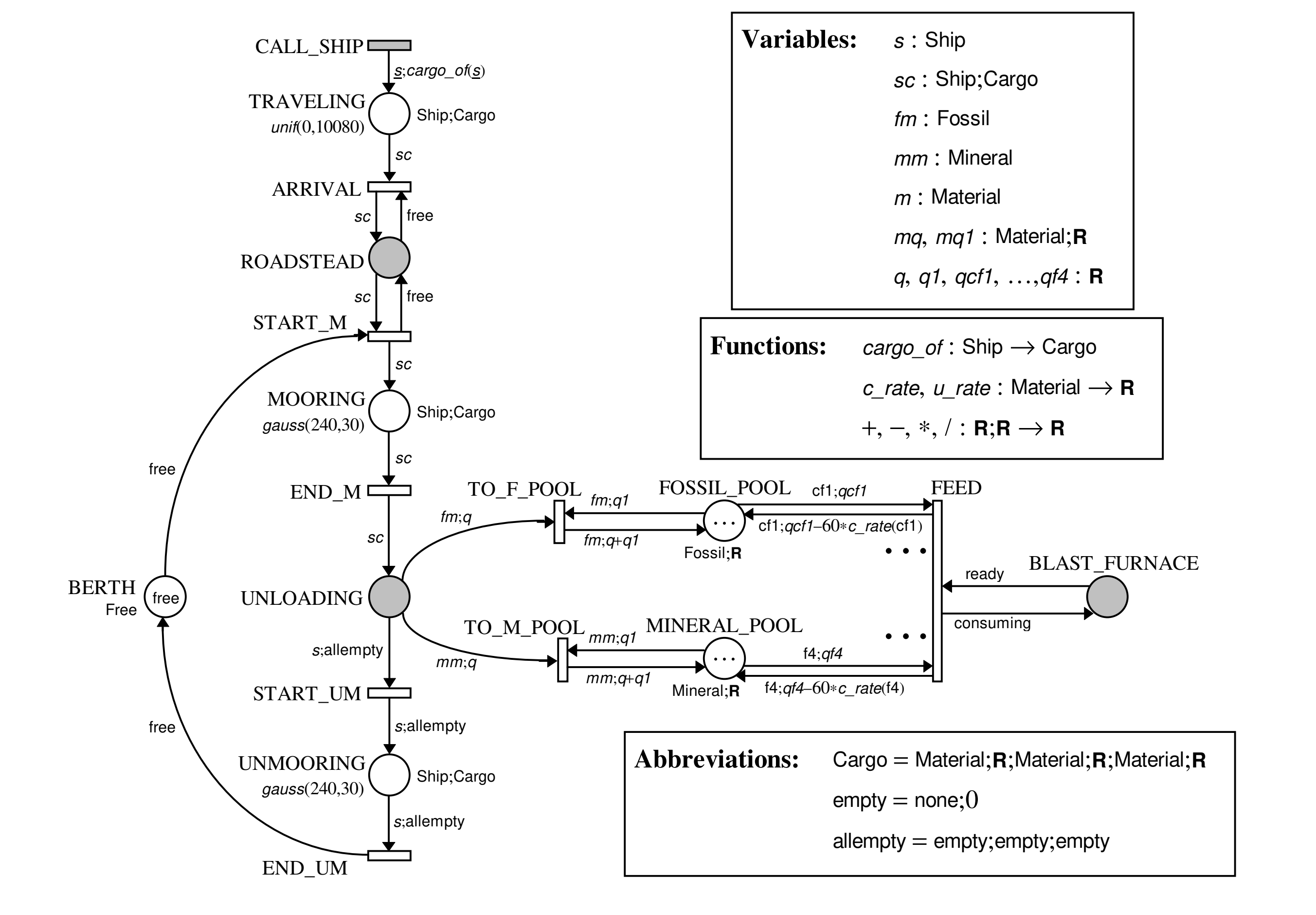

Fig. 4: The ESCP-net for the Servola-Trieste plant.

Material;**R**, but not type **R**;Fossil or Ship). Even though token taxonomies often happen to consist of one or more disjoint trees (as in Fig. 3), in general a DAG allows more flexibility (e.g. overlapping independent classifications of materials).

An ESCP-net contains places, transitions, and arcs connecting them, as in Fig. 4. There are also *macro places* (e.g. ROADSTEAD, as indicated by the shading), i.e. place-holders for ESCP-net fragments called *ESCP-subnets*, like those in Fig. 5. An ESCP-net may contain recursively nested ESCP-subnets, but it can be always flattened by recursively replacing each macro place with its corresponding ESCP-subnet.

Each place is labeled by a type, and it can be *marked* by tokens of that type only. For instance, FOSSIL_POOL is labeled by Fossil;**R**, and it is always marked by four tokens cf1;x_1, cf2;x_2, cf3;x_3, and cf4;x_4, where x_1, x_2, x_3, and x_4 are the tons of the respective materials present in the fossil pool (we have indicated such tokens by ellipses to reduce graphical cluttering). MINERAL_POOL, labeled by Mineral;**R**, is marked analogously.

Each arc is labeled by an *expression*, which may contain tokens, *variables*, and *functions*. Each variable has a type, and it can only be assigned tokens of that type (e.g. *mq* has type Material;**R**, and can be assigned f1;32000 but not s2 or ready). Each function maps tokens of its domain to tokens of its range, e.g. *cargo_of* maps each ship (identifier) to the fixed materials in fixed quantities carried by it (note that none;0 denotes the absence of a material):

cargo_of (s1) = peg;32000;p1;32000;none;0 ,
cargo_of (s2) = peh;64000;none;0;none;0 ,
cargo_of (s3) = peg;32000;none;0;none;0 ,
cargo_of (s4) = p2;32000;f1;32000;none;0 ,
cargo_of (s5) = p2;32000;none;0;none;0 ,
cargo_of (s6) = f2;8797;f3;26165;f4;25038 ,
cargo_of (s7) = cf1;15000;cf2;15000;cf3;30000 ,
cargo_of (s8) = cf4;30000;none;0;none;0 .

A *binding* is an assignment of tokens to variables. Given a binding for the variables of an expression, the expression evaluates to a token (e.g. if we assign s8 to *s*, the expression *s*;*cargo_of*(*s*) evaluates to s8;cf4;30000;none;0;none;0).

The *firing* of a transition with a binding for the variables "surrounding" the transition, amounts to first removing tokens from the input places and then adding tokens to the

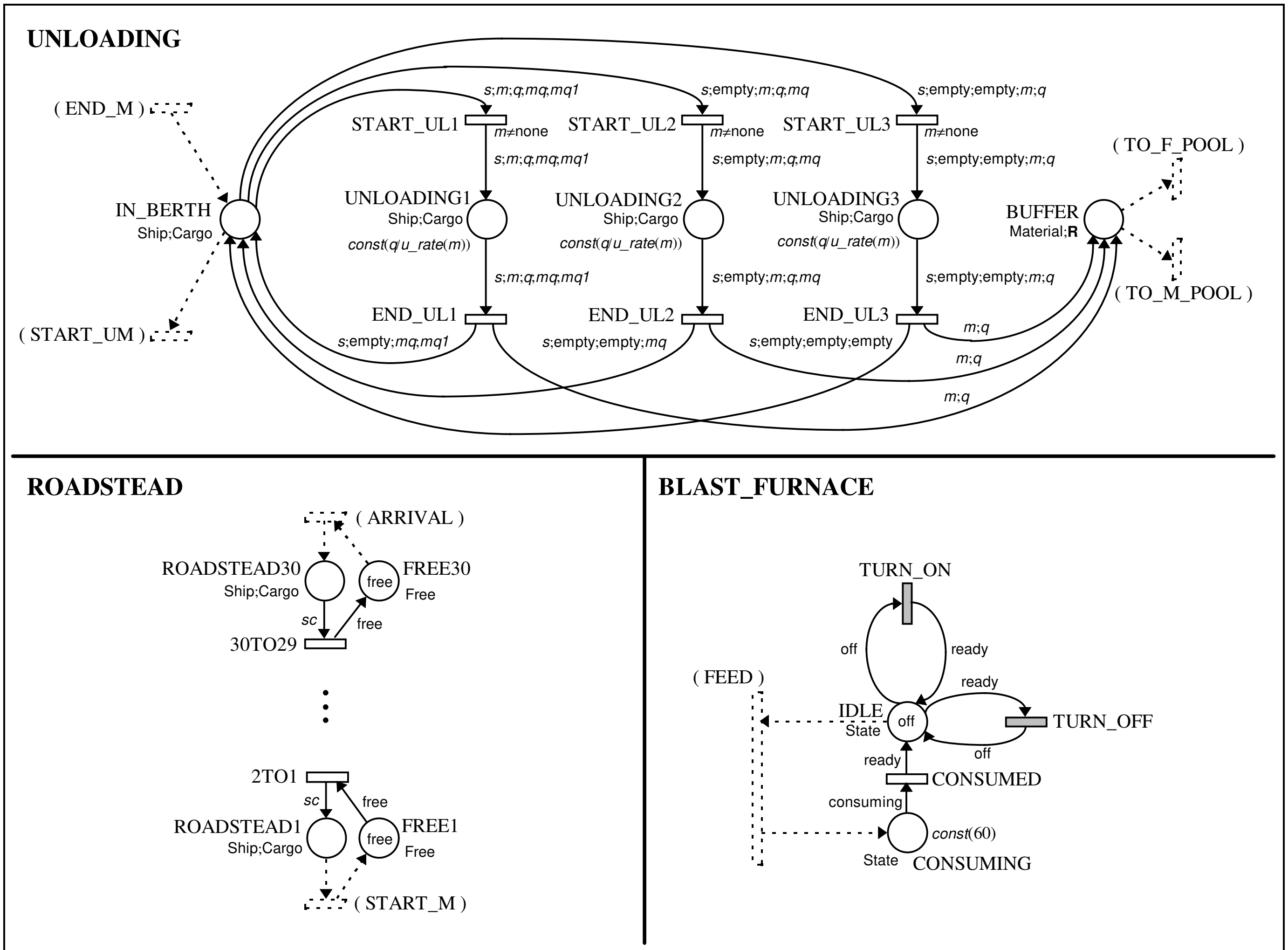

Fig. 5: The ESCP-subnets for the Servola-Trieste plant.

output places, as indicated by the tokens which the corresponding arc expressions evaluate to. Of course, the tokens to be removed must be present in the input places, in order for the transition to be *enabled* with the binding. For instance, TO_F_POOL is enabled with a binding which assigns cf2 to *fm*, 15000 to *q*, and 80000 to *q1*, iff BUFFER contains a token cf2;15000 and FOSSIL_POOL a token cf2;80000. If enabled, its firing removes cf2;15000 from BUFFER and replaces cf2;80000 with cf2;95000 in FOSSIL_POOL.

Each transition can be labeled by a *guard*, which, if present, must be satisfied in order for the transition to be enabled. For instance, START_UL1 is labeled by $m \neq$ none, satisfied iff the token assigned to *m* is not none.

Each token must wait a certain amount of time before being allowed to leave a place. In fact, each token marking an ESCP-net has an associated *waiting time*, i.e. a natural number indicating the residual time (in some discrete units) to wait, to be decremented as time passes. A token may leave a place only if its waiting time is 0. The waiting time of a token entering a place is initialized according to a *stochastic time* labeling the place. For instance, MOORING and UNMOORING are both labeled by *gauss*(240,30), which means that initial waiting times of tokens entering them have a gaussian distribution with mean 240 and standard deviation 30 (both values representing minutes). The initial waiting times in TRAVELLING have a uniform distribution between 0 and 10080 (10080 minutes is 7 days). Stochastic times may contain non-constant expressions, so that different tokens may have slightly different distributions in a same place, as in UNLOADING1, where *const*(*q*/*u_rate*(*m*)) indicates a constant (i.e. deterministic) initial waiting time determined by the quantity *q* of material and the unloading rate of the material *u_rate*(*m*).

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*). Controlled transitions must be explicitly enabled by the external supervisor in order to fire, and the supervisor must also specify the tokens to be assigned to controlled variables. For instance, CALL_SHIP is a controlled transition (as indicated by the shading), and *s* is its (only) controlled variable (as indicated by the underlining), as the expert system decides which ship must be called when.

We conclude this subsection with some remarks to complete the description of Fig. 4 and Fig. 5. To reduce graphical cluttering, we have omitted all the *const*(0) stochastic times. The roadstead is modeled as a FIFO buffer with 30 slots (which is enough). The expressions and guards in the ESCP-subnet UNLOADING constrain the materials carried by a ship to be unloaded in order. The working of the blast-furnace is "discretized" into one-hour cycles (as in the Fortran simulator of the plant) each starting with the firing of FEED, which causes the quantity of each material consumed in one hour by the blast-furnace to be removed from the pools (most arcs between FEED and FOSSIL_POOL, and between FEED and MINERAL_POOL, are indicated by ellipses to reduce graphical cluttering). The function *c_rate* in fact returns the consumption rate of the argument material. The blast-furnace is turned off or on (upon decision of the expert system) by respectively firing TURN_OFF or TURN_ON (which are in fact controlled transitions).

B. The Executor

The input *CEnab* of the executor is a finite multiset of pairs $\langle t, \beta' \rangle$, where t is a controlled transition and β' is a binding for its controlled variables. *CEnab* specifies which controlled transitions are allowed to fire with which values assigned to their controlled variables, and it is a multiset, instead of simply a set, to allow a same firing to take place more than once. The output *Fseq* of the executor is a finite sequence of pairs $\langle t, \beta \rangle$, where t is a transition and β is a binding for its surrounding variables. *Fseq* orderly specifies which transitions have fired with which bindings.

The actions performed by the executor when it is called, can be conceptually explained as follows. First, the waiting time of each token marking the ESCP-net is decremented by 1 (unless it is 0, in which case it is unaltered). Furthermore, *Fseq* is set to the empty sequence. After that, the executor computes the set *Enab* of all pairs $\langle t, \beta \rangle$ such that t is a transition, β is a binding for its variables, t is enabled with β in the current marking, and, in case t is controlled, $\langle t, \beta' \rangle$ is in *CEnab* for some β' assigning the same tokens as β to the controlled variables. By virtue of some restrictions (which we did not mention in the previous subsection for brevity) about the expressions labeling incoming and outgoing arcs of transitions, it can be shown that *Enab* is always finite. So, an element $\langle t, \beta \rangle$ in *Enab* is randomly chosen, and t fires with β. The pair $\langle t, \beta \rangle$ is appended to *Fseq*. If t is a controlled transition, one occurrence of the $\langle t, \beta' \rangle$ from which $\langle t, \beta \rangle$ was computed, is removed from *CEnab*; otherwise *CEnab* is unaltered. The set *Enab* is re-computed, and things go on this way until *Enab* is empty (note that if *Enab* never gets empty, the executor loops forever, but this means that the ESCP-net is ill-designed). At that point, the executor returns *Fseq*.

C. The Rule-based Enabler

The rule-based enabler computes *CEnab* as a function of the current marking of the ESCP-net, by means of user-supplied rules (different rules are supplied for different ESCP-nets). Rules follow a specified syntax, and are "executed" by the enabler, to determine *CEnab*, according to a specified semantics.

For instance, in our simulator of the Servola-Trieste plant there is a rule of the following form:

marked (MINERAL_POOL, peh;*q*, 1) $\wedge$
q < 21600∗*c_rate*(peh) $\wedge$
¬ ***contains*** (TRAVELING, peh) $\wedge$
¬ ***contains*** (ROADSTEAD1, peh) $\wedge$
…
¬ ***contains*** (ROADSTEAD30, peh) $\wedge$
¬ ***contains*** (MOORING, peh) $\wedge$
¬ ***contains*** (UNLOADING1, peh) $\wedge$
¬ ***contains*** (UNLOADING2, peh) $\wedge$
¬ ***contains*** (UNLOADING3, peh) $\Rightarrow$
    enabled (CALL_SHIP, {*s*:s2}, 1) .

The execution of this rule starts with checking the truth of the antecedent, which is true iff the quantity of peh in the mineral pool is below a threshold corresponding to 15 days (i.e. 21600 minutes) of blast-furnace's working, and no ship carrying peh is traveling, waiting in the roadstead, being moored, or unloading its cargo. If it is true, the consequent causes one occurrence of $\langle$CALL_SHIP, $\beta\rangle$, where β is the binding which just assigns s2 to *s*, to be added to *CEnab*. There are other eleven similar rules, for the other materials. There is also a rule to turn on the blast-furnace if it is off but there are sufficient quantities of all materials:

marked (IDLE, off, 1) $\wedge$
marked (FOSSIL_POOL, cf1;*qcf1*, 1) $\wedge$
…
marked (MINERAL_POOL, f4;*qf4*, 1) $\wedge$
qcf1 ≥ 60∗*c_rate*(cf1) $\wedge$
…
qf4 ≥ 60∗*c_rate*(f4) $\Rightarrow$
    enabled (TURN_ON, {}, 1) .

Finally, there is a rule to turn off the blast-furnace if it is on but some material runs out:

marked (IDLE, ready, 1) $\wedge$
marked (FOSSIL_POOL, cf1;*qcf1*, 1) $\wedge$
…
marked (MINERAL_POOL, f4;*qf4*, 1) $\wedge$

$$(qcf1 < 60 * c\_rate(\mathrm{cf1}) \;\vee$$
$$\ldots$$
$$qf4 < 60 * c\_rate(\mathrm{f4})) \;\Rightarrow$$
$$\textbf{\textit{enabled}}\ (\mathrm{TURN\_OFF}, \{\}, 1)\ .$$

D. The Monitor

The monitor is in charge of performing all the "output" activities of the simulator. These may include logging some meaningful events and the times they happen, computing and logging statistical data, providing animated graphical representations of the working of the plant, and so on. At each call, the monitor produces its output from *Fseq* and from the current marking of the ESCP-net.

For instance, the monitor of our simulator of the Servola-Trieste plant produces an ASCII log like that produced by the existing Fortran simulator. However, the modularity of our architecture allows an easy replacement with an improved monitor which also shows animations of moving ships, heaps varying their size, etc.

E. Implementation of the Architecture

For the implementation of the simulation master, the rule-based enabler, and the monitor, we have employed Gensym G2 [4], a state-of-the-art object-oriented environment for the development and execution of knowledge bases. G2 knowledge bases may in fact contain inference rules, generic procedures, and graphical interfaces (through which animations can be realized with relative ease). We have realized a knowledge base to be used as a starting point for each plant simulator: the simulator developer just has to enhance it by writing the rules, and by implementing the monitor through the extensive and easy-to-use facilities offered by G2.

We have implemented the ESCP-net executor in C++, as a separate process which communicates with the G2 process. We have also implemented a visual editor in C++ by which the simulator developer, through and intuitive graphical interface, can create and edit the ESCP-nets to be executed.

IV. CONCLUSIONS AND FUTURE WORK

The previous section gives evidence of the validity of our architecture for the simulation of real-world plants (even more complex than the Servola-Trieste one), as development times and errors can be dramatically reduced. In fact, while the development of the Fortran simulator had taken weeks, the development of our simulator just took days. Furthermore, ESCP-nets and rules are far more readable and easy to understand than source code listings.

The main advantage in using ESCP-nets instead of CP-nets, lies in their minor complexity, while still allowing much greater convenience and compactness than P-nets in modeling plants. That results in easier implementation (and maintenance), and makes them more amenable to formal analysis.

While currently the rule-based enabler's decisions only depend on the marking of the ESCP-net, more elaborate decisions might be taken by embedding some kind of state (e.g. statistical information about some activities of the plant) into it, and letting decisions also depend on this additional state. This is a direction for future work.

Our current implementation of the ESCP-net visual editor allows the developer to recursively nest ESCP-subnets in macro places. We have planned, for the near future, to allow the creation and editing of parameterized ESCP-subnets, which can then be instantiated with actual parameters and used in larger ESCP-nets or ESCP-subnets. In this way, libraries of ESCP-subnets can be defined and used for modular development. For example, we could define a "buffer" ESCP-subnet parameterized on the dimension, on the type of tokens to be stored, on the buffering policy (FIFO, LIFO, etc.), and so on.

V. ACKNOWLEDGEMENTS

This research has been partially funded by Demag Italimpianti; we thank Stefano Barozzi, Piergiorgio Fontana, Carlo Morelli, and Carlo Timossi for useful discussion. Our colleagues at DIST Paolo Coletta and Marco Magagnini contributed to the implementation of the G2 knowledge base and of the ESCP-net visual editor.

VI. REFERENCES

[1] A. Camurri, A. Coglio, "Extended Simple Colored Petri Nets: A Tool for Plant Simulation", to appear in *Proceedings of the 1997 IEEE International Conference on Systems, Man, and Cybernetics*, Orlando, Florida, USA, October 12–15 1997.

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

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

[4] Gensym Corporation, *G2 Reference Manual* (version 4.x).

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