URL: http://www.elsevier.nl/locate/entcs/volume2.html 15 pages

Concurrent Computing: from Petri Nets to Graph Grammars [1]

Andrea Corradini [a]

[a] *Dipartimento di Informatica, Università di Pisa, I-56125 Pisa, Italy*
andrea@di.unipi.it

Abstract

Petri nets are widely accepted as a specification formalism for concurrent and distributed systems. One of the reasons of their success is the fact that they are equipped with a rich theory, including well-understood concurrent semantics; they also provide an interesting benchmark for tools and techniques for the description of concurrent systems.

Graph grammars can be regarded as a proper generalization of Petri nets, where the current state of a system is described by a graph instead as by a collection of tokens. In this tutorial paper I will review some basic definitions and constructions concerning the concurrent semantics of nets, and I will show to what extent corresponding notions have been developed for graph grammars. Most of such results come out from a joint research by the Berlin and Pisa COMPUGRAPH groups.

1 Introduction

The nets which owe their name to Carl Adam Petri [28,29] have been the first formal tool proposed for the specification of the behaviour of systems which are naturally endowed with a notion of concurrency. The success of Petri nets in the last thirty years can be measured by the looking not only at the uncountably many practical applications of nets, but also at the developments of the theoretical aspects, which range from a complete analysis of the various phenomena arising in simple models of nets to the definition of more expressive (and complex) classes of nets.

Such a success of Petri nets as specification formalism for concurrent or distributed systems is due (among other things) to the fact that they can describe in a natural way the evolution of systems whose states have a distributed nature. In fact, thinking for example to the so-called Place/Transition nets, a state of the system to be specified is represented by a *marking*, i.e., a set of *tokens* distributed among a set of *places*. Thus the state is intrinsically

[1] Research partially supported by the COMPUGRAPH Basic Research Esprit Working Group n. 7183

distributed, and this makes easy the explicit representation of phenomena like *mutual exclusion*, *concurrency*, *sequential composition* and *non-determinism*.

While for sequential, deterministic systems an input/output semantics is often satisfactory, for concurrent or reactive systems (which are intrinsically non-deterministic) such a semantics is usually not sufficient. Indeed, in general one desires a more complete description of the relationships among the elementary steps of a computation, possibly including information about causality, concurrency, or about the points where non-deterministic choices were taken. Such semantics may for example help the understanding of the operational behaviour of a net, can be used to analyze the relationships with other nets or with other systems, or can play an important role in building bigger systems from the composition of elementary ones. As a matter of fact, Petri nets have been equipped along the years with rich, formal computation-based semantics, including both interleaving and truly-concurrent models (see, among others, [30,29]). In many cases such semantics have been defined via well-established categorical techniques, often involving adjunctions between suitable categories [27,25].

Many researchers agree on the claim that graph grammars are more expressive than Petri nets for the specification of concurrent and distributed systems. However, in the classical literature of the area, graph grammars have been considered in most cases as a generalization of string grammars or of term rewriting systems to the rewriting of more complex structures. As a consequence, the many results concerning parallelism and concurrency of the algebraic theory of graph grammars (see [22,23,13]), recast in this more general framework notions and results of (term) rewriting systems, exploring properties like confluence, Church-Rosser, orthogonality of redexes, parallel moves, and so on.

Actually, many different encodings of nets into grammars have been proposed in the literature along the years (see [33] for an overview), and all of them tightly relate some basic concepts of the two formalisms, like *concurrency* of transitions and *parallel independence* of productions. In this tutorial we report mainly about some joint research activities of the Berlin and Pisa COMPUGRAPH groups that go much further in this direction. Starting from a very natural encoding of nets into grammars (where a net is regarded simply as a graph grammar acting on discrete graphs, i.e., labeled sets [5,10]), we will show how many relevant concepts and constructions concerning the concurrent semantics of nets can be extended to grammars. These include graph processes [12,21] that generalize Golz-Reisig processes, event structure semantics for grammars [5,8,32], and a definition of *grammar morphisms*, that is at the basis of the definition of *categories of graph grammars* [3], a concept that looks fundamental for relating grammars, and that, quite surprisingly, has been introduced just recently in literature.

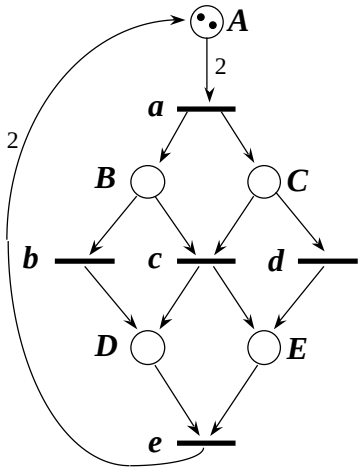

Fig. 1. A safe Place/Transition Petri net

2 Graph Grammars as generalization of Petri nets

As stated in the introduction, Petri nets are widely accepted as an adequate formalism for the specification of concurrent/distributed systems. Indeed, the *state* of a net, i.e., a set of *tokens* distributed among a set of *places*, has an intrinsically distributed nature. As a consequence, nets can specify in a natural way phenomena like *mutual exclusion*, *concurrency*, *sequential composition* and *non-determinism*. Moreover, they have a pleasant graphical presentation, which makes their use appealing also for the non technical user. In this paper we focus on the class of Place/Transition Petri nets and on its subclass of safe nets, presenting them and their semantics in an informal way [29].

The sample net of Figure 1 has a set of *places* $S = \{A, B, C, D, E\}$ (drawn as circles) and a set of *transitions* $T = \{a, b, c, d, e\}$ (represented as thick line segments). Places and transitions are related by a *causal dependency relation* F, which is represented by arrows (for example, $(A, a), (c, D) \in F$, but $(d, D) \notin F$). A natural number near such an arrow indicates its *weight*, as for (A, a) and (e, A); by default an arrow has weight 1. A *marking* M for a net $\mathbf{N}$ is a function $M : S \to \mathbb{N}$. For example, the initial marking $\underline{M}$ of the net in Figure 1 is defined as $\underline{M}(A) = 2$ and $\underline{M}(X) = 0$ for $X \neq A$. Such a marking is represented pictorially by a set of $\underline{M}(X)$ *tokens* (black dots) in each place $X \in S$.

The operational behaviour of a net is described by the so-called "token game". A transition is *enabled* to fire at a given marking if enough tokens are present in all the places that directly cause the transition. The *firing* of an enabled transition removes some tokens from its *preconditions* and creates some new tokens in its *postconditions*, according to the weight function. More transitions can fire simultaneously if each consumes a disjoint set of tokens.

In the sample net of Figure 1 transition a is the unique enabled in the initial marking. Its firing deletes the two tokens in A, and generates a new marking, say M_1, having one token in B and one in C. In marking M_1 there are three enabled transitions: b, c, and d. Moreover the (multi)set $\{b, d\}$ is

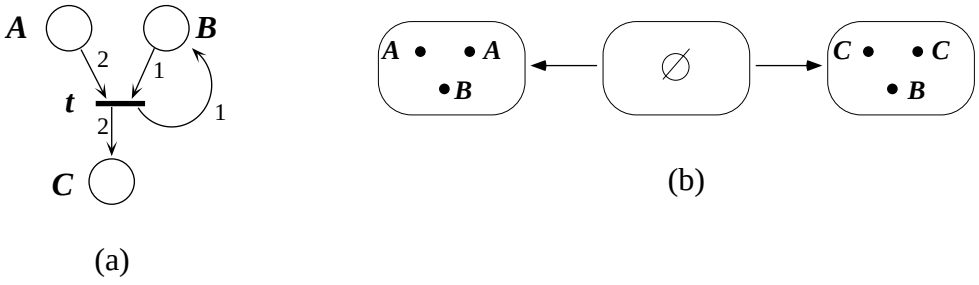

Fig. 2. (a) A transition of a P/T net. (b) The corresponding graph production.

enabled as well, but neither $\{b, c\}$ nor $\{c, d\}$ are (they would need two tokens in B and C, respectively). Thus b and d are *concurrently enabled*, while for example b and c are *in conflict* or *mutually exclusive*: the firing of one of the two prevents the firing of the other. After the firing of either $\{b, d\}$ or c we obtain marking M_2 having one token in D and on in E. Transition e is enabled in M_2, and its firing produces the initial marking. Thus the net has a cyclic behaviour.

Looking at nets from the viewpoint of graph grammars, it is quite natural to regard them as grammars acting on discrete graphs. For example, the transition in Figure 2 (a) is represented in a faithful way by the graph production (i.e., a pair of coinitial graph morphisms, according to the double pushout approach) of Figure 2 (b) (see also [5,10]): Such a production consumes the tokens in the preconditions and generates the tokens in the postconditions of the transition, while the *interface* graph is always empty.

A marking is clearly represented as a set of nodes (the tokens) labeled by the place where they are. In such a representation the topological structure of the net is not represented at all. [2]

It is easy to check that such a representation satisfies all the properties one would expect: a production can be applied to a given marking iff the corresponding transition is enabled; the double pushout construction produces the same marking as the firing of the transition; two occurrences of productions are *parallel independent* in a marking iff the corresponding transitions are concurrent; and so on. For example, I showed in Figure 3 (a) a firing of the transition of Figure 2 (a) from the marking containing three tokens in A, two in B, and one in C, and in Figure 3 (b) the corresponding direct derivation using the production of Figure 2 (b).

This representation of nets by grammars suggests a point of view that has proved to be very fruitful (we intend here graph grammars in the algebraic,

[2] On the contrary, according to some classical encodings of nets into grammars (see [33] and the references therein), the whole net structure (including transitions and places) is represented in the graph, and tokens are represented for example by additional nodes with arcs incident to the corresponding places. In our view such representations have two main drawbacks. First, they are unnecessarily complex, because transitions are represented twice: statically as nodes in the graph representing the net, and dynamically as productions of the grammar simulating the effect of the transition. Second, and more importantly, such representations hide the strong similarities between nets and grammars, i.e., the fact that they are rewriting formalisms acting on different structures: sets and graphs, respectively.

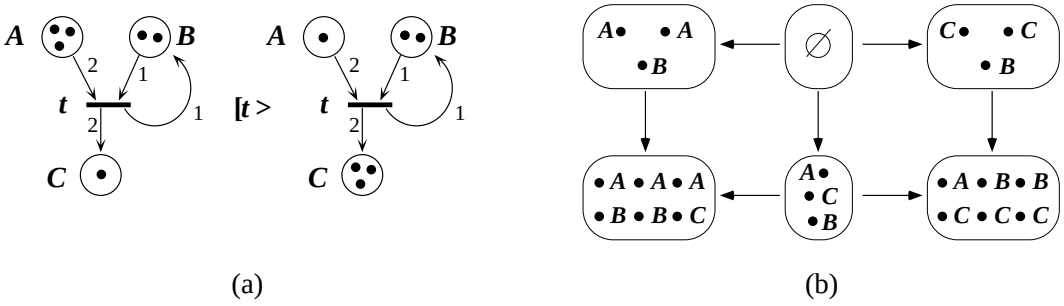

Fig. 3. (a) The firing of a P/T net transition. (b) The corresponding double pushout diagram

double-pushout approach [13], although the same certainly applies to other approaches):

> *A P/T Petri net is a rule-based formalism that rewrites labeled sets (the markings) over a fixed set of labels (the places). Graph grammars generalize P/T Petri nets by replacing labeled sets with labeled graphs, and by allowing a non-empty interface graph in productions.*

A number of recent paper have elaborated this basic idea in various directions. In [26], Montanari and Rossi have proposed *Contextual Nets*, an extension of nets that allows one to have tokens in the interface part of a transitions, or, in their terminology, where a transition can have *context conditions*, i.e., tokens that have to be present for the enabling but that are not consumed. They also showed that the notion of *net process* for such contextual nets is quite more elaborated than for usual nets, and that actually many kinds of processes may be defined. In [20], Korff and Ribeiro generalize the above relationship to *colored* (or *algebraic high-level*) nets and to *attributed* grammars (in the single pushout approach), respectively. Quite interestingly, they show that the construction that transforms a colored net into an attributed grammar commutes both with the *flattening* constructions (that transform colored net and attributed grammars in P/T nets and usual grammars, respectively), and with a semantics based on derivation trees.

This is a further confirmation that the correspondence stated above between nets and grammars is indeed robust. As a matter of fact, a remarkable amount of work has been done in the last years, aiming at generalizing to grammars many definitions, constructions, and results already introduced for nets. In the next sections we will review some of these notions.

2.1 Individuality of tokens and abstractness

There is a subtle mismatch between the transition in part (a) of Figure 2 and our proposed encoding as production shown in part (b). In fact, in the initial marking depicted in Figure 3 (a) the three tokens in place A are indistinguishable (i.e., in P/T nets tokens do not have an *identity*): this is formalized in literature by saying that a marking is a *multiset* over the set of places S [29], or, equivalently, that it is an element of the free commutative monoid over S [24]. As a consequence, there is only one possible firing of the transition t

from the marking depicted.

On the contrary the three nodes labeled by A in the starting graph of the double-pushout of Figure 3 (b) do have a distinguished identity, and indeed there are *twelve* different injective morphisms from the left hand side of the production to that starting graph.

Thus nets are "more abstract" than the corresponding grammar, and in fact there are many grammars that represent the same net. This point arose quite early when the Berlin and Pisa groups started studying a truly concurrent semantics for graph grammars. In fact, it turned out that since graph derivations contain more infomation than firing or step sequences (the corresponding notion for nets), even simple grammars manifest an *infinite-branching non-determinism*, a property not desirable for a formalism, like grammars indeed, acting on *finite* structures through a *finite* number of rules. The problem was addressed in [6,4,7] where we proposed suitable notions of equivalence on graph derivations, able to equate all derivations which are equivalent from a concurrency perspective, and thus reducing the non-determinism of a grammar to a finite degree. Such equivalences were the basis of the event structure semantics of grammars discussed below.

In [20], Korff and Ribeiro put in evidence that analogous differences of abstraction level also hold for the high-level versions of nets and grammars. In fact they show that the derivation tree semantics of a net is isomorphic to some *abstract* derivation tree semantics of the grammar encoding it.

3 Computation-based semantics of Petri nets

For sequential systems it is often sufficient to consider an input/output semantics (thus usually the semantic domain consists of a suitable class of functions). For concurrent/distributed systems, in the semantics one often wants to record more information about the actual computations performed by the system: e.g., one may want to know which steps of a computation are independent (concurrent), or which are causally related. For example, such information is necessary if one wants to compose concurrent systems, keeping the semantics compositional, or if one wants to allocate a computation on a distributed architeture.

There are many computation-based semantics for Petri nets. They differ for the amount of information one wants to record in the semantics, and for the way it is recorded. For example, as nets are intrinsically nondeterministic devices (because of the mutual exclusion phenomenon), the *non-determinism* can be described in two quite different ways: (1) by collecting all the possible net computations in a *set*, and (2) by collecting all the possible computations in a *branching structure* (e.g., a tree), which also records at which points of the computations certain choices have been made.

Orthogonally, the *concurrency* aspects of a net can be represented using a *true concurrency* approach, where the fact that two events are "not causally related" is represented directly in the semantics using a partially ordered structure; or an *interleaving* approach, where computations of the

Non-determinism \ *Concurrency*	**Interleaving**	**True concurrency**
Set-based	(Set of) Firing Sequences	(Set of) Deterministic processes
Branching structure	Tree of firing sequences	Non-deterministic processes
		Event structures

Table 1 Some possible semantic domains for nets

system are sequences of events, and the independence of two events in a computation must be represented by two different sequences where the two events appear in different orders; thus concurrency is reduced to non-determinism.

In Table 1 I indicate some possible semantic domains for nets, for the four possible ways of representing non-determinism and concurrency.

For the net of Figure 1, I depicted in Figure 4 two most informative semantics. Part (a) shows a *non-deterministic process*. A non-deterministic process of a net N is an acyclic net (also called *occurrence net*) without backward conflicts (i.e., each place has at most one ingoing arc), together with a net homomorphism to the original net; in Figure 4 the net homomorphism is indicated by labeling places and transitions of the occurrence net with places and transitions of the net of Figure 1. It is not difficult to construct a non-deterministic process of a given net by "unfolding" it, and by duplicating places when needed to avoid backward conflicts. A non-deterministic process enjoys the property that if we put one token on every "minimal" place (i.e., in the running example on the topmost places labelled by A), then in the resulting marked net every firing sequence individuates uniquely a firing sequence of the original net (through the net homomorphism), and the causal dependencies among transitions are preserved.

Figure 4 (b) shows an event structure for the net of Figure 1. A *(prime) event structure* $\mathcal{E}$ is a triple $\mathcal{E} = \langle E, \leq, \# \rangle$ where: E is a set of *events*; $\leq$ is a partial order relation on E, the *causal dependency relation*, which satisfies the *axiom of finite causes*, i.e., no event can have infinitely many causes; and $\#$ is a binary, symmetric, irreflexive relation on E, the *conflict relation*, which is *hereditary*, i.e., if $e \# e_1$ and $e_1 \leq e_2$, then $e \# e_2$.

In the event structure of Figure 4 (b) the causal dependency relation is represented by its Hasse diagram with directed arcs, and the conflict relation is drawn as undirected arcs labelled by a "#": only conflicts which cannot be inherited are shown. The event structure is easily obtained from the non-deterministic process by deleting all places, letting $t \leq t'$ if a postcondition of t is a precondition of t', letting $t \# t'$ if t and t' have a common precondition (i.e., if they are in conflict), and closing relation $\leq$ under transitivity and symmetry and $\#$ under inheritance.

4 Computation-based semantics for Graph Grammars

A natural question is: How far can the computation-based semantics of nets just sketched be generalized to graph grammars?

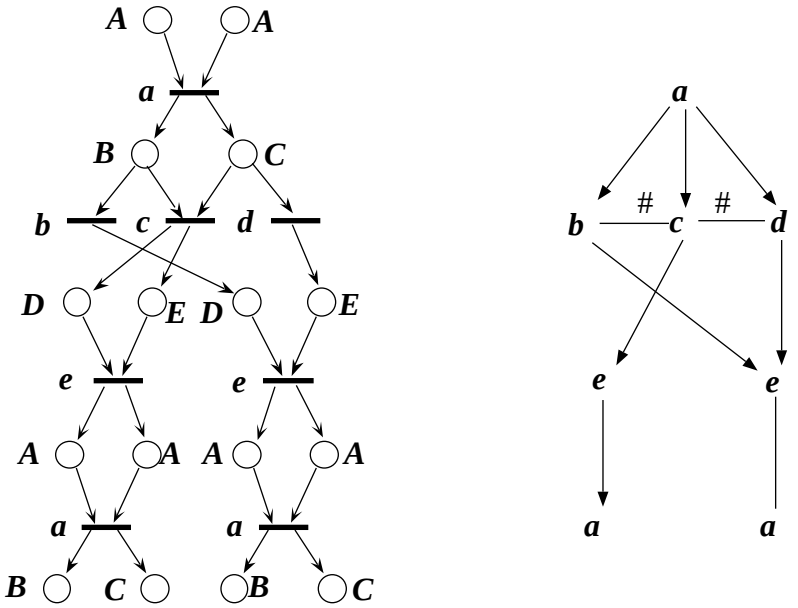

Fig. 4. (a) A non-deterministic process of the net of Figure 1. (b) The corresponding event structure.

The algebraic theory of graph grammars comprises many results concerning parallelism and concurrency (see [22,23,13]). Most of those results recast, in the more general framework of graph rewriting, related notions and results of (term) rewriting systems, exploring properties of confluence, Church-Rosser, orthogonality of redexes, parallel moves, and so on. Thus they are mainly concerned with the study of properties of derivations and of their syntactic manipulations. Such aspects of a graph grammar are surely of great interest. However, in the perspective of presenting graph grammars as a more adequate formalism than nets for the specification of concurrent/distributed systems, also truly-concurrent semantics like those sketched in the previous section for Petri nets would be interesting.

The development of such semantics if the main topic of a research activity carried on by the Berlin and Pisa COMPUGRAPH gropus in the last years. As a matter of fact, the constructions and results about nets cannot be generalized straightforwardly to graph grammars, because they extend Petri nets with some non-trivial features, the most relevant of which is the ability to specify items, in the interface of productions, that have to be present but are not consumed. It turns out that this context-dependent aspect of graph rewriting is even more difficult to be treated formally than the generalization from sets to graphs of the rewritten structures.

In the next subsections we comment on two recently developed computation-based semantics for grammars, namely processes and event strutures.

4.1 Processes for Graph Grammars

A first contribution in the development of a theory of non-sequential processes or graph grammars is [12] where we considered just deterministic processes and *safe* grammars, i.e., grammars where each reachable graph has no non-trivial automorphisms. One of our main goals was to be as close as possible to the corresponding theory for nets: in particular, a graph processes must be a graph grammar as well, exactly like net processes are nets as well, and possibly with some kind of morphism to the original grammar.

We succeded in giving such a definition by slightly changing the classical definition of grammar, introducing the so-called *typed graph grammars*: These are standard grammars where all the involved graphs have a morphism to a fixed *type graph*, which plays the role of the set of places in a net (in other words, all involved graphs belong to the comma category of objects over the type graph). In practice, the type graph can be regarded also as a more structured presentation of the color alphabets for nodes and arcs that are usually part of the definition of labeled graphs.

A *(determistic) graph process* for a given grammar $\mathcal{G}$ is a *strongly safe* grammar (i.e., a safe grammar satisfying suitable aciclicity requirements, similar to those for occurrence nets), equipped with a mapping to $\mathcal{G}$, which is composed of a graph morphism between the type graphs and of a function relating the productions that satisfy suitable commutativity requirements. Such deterministic processes enjoy some interesting properties. First, they can be constructed with a simple colimit construction: Given a graph derivation of grammar $\mathcal{G}$, i.e., a sequence of double-pushout diagrams, the corresponding process is obtained simply by taking as type graph the colimit object in category **Graph** of all the diagram, and as productions all the occurrences of productions of $\mathcal{G}$ that appear in the derivation; such productions have morphisms to the type graph given by the colimit injections.

Second, all the derivations that are *shift-equivalent* (i.e., that differ only for the order in which independent direct derivations are performed [23]) have isomorphic corresponding processes. This suggest that such graph processes are a good candidate as representatives of shift-equivalence classes of derivations, and may be even more adequate than the classical *canonical derivations*, because they have an almost immediate representation of the causalities among production applications.

Other proposals for processes for graph grammars have been elaborated in the recent years. For example, Korff proposes in [19] (and elaborates on this with Ribeiro in [21]) a notion of non-deterministic process (called *concurrent derivation*) for grammars in the single-pushout approach, showing an application to Actor Systems. Also concurrent derivations are obtained as colimits of derivation diagrams, and they can be considered as grammars; in particular, their *core graph* corresponds exactly to our type graphs. Although the similarities of the two approaches are evident, up to now it is not yet clear if they are completely equivalent.

Also Dirk Janssens is recently working on some notion of processes for

the Extended Structure Morphisms systems, which are an evolution of Actor Grammars and are based on the NLC approach to graph rewriting (see his contribution in this volume). His approach is quite different from ours, because ESM systems automatically generate a graphical structure that can be considered as a process. In fact, the application of a rule monotonically augments the graph describing the current state, and therefore at each stage of a computation the current graph includes some representation of the whole history of the derivation.

4.2 Event Structures for Graph Grammars

An event structure semantics has been defined for the subclass of safe graph grammars in [5], and for *consuming* grammars in general in [8] (i.e., those where each production deletes something). An important point is that the construction of the event structure of a grammar we proposed is quite simple, as it does not go through the definition and construction of a non-deterministic process, as shown for nets in Section 3. The construction, at least in the more general case, is based on the equivalence on graph derivations introduced in [4], and on the corresponding construction of the category of abstract derivations of a grammar. In fact, we were able to show that the comma category of the objects under the initial graph in the such a category of abstract derivations is a preorder, and that the induced partial order is a *prime algebraic domain*; thus a prime event structure can be extracted from it, thanks to general results [27].

A closely related paper is [32], where Georg Schied for the first time showed how to construct a prime event structure from an arbitrary, consuming graph grammar. His approach substantially differs from ours for two main reasons, although the results are similar. First, he uses a different technique to get to the event structure, constructing as an intermediate step a *trace language* and then applying general results from [1]. Second, he uses a more set-theoretical definition of graph rewriting, where the result of a direct derivation is determined in a unique way (not up to isomorphisms as in our case).

Besides giving such relevant references to papers concerning event structures for graph grammars, I would like to raise here the following question:

> *How far are event structures adequate as a concurent semantics for graph grammars?*

More precisely, since graph grammars act on graphs and not on sets, as nets do, is *a set of events* sufficient (together with the corresponding causal and conflict relations) for describing the concurrent behaviour of a grammar, as it is for a net? An event structure semantics abstracts completely from the structure of states, as it only shows the causal and conflict relations among the transitions of a system. Thus, since both nets and grammars have a *set* of transitions, it comes of no surprise that, under a certain degree of abstraction, they have a similar semantics. Other kinds of concurrent semantics keep instead the information concerning the state, like the processes recalled in Section 4.1, and thus greatly differ for nets and grammars. This is evident for

example in the definitions of processes in [12,21].

5 From objects to categories

The previous section showed that as far as concurrent semantics constructions and results are concerned, the gap between nets and grammars is being filled up quickly. However, with respect to Petri nets and to other formalisms aimed at describing concurrent and distributed systems (like Event Structures, Asynchronous Transition Systems and others [27,1,24,31]) the classical theory graph grammars still lack a formal categorical, in-the-large treatment. In the works just referred to, the above mentioned formalisms are equipped with a categorical semantics where the leading idea is to define suitable categories of "systems", and to relate them with pairs of adjoint functors. Such semantics (that we call "in-the-large" because the emphasis is on properties of a whole class of systems, as opposed to the "in-the-small" semantics, which study properties of a single system) are useful to understand the relationships between different formalisms, or also to relate different aspects of the same formalism.

As paradigmatical examples of the use of categories in the semantics of nets I cite [35] and [24]. In [35] Winskel shows that the event structure semantics of *safe* nets can be given through a chain of adjunctions starting from the category **Safe** of safe nets, thorugh category **Occ** of occurrence nets (this result has been generalized to arbitrary P/T nets in [31]). In other words, this implies that the construction of the non-deterministic process and of the event structure of a net of Section 3 can be made *functorial*, i.e., consistent with a reasonable notion of net morphism. Also, in [24] Meseguer and Montanari show that there is an adjunction between the category of P/T nets **Petri** and the category of their computational models **CatPetri**. Intuitively, the free model of a net is a small category, equipped with a suitable algebraic structure, where each arrow is a computation of the net.

A natural question arises at this point:

> *What's the point in using category theory to relate systems (e.g., nets) with their semantics (e.g., event structures or categories of computations)? Is it not sufficient to give the explicit construction of the semantics for each given system?*

There are (at least) three good reasons for using categories:

1. When defining a *category* of systems one is forced to provide a notion of morphism, checking that the axioms of categories are satisfied. Often this procedure gives important insights about the structure of systems. For example, the notion of *isomorphism* is derived by that of morphism, and relates system which are "conceptually" the same: all the categorical constructions will handle isomorphic systems in a uniform way.

Moreover, one can check for the existence of some categorical constructions (like products and coproducts, or limits and colimits in general) which should correspond to suitable operations on systems. Performed in a category, such

operations are in general not deterministic (limits and colimits are unique only up to isomorphisms), but this apparent drawback turns out in many situations to be a real simplification. Indeed, thinking for example to operations which glue together (parts of) systems, all the (syntactical) problems related with naming (like α-conversion in logical systems, the "renaming apart" of variables in logic programming, the choice of new names when building the disjoint union of systems) simply disappear in the categorical framework: such operations often correspond to colimit constructions (see also [17]).

2. Once a category of systems and one of "denotations" (semantics) are defined, there are in general many ways (if any) to map the first ones to the others and viceversa. The categorical framework forces you to define these mappings in a consistent way on morphisms as well (because they have to be functors).

3. There are in general many pairs of functors relating two categories. Nevertheless, often (but by no mean always) given two "related" categories (of systems, denotations or whatever) there happen to be an "obvious" functor in one direction (e.g., an inclusion or a "forgetful" functor). Keeping such functor fixed, one can look for functors in the opposite direction forming an adjunction: if such a functor exists it is unique (up to a natural transformation) by general categorical results. The fact that two functors form an adjunction is often regarded as a good argument in favour of the "correctness" and "naturalness" of the relationship established between two categories.

The chains of adjunctions mentioned above are just prototypical examples of a general technique in the categorical semantics of concurrency. Other adjuctions relating categories of systems can be found in [1,25,31].

5.1 Grammar morphisms

Trying to define a categorical semantics for grammars like that for nets just sketched, a necessary precondition is clearly the definiton of a reasonable notion of grammar homomorphism. Such a concept does not appear in the previous literature of the area: A proposals only appeared recently in [3], where we borrowed the idea of grammar morphism from the theory of nets, through a non-trivial elaboration of the formal definitions that has been possible by making yet more precise the relationship between the two formalisms.

In particular, the idea was to regard the definitions of nets and of their morphisms as suitable diagrams in the category **Set**, and to consider exactly the same diagrams, but in the category **Graph**, as the corresponding definitions for grammars. This procedure required the use of typed graph grammars (see Section 4.1) instead of classical grammars. I do not describe here the technical details of the definition of grammar morphisms, but only summarize the on-going work in the categorical semantics for grammars.

The main contribution in this field is [3], where we introduce grammar morphisms and propose a semantics for graph grammars borrowing the general outline from [24]. We define three categories: **GraGra**, having typed graph grammars as objects and grammar morphisms as arrows; **GraTS**, with

(typed) graph transition systems as objects; and **GraCat**, having small categories of (typed) graph derivations as objects. The main result is that there exist left adjoint functors TS : **GraGra** $\rightarrow$ **GraTS** and C : **GraGra** $\rightarrow$ **GraCat** to the forgetful functors U : **GraTS** $\rightarrow$ **GraGra** and V : **GraCat** $\rightarrow$ **GraGra**, respectively.

Grammar morphisms as defined in [3] are also used in [9], where the expressive power of morphisms with respect to structuring and refinement of grammars is explored.

As future developments of there ideas we are trying to make the construction of the event structure functorial, as it is for nets. This both in the direction of extending the adjunction with categories of derivations (thus building on top of [3]) and also through a completely different approach that mimics for grammars Winskel's chain of adjunctions from **Safe** to **ES**.

6 Conclusions

I recalled some interesting concurrent semantics for Petri nets, arguing that similar semantics are also interesting for graph grammars, which are more expressive than nets for the specification of concurrent and distributed systems. I summarized some recent and on-going work that propose for grammars suitable generalizations of relevant definitions, constructions and results of the Petri net theory, including a definition of graph processes, an event structure semantics, the definition of grammar morphisms, and the development of a categorical semantics based on chains of adjunctions.

More importantly, I stressed that Place/Transitions Petri nets and Graph Grammars can be regarded in most situations as the same diagrammatical structures, but in different categories: **Set** and **Graph**, respectively. This is a powerful "meta-result" that should be possible to exploit for the cross-fertilization of the two fields. In fact, for any concept about nets one can try to generalize it to graph grammars simply by isolating the notions based on sets, and by replacing them with the corresponding notions based on graphs. Such a procedure is by no way automatic: the same set-based notion can have many different generalizations to graphs, and the choice of the right one is in general not trivial. However, the categorical framework often narrows the possible choices, making the work easier. Just to mention a possible application of this idea, it would be interesting to explore how the theory of net invariants translates to graph grammars.

Generalizing even further, one may think to nets and grammars as two instantiations of a more abstract theory of concurrent semantics where **Set** and **Graph** are replaced by an arbitrary category **C** satisfying suitable requirements (in the same spirit of High-Level Replacement Systems [14]). However, it is not clear if some relevant constructions can be described in such an abstract framework, neither if there exist other interesting instantiations besides nets and grammars.

References

[1] M.A. Bednarczyk. *Categories of asynchronous systems*. PhD thesis, University of Sussex, 1988. Report no. 1/88.

[2] W. Brauer, W. Reisig, and G. Rozenberg, editors. *Petri Nets: Central Models and Their Properties*, number 254 in LNCS. Springer-Verlag, 1987.

[3] A. Corradini, H. Ehrig, M. Löwe, U. Montanari, and J. Padberg. Typed graph grammars and their adjunction with categories of derivations. Submitted for publication, 1995.

[4] A. Corradini, H. Ehrig, M. Löwe, U. Montanari, and F. Rossi. Abstract graph derivations in the double-pushout approach. In *[34]*, pages 86–103, 1994.

[5] A. Corradini, H. Ehrig, M. Löwe, U. Montanari, and F. Rossi. An event structure semantics for safe graph grammars. In *Programming Concepts, Methods and Calculi - PROCOMET '94*, IFIP Transactions A-56, 1994.

[6] A. Corradini, H. Ehrig, M. Löwe, U. Montanari, and F. Rossi. Note on standard representation of graphs and graph derivations. In *[34]*, pages 104–118, 1994.

[7] A. Corradini, H. Ehrig, M. Löwe, U. Montanari, and F. Rossi. Algebraic approach to graph transformation II: Models of computation in the double pushout approach. Submitted for publication, 1995.

[8] A. Corradini, H. Ehrig, M. Löwe, U. Montanari, and F. Rossi. An event structure semantics for consumptive graph grammars with parallel productions. Submitted for publication, 1995.

[9] A. Corradini and R. Heckel. A compositional approach to structuring and refinement of typed graph grammars. In *[11]*, 1995.

[10] A. Corradini and U. Montanari. Specification of concurrent systems: from Petri nets to graph grammars. In Kleuver, editor, *Proceedings of the Workshop on Quality of Communication Basd Systems*, 1994. To appear.

[11] A. Corradini and U. Montanari, editors. *Proceedings SEGRAGRA'95 Workshop on Graph Rewriting and Computation*, volume 2 of *Electronic Notes in Theoretical Computer Science*. Elsevier Sciences, 1995.
URL: `http://www.elsevier.nl/locate/entces/volume2.html`.

[12] A. Corradini, U. Montanari, and F. Rossi. Graph processes. *Fundamenta Informaticae*, 1995. To appear.

[13] H. Ehrig. Tutorial introduction to the algebraic approach of graph-grammars. In *[15]*, pages 3–14, 1987.

[14] H. Ehrig, A. Habel, H.-J. Kreowski, and F. Parisi-Presicce. Parallelism and concurrency in high-level replacement systems. *Mathematical Structures in Computer Science*, 1:361–404, 1991.

[15] H. Ehrig, M. Nagl, G. Rozenberg, and A. Rosenfeld, editors. *Proceedings of the 3rd International Workshop on Graph-Grammars and Their Application to Computer Science*, number 291 in LNCS. Springer-Verlag, 1987.

[16] G. Engels and G. Rozenberg, editors. *Preliminary Proceedings of the Fifth International Workshop on Graph Grammars and their Application to Computer Science*, 1994.

[17] J.A. Goguen. A categorical manifesto. *Math. Struc. Comput. Sci.*, 1, 1991.

[18] D. Janssens. Process Languages for ESM Systems. In *[11]*, 1995.

[19] M. Korff. True concurrency semantics for single pushout graph transformations with appications to actor systems. In *Proceedings IS-CORE 94 Workshop*, pages 244–258. Vrije Universiteit Press, 1994.

[20] M. Korff and L. Ribeiro. An attributed graph transformation approach to the behaviour of algebraic high-level nets. In *[16]*, pages 113–122, 1994.

[21] M. Korff and L. Ribeiro. Concurrent derivations as single pushout graph grammar processes. In *[11]*, 1995.

[22] H.-J. Kreowski. *Manipulation von Graphmanipulationen.* PhD thesis, Technische Universität Berlin, 1977.

[23] H.-J. Kreowski. Is parallelism already concurrency? part 1: Derivations in graph grammars. In *[15]*, pages 343–360, 1987.

[24] J. Meseguer and U. Montanari. Petri nets are monoids. *Info. and Co.*, 88:105–155, 1990.

[25] J. Meseguer, U. Montanari, and V. Sassone. On the semantics of Petri nets. In *Proceedings CONCUR '9*, number 630 in LNCS, pages 286–301. Springer Verlag, 1992.

[26] M. Montanari and F. Rossi. Contextual nets. Technical Report TR 4-93, Dipartimento di Informatica, University of Pisa, 1993.

[27] M. Nielsen, G. Plotkin, and G. Winskel. Petri nets, event structures and domains, part 1. *Theoret. Comput. Sci.*, 13:85–108, 1981.

[28] C.A. Petri. *Kommunikation mit Automaten.* Schriften des Institutes für Instrumentelle Matematik, Bonn, 1962.

[29] W. Reisig. *Petri Nets: An Introduction.* EACTS Monographs on Theoretical Computer Science. Springer-Verlag, 1985.

[30] G. Rozenberg. Behaviour of elementary net systems. In *[2]*, pages 60–94, 1987.

[31] V. Sassone, M. Nielsen, and G. Winskel. Relationships between models of concurrency. In *Proceedings REX '93*, 1993.

[32] G. Schied. On relating rewriting systems and graph grammars to event structures. In *[34]*, pages 326–340, 1994.

[33] H.-J. Schneider. Graph grammars as a tool to define the behaviour of process systems: From Petri nets to Linda. In *[16]*, 1994.

[34] H.-J. Schneider and H. Ehrig, editors. *Proceedings of the Dagstuhl Seminar 9301 on Graph Transformations in Computer Science*, number 776 in LNCS. Springer Verlag, 1994.

[35] G. Winskel. Event structures. In *Petri Nets: Applications and Relationships to Other Models of Concurrency*, number 255 in LNCS, pages 325–392. Springer Verlag, 1987.