On Graph(ic) Encodings

Roberto Bruni and Ivan Lanese

Dipartimento di Informatica, Università di Pisa, Italia.
{bruni,lanese}@di.unipi.it

Abstract. This paper is an informal summary of different encoding techniques from process calculi and distributed formalisms to graphic frameworks. The survey includes the use of solo diagrams, term graphs, synchronized hyperedge replacement systems, bigraphs, tile models and interactive systems, all presented at the Dagstuhl Seminar 04241. The common theme of all techniques recalled here is having a graphic presentation that, at the same time, gives both an intuitive visual rendering (of processes, states, etc.) and a rigorous mathematical framework.

1 Introduction

Process calculi have been devised as a useful paradigm for the specification and analysis of concurrent, distributed and mobile systems, where processes running on a network are represented as terms of a suitable algebraic theory. Typical ingredients are process signatures and axiomatic structural congruences for the syntax; reduction rules or labeled transitions for the operational semantics, plus a number of observational equivalences and congruences for the abstract semantics; modal and spatial logics for the specification, analysis and verification of processes.

Roughly, process calculi situate in between the realm of mere mathematical abstractions and that of running programming languages, in the attempt to exploit the advantages coming from both worlds: on the one hand, they facilitate rigorous system analysis by focusing on the aspects of interest (e.g. security, distribution, mobility), and on the other hand they may offer the basis for prototypical implementations and for verification tools. For example, on the more theoretical side, process calculi favor the application of standard algebraic and coalgebraic techniques like definition of initial and final models, universal constructions and observational congruences, while on the more practical side, there are now many running implementations of languages based on calculi that were originally proposed to study and to experiment with basic primitives, like name passing (π-calculus [43]), process passing (HOπ-calculus [50]), cryptographic messaging (spi-calculus [1]), distributed dataspaces (KLAIM [13]), mobile environments (ambient calculus [9]), just to cite a few. Furthermore, process calculi allow to evaluate the expressiveness and usability of different mechanisms and applications before implementing them, hence reducing costs and efforts in the (re)design phase and increasing the robustness of the final product.

The need of graphic representations for processes emerged at the very beginning of process algebra developments (take as a prominent example the CCS agent diagrams in [40]). The point was that many important aspects like network topology, connectivity, distribution and many others are not so evident when looking at the terms associated to processes, because these aspects are often encoded by means of ad-hoc name disciplines with all related issues (name sharing, binders, scoping, α-conversion, renaming, substitutions). The situation is even worst when the above aspects can change dynamically upon communication (dynamic reconfiguration, name extrusion, isolated names). All these aspects are more amenable to visual descriptions than to syntax-driven encodings and therefore they can be conveyed and analyzed in a more intuitive way by giving suitable graphic representations to processes and delegating the dynamics to (different flavors of) graph transformation systems (GTSs).

Likewise process calculi, also GTSs come equipped with a rigorous theory that can be used to complement, to some extent, that of process calculi, so that the advantages of one approach can be transferred to the other and that certain deficiencies can be overcome by the combined use of both approaches. For example, we have mentioned above that process terms are easy to manipulate but too syntax-dependent for expressing topology, distribution and connectivity in a natural, intuitive way. On the other hand, graph representations make evident the way in which entities are distributed and connected over the network, but can complicate the reading of the computation flow and the behavioral equivalence proofs. An example of the synergic use of process calculi and graphs is the definition of concurrent abstract semantics for process calculi by means of GTS encodings.

GTSs exploit only one “graphic dimension” for drawing graphs that model system configurations (the dynamics consists of *silent* reductions that replace subgraphs). In some cases, it can be convenient to draw graphs in more than one dimension. For example, different graph structures can be used for separating different structures coexisting in the same system, like the physical structure of locations (spatial distribution) and the logic structure of known channels (communication network). As another example, one may consider an enhancement of GTSs where graph transformations are labeled with located actions or causality links to past events: in this case the actions observed during computations form themselves graphs that span across time and therefore they are orthogonal w.r.t. the graphs that model system configurations (which span across space).

Several interesting presentations on graphic encodings of process calculi and distributed systems were held at the Dagstuhl Seminar 04241. The talks often concerned different models and techniques, exploited with different aims in mind. In the next sections we try to account for the distinguishing features of each approach and to place them in the right context with respect to the related literature. Although each approach would deserve a whole section (and several pages) by itself, we have preferred a more concise exposition, where some approaches are collected together and discussed informally on the basis of their key features.

For the sake of presentation, we partition all techniques into two classes, depending on the number of “graphic dimensions” in which graphs are drawn (in the sense sketched above). Although this grouping discipline may appear rather arbitrary, it has been devised to expose in a structured manner the main analogies and divergences between different approaches.

2 One-dimensional graphic encodings

Our survey starts by focusing on three proposals that exploit graph rewriting techniques for simulating reductions in process calculi. Roughly, they fall in a wider research thread also inspired by the so-called optimal implementation of λ-calculus [38,31] (see [2] for a survey) , and as such they follow the same *pattern*:

1. first a mobile process calculus is selected and its processes are mapped to graphs of a suitable kind;
2. second, the mapping is proved sound and complete w.r.t. the structural congruence on processes (in the sense that two processes are structurally equivalent if and only if they correspond to the same graph);
3. finally, graph rewriting techniques are used to simulate process reductions.

When the whole procedure can be carried out, there are two main achievements. The first is that the graph encoding allows one to get rid of all the issues related to naming (like α-conversion of bound names) and to other syntactic issues (like associativity, commutativity and unit of parallel composition). The second is that not only graph rewrites can simulate the interleaving reductions of the original operational semantics, but they define a straightforward concurrent semantics by making explicit the places where such reductions occur in the process-as-graph (for example, it is immediate that non-overlapping rewrites can be executed concurrently).

In the following we contextualize and compare the approaches presented in the talks by Björn Victor, by Fabio Gadducci, and by the second author. As it will be immediately noticed, all talks focused on different process calculi and exploit different encoding techniques, yet they share many similarities.

2.1 On solo diagrams

The remarkable success and diffusion of the π-calculus led to the definition of many variants with the combined aim of retaining as much as the expressiveness of the π-calculus while favoring practical implementation in distributed platforms.

The *fusion calculus* [46] is at the same time a simplification and an extension of π-calculus: the fusion calculus has only one binding operator (the restriction, but not the input prefix), because input and output prefixes are completely symmetric (unlike π-calculus), yet the effects of synchronization are not just local to the receiver and they consist of name *fusions* rather than substitutions. The combination of these features makes it possible to encode the π-calculus as a proper subcalculus of the fusion calculus (roughly, the ν binder becomes restriction, input prefixes $x(y).P$ of the π-calculus are translated as $(y)xy.[\![P]\!]$, where the object name y is bound by restriction, and all the remaining operators are translated trivially). On the other hand, the binding mechanism of the fusion calculus ignores the issue of unicity of newly generated names, so that, conceptually speaking, the ν operator of the π-calculus and the restriction operator of the fusion calculus are very different in spirit.

The *solos calculus* [35] takes inspiration from both the fusion calculus and the *asynchronous π-calculus* [27] (an easier-to-implement simplification of the π-calculus where continuations are banned from output prefixes, so that only output particles are allowed): it is obtained from the fusion calculus by removing continuations after all action prefixes, hence only input / output particles are allowed (the so-called *solos*). The solos calculus maintains the same expressive power of the whole fusion calculus, whereas this is not the case for the asynchronous π-calculus versus the π-calculus. (Actually, the polyadic solos calculus can be encoded into the dyadic one, where solos carry at most two names.) In the solos calculus, recursion is coded by means of replication $!P$ and unguarded unfolding is dealt with additional reduction rules that induce a behavioral flattening law, according to which nested replications can be safely removed. (This is necessary because the absence of action prefixes prevents the use of guarded variants of replication.)

Solo diagrams were introduced in [34] to define an efficient implementation of the solos calculus, adapting ideas from interaction diagrams [45] and pi-nets [41]. Solo diagrams are essentially boxed hypergraphs over an infinite set $\mathcal{U}$ of nodes with two kinds of hyperedges: *input edges* $\langle a, a_1, \ldots, a_k \rangle_{\mathsf{i}}$ and *output edges* $\langle a, a_1, \ldots, a_k \rangle_{\mathsf{o}}$, where in both cases a is the subject node and $a_1, \ldots, a_k$ are the object nodes. A *box* is just a pair consisting of a finite multiset of edges G and a distinguished subset S of internal nodes of G, which are local to that box. Multisets of edges and boxes form solo diagrams.

The graphic representation of agents can be sketched by picking one node for each different name, then drawing output solos $\overline{u}\widetilde{x}$ as output edges $\langle u, x_1, \ldots, x_k \rangle_{\mathsf{o}}$ from the nodes labeled $\widetilde{x}$ to the node with label u and, respectively, input solos $u\widetilde{x}$ as input edges $\langle u, x_1, \ldots, x_k \rangle_{\mathsf{i}}$ from the node labeled u to the nodes labeled $\widetilde{x}$. Parallel composition becomes graph union (where nodes labeled with the same name are identified and thus shared). Scope restriction (x) erases the label from node x, so that it becomes anonymous and cannot be further shared when composing in parallel. Replication is modeled by boxing the graph to be replicated.

As the reader can expect, diagram isomorphism coincides with the usual structural congruence of solos (namely, the abelian monoid laws for parallel composition, α-renaming of bound names, and the usual laws for scoping).

Solo diagrams are given four rewrite schemes in set-theoretical style, to take into account reductions inside, outside or across boxes. The basic reductions involve an output edge and an input edge with the same subject. As a result the two edges disappear and the object nodes are fused pairwise (unless they have different labels, in which case the reduction cannot take place). If boxes are involved, first a fresh copy of the content of the box is created and then the reduction takes place.

The talk by Björn Victor illustrated how solo diagrams can be extended so to encode a fully asynchronous flavor of the *D-Fusion calculus* [4]. The D-Fusion calculus is itself a variant of the fusion calculus where the π-calculus binder ν for fresh name generation is (re)introduced to prevent unification of newly generated names, which in fact should always kept distinct according to the discipline of the π-calculus. The D-Fusion calculus contains both the fusion calculus and the π-calculus as proper subcalculi. Moreover, the combined use of the two binders in the D-Fusion calculus allows to express an interesting form of pattern matching (that can be expressed neither in the π-calculus, nor in the fusion calculus) and to encode mixed guarded choice.

In the same way as the solos calculus is derived from the fusion calculus, the *D-solos calculus* is straightforwardly obtained from the D-Fusion calculus by removing continuations after action prefixes. The encoding of D-solos processes in D-solo diagrams is defined along the same line of the encoding of solos processes. However, in this case, solo diagrams must first be extended with a coloring of nodes (to take into account the different binders) and by slightly modifying the graph rewrite rule schemes. Then, the resulting D-solo diagrams are shown to encode the processes of D-solos calculus in such a way that diagram isomorphism captures the structural equivalence on processes and that reductions on diagrams are in bijective correspondence with reductions on processes.

2.2 On term graph rewriting

The solo diagram approach was mainly motivated by the search for efficient implementations (of the fusion calculus). To this aim, the applied graph rewrite techniques were designed ad hoc and tailored to the case study. In particular, concurrency issues were not addressed by the set-theoretical reduction schemes that defined the (interleaving) semantics of solo diagrams. Other graphic approaches address concurrency as a primary issue and exploit the standard theory of graph transformation systems to derive concurrent semantics. Concurrent semantics can serve both to fix an upper limit to the amount of parallelism in the system and to establish causal dependencies between reduction steps, two aspects that have important consequences at the level of specification, design and implementation of distributed applications.

The talk by Fabio Gadducci illustrated a methodology for encoding calculi with name passing in a flavor of hypergraph rewriting. In this case, the encoding is designed so to exploit well-consolidated techniques and results. In particular, the encoding is based on term graph rewriting systems [48] for which a well-developed concurrent semantics is available [3]. The cuncurrent semantics for GTSs generalizes the standard concurrent (and causal) semantics of Petri nets equipping graph transformation systems with the appropriate notions of graph processes, event structures and prime algebraic domains. The full generality of the methodology is sustained by its application to two of the most popular nominal calculi, namely the π-calculus and the (communication-free fragment of the) *ambient calculus* [9]. We do not discuss the details of the two implementations (that can be found in [18,20]), but rather we try to compare the methodology against analogous approaches in the literature, privileging solo diagrams in the discussion.

Informally, term graphs are labeled dags representing terms where the same occurrence of a subterm can be shared explicitly in many positions. The algebraic structure of (ranked) term graphs is explained in [11] and it includes parallel and sequential compositions. Nodes in the graphs can have different sorts. Typically they can represent names, or connection points for sequential processes, or connection points for parallel processes. Unlike solo diagrams, the (typed) hyperarcs needed in the encodings are strongly influenced by the signature of the process calculus under consideration. Typically an additional unary arc with a special label is introduced, whose instances are then attached to any active process node to enable reductions (it is especially useful when modeling calculi with non-reactive contexts, like action prefixes of π-calculus). Like solo diagrams, the encodings based on term graphs can be designed in such a way that two processes are mapped to isomorphic term graphs if and only if they are structurally equivalent. However, unlike solo diagrams, it is often the case that the encoding is not surjective, i.e., there can exist term graphs that are not images of any process. Moreover, the usual axiom $\nu x.0 = 0$ should not be part of the structural congruence, because it would complicate the encoding too much with additional rules for garbage collecting unused names. Fortunately this is not a big issue, because then the two processes are trivially behavioral equivalent anyway. Another difference is that encodings are given to nominal calculi that have prefixes (and therefore non-reactive contexts) and that contain recursive process definitions (instead of replication).

The dynamics of the term graph encodings is then expressed by few graph productions in the double-pushout (DPO) style. Roughly they are (labeled) spans of graph morphisms $(d_L \xleftarrow{l} d_K \xrightarrow{r} d_R)$. Operationally, productions are applied to a larger graph G by first finding a match $m_L : d_L \to G$ that fixes an occurrence $m_L(d_L)$ of d_L in G, then removing all objects of G that are matched by $d_L \setminus l(d_K)$, and finally adding fresh occurrences of the objects in $d_R \setminus r(d_K)$. Note that the rewrite preserves as-they-are all the objects of G matching the middle interface subgraph d_K, which are also used to connect the fresh elements in a proper way to the remainder of G. These are the elements read but not consumed by the derivation. Their role is particularly relevant from the point of view of concurrency, because they can be read simultaneously by many concurrent rewrites. A slight generalization of graph processes from [3] can then be used to characterize exactly concurrent computations, identifying those derivations that differ only for the scheduling of independent steps. With respect to solo diagrams, the DPO semantics offer a standard concurrent semantics instead of an ad hoc interleaving semantics.

Finally, it is worth remarking that unlike other graphic encodings of nominal calculi (e.g., [41,45,24,22,44,30]), both the term graph approach and the solo diagram approach exploit non-hierarchical graphs (another exception is [54]).

2.3 On synchronized hyperdge replacement

The talk by the second author considered a mapping from fusion calculus into *synchronized hyperedge replacement* (SHR), a modular approach to (flat) hypergraph rewriting. A detailed presentation of the mapping can be found in [32]. The term graph approach coded the structure of the process term in the graph, while the SHR approach maps in the graph the structure determined by the sharing of names among concurrent fusion calculus processes, as it is done in the solo diagram approach. However fusion processes can be arbitrarily complex (not just solos), and the additional structure is stored inside hyperedge labels.

SHR [14,25,16] has been developed to model dynamic reconfigurations of distributed systems. In particular processes or system components are modeled as hyperedges, and communication channels between them as shared nodes. Graph rewrite rules are derived by combining *productions*, i.e., basic rules that replace a single hyperedge h with a generic graph, preserving all the nodes attached to h (but some of them may be merged), which act as interface. Moreover observable effects are produced on the nodes of the interface: for each node the corresponding effect contains an action and a tuple of references to nodes. Having a single hyperedge in the left-hand side guarantees that rewrites are local (and thus easier to implement also in a distributed environment [14]).

A set of productions can be applied in one step only if the multiset of actions performed on shared nodes is compatible. What exactly compatible means is defined by a suitable *synchronization algebra* [33] that defines the composition of actions (the result is undefined if the actions are not compatible). Furthermore, when actions are combined, the referenced nodes can be merged and communicated via the resulting action, according to a *mobility pattern*, which is part of the synchronization algebra. Finally nodes can be both free (i.e. part of the interface) or restricted, and in that case some actions may be forbidden on them.

SHR is traditionally equipped with a representation of graphs as syntactic judgements [26], where nodes are names, restricted nodes are bound names (and thus are α-convertible) and an edge with label L attached to nodes $x_1,\ldots,x_n$ becomes a term $L(x_1,\ldots,x_n)$. Thus the semantics can be expressed as in usual process calculi via an LTS. The resulting abstract semantics is a congruence w.r.t. the operators of an underlying algebra of graphs.

The instance of SHR employed for encoding fusion calculus exploits the so called Milner synchronization [16] as synchronization mechanism and a set of mobility patterns that models message passing. The aim of the work was to show that SHR can be considered as a generalization of fusion calculus and that SHR is a good execution framework for fusion processes. The first part is proved by the fact that the mapping is very simple (both parallel composition and restriction are mapped omomorphically). The main point of the translation is the use of labels of hyperedges to describe schemas of sequential processes (i.e. sequential processes that have distinct standardized names). Thus the labels of the hyperedges, which take into account the set of topmost action prefixes, define which productions are available and thus which are the allowed behaviours (we have a production for each enabled prefix), while the name handling part is done by the corresponding mechanisms of SHR, which thus prove to be a generalization of the fusion calculus ones. As usual graphs up to isomorphisms correspond to processes up to structural congruence.

Leaving aside that SHR provides a graphical representation of the topological structure of processes, a main advantage of the encoding is that a concurrent semantics is naturally defined (with the meaning that many independent transitions can be executed in one step). Furthermore alternative semantics for fusion processes can be easily defined by changing the synchronization algebra, thus allowing to model systems based on different communication primitives, such as broadcast, and all these semantics are compositional w.r.t. the operations of parallel composition, name restriction and name fusion.

3 Two-dimensional graphic encodings

Sometimes drawing graphs in “one dimension” is not enough to convey the conceptual separation between certain aspects that are consequently mixed together during the encoding. When this happens, there is a price to pay in terms of increased complexity in the specification, analysis and verification over the graphic encoding, even if the encoding itself may become easier to define. The fact is that the separation of concerns removed during the encoding must then be recovered or at least approximated at the level of the analysis techniques.

As we have mentioned in the introduction, examples range from orthogonal graph structures that describe different aspects of a configuration (like spatial distribution and communication infrastructure) to graphs that model both configurations and some observable effects of computations (i.e., both static and dynamic aspects).

A possible option is to preserve the separation of concerns by developing models and techniques that exploit additional graphic dimensions. Approaches of this kind are feasible as long as they are applicable without complicating the encoding too much.

In the following we survey the talks by Robin Milner, by the first author and by Gheroghr Stefanescu that exploit two graphic dimensions for the visual rendering of the encoding graphs (as well as for their mathematical understanding).

3.1 On bigraphs

Bigraphs [42,28,29] are a general model aimed at describing systems by considering at the same time two orthogonal issues, namely their spatial distribution and their communication infrastructure. In fact, the slogan of bigraphs is "Where you are does not determines whom you can talk to." Thus bigraphs integrate two graph structures, a tree called *place graph* that describes the nesting of locations and a *link graph* that shows how located agents are connected. Notably the two structures are defined over the same set of nodes. The visual representation shows both structures at once, with nodes represented by ovals which are nested according to the spatial containment relation and which are connected by edges according to the link structure. On the contrary, the mathematical treatment of bigraphs is eased by considering the two structures separately. More precisely, bigraphs are arrows in a precategory whose objects are interfaces (for both the place and the link structures) that can be used to compose small bigraphs to build larger ones. An extension of bigraphs allows also to model restriction, specifying that some links are local to a place.

The dynamics of bigraphs is defined by silent rewrite rules, which can be instantiated and applied to the starting bigraph. An important aim for bigraphs is to develop an LTS from the reduction rules [51], using as labels the minimal contexts that enable a reduction. General results prove that the resulting abstract semantics is a congruence. Suitable categorical constructions are provided both for applying rewrite rules and for selecting a minimal set of labels (through the so-called IPO construction [37]) from which all others can be derived.

An encoding is provided from a fragment of asynchronous π-calculus [27] without summation and recursion (similar encodings exist also for ambient calculus [9] and λ-calculus) into bigraphs in order to show that the separation of the distribution structure from the communication structure allows to easily model the kind of systems arising in the global computing setting. The place graph is used to model the structure of the term while the link graph is used for name handling. An interesting point is that places can be either active or passive. The difference is that passive places forbid any inside reduction, and thus they allow, e.g., to forbid reductions under action prefix in π-calculus (while in other approaches, such as the term graph one, one has to manage special edges representing the capability to perform reductions). The isomorphism relation on graphs corresponds to standard structural congruence (extended with the axiom $\nu z.x(y).P = x(y).\nu z.P$ iff $z \notin \{x,y\}$, which is correct w.r.t. the standard semantics).

The general theory of bigraphs allows to derive an LTS from the encoding in a standard way. In the case of asynchronous π-calculus the bisimulation equivalence over the derived LTS coincides with standard barbed congruence [29].

3.2 On tile systems

Both graphic dimensions exploited by bigraphs address configuration features. The SHR approach already evidenced that local graph rewrites can produce complex effects on the interface nodes, including names and links to other nodes. The tile approach [21] builds on this aspect by offering a framework where graphs are not just used for describing configurations, but also for observations. The orthogonality between the two dimensions is emphasized by drawing configuration graphs horizontally and observation graphs vertically. Since in general components have two interfaces (the input one and the output one), each configuration has two corresponding distinguished sets of nodes where actions can be observed. Tile rewrite rules can be depicted as “squares” (whence the word *tile*) whose sides are labeled by graphs: the top graph is the initial configuration to be rewritten; the left graph is the observable projection of the rewrite on the input interface and similarly for the right graph w.r.t. the output interface; the bottom graph is the final configuration, which replaces the initial one after the rewrite. Adjacent graphs have a common interface (one vertex), but are otherwise disjoint. Noticeably, configuration graphs (respectively observation graphs) can be composed sequentially and in parallel. In fact the two kinds of graphs form the arrows of two monoidal categories over the same set of objects (the interfaces). By varying the algebraic structure of configurations and observations, tiles can model many different aspects of dynamic systems, ranging, e.g., from concurrency and causalilty aspects for located and name-passing calculi [15], to elegant handling mechanisms for names abstraction, name generation and higher order structures [8,7].

The tile model [19] takes inspiration from and bears many analogies with various other formalisms, including SOS [47], *context systems* [36], *conditional transition systems* [49], *structured transition systems* [12], and *rewriting logic* [39]. In general, tile configurations and observations are not necessarily graphs, as they can have complex algebraic structures. For example, term graphs can be used, in which case both the algebraic and the graph view are reconciled thanks to [11]. Tile logic [5] extends rewriting logic with a built-in mechanism, based on observable effects, for coordinating local rewrites. Likewise SHR, the dynamics of a system is a coordinated evolution of its local subconfigurations. In fact, tiles can be composed horizontally, vertically, and in parallel to generate larger steps. Horizontal composition yields rewrite synchronization. Vertical composition models the sequential composition of computations. Parallel composition corresponds to building concurrent steps. Thus, tile systems allow to define models that are compositional both in “space” (i.e., according to the structure of the system) and in “time” (i.e., according to the computation flow). Operationally, tiles can be seen as transitions labeled with the (pairs of) observations on input / output interfaces produced during the rewrites. This interpretation gives rise to abstract semantics that generalize those based on ordinary LTSs, like tile trace equivalence and tile bisimilarity.

The talk by the first author presented the tile encoding of *CommUnity* [17]. CommUnity is not exactly a process calculus, but rather a (parallel) program design language in the style of *Unity* [10]. Unlike Unity, CommUnity is based on action sharing and it was initially proposed to show how programs fit into Goguen's categorical approach to General Systems Theory [23]. Since then, it has evolved into an architectural description language, capitalizing on the fact that CommUnity provides a conceptual distinction between between "computation" and "coordination" concerns in communicating distributed system. The individual components (called *designs*) of a CommUnity system can be defined in terms of (input / output) *channels* and *actions*. Input channels are read-only and are controlled by the environment while output channels are controlled locally by the component. Actions represent possible interactions between the component and the environment and consist of multiple assignments guarded by an enabling condition. The interaction between designs is based on action synchronization and interconnection of input and output channels. Design composition is specified by giving diagrams in a suitable category whose objects are designs. The colimit construction on such diagrams returns a monolithic, unstructured design representing the system as a whole.

The tile encoding exploits a novel decomposition of CommUnity diagrams in terms of elementary designs. The resulting tile system gives both an operational semantics and an abstract semantics which is correct w.r.t. the colimit construction, in the sense that the translation of a diagram is tile bisimilar to the translation of its colimit [6]. The tile encoding introduces ad hoc connectors for synchronization, hiding and mutual exclusion, which are used to coordinate elementary components. Roughly, such connectors are special arcs of configuration graphs to which are delegated coordination activities (but they perform no computational activity). A stronger relation between the colimit construction and the abstract semantics is established by showing that the encoding of a CommUnity diagram is *equal* to the encoding of its colimit up to a suitable axiomatization of the connectors. The results provide a nice integration of graph encodings (configuration graphs modeling CommUnity diagrams) with algebraic techniques (the axiomatization of connectors) and co-algebraic techniques (tile bisimilarity).

3.3 A word on interactive systems

We conclude our excursus around graphic approaches by mentioning the talk by Gheorghe Stefanescu on *interactive systems*. Although interactive systems are not really graph-based, they offer a graphic formalism that exploits space and time dimensions in a way which is very similar to tile systems.

Interactive systems come in a number of flavors. The simpler case of *finite interactive systems* (FISs) [52] consists of two-dimensional automata expressing at the same time state-transformations and process interactions. Roughly, FISs correspond to a special kind of tile systems where both configurations and observation are strings. Their main applications are as both specification and operational framework for concurrent object-oriented agents and as recognition devices for planar words (i.e., grids of symbols on a given alphabet) in connection with two dimensional regular expressions (that offer an algebraic view of FISs).

The more general case of *interactive systems with registers and voices* (RV-FISs) accomodates for a large class of open, interactive computing systems with object-oriented features. RV-FISs include register machines and full space-time duality, introducing the concept of *voices* (the time dual of a register). FISs define space-invariant models which are highly compositional and offer the basis for a programming language that uses novel techniques for syntax and semantics. We refer the interested reader to [53] for a detailed survey.

4 Conclusion

It is a common understanding that in the same way as the λ-calculus is the foundational calculus for functional and sequential programming, the π-calculus is the foundational calculus for distributed and mobile programming. We think that the most evident proof of the importance of graphic encodings for process calculi is the fact that passing from standard interleaving semantics for the π-calculus to concurrent and distributed ones is really cumbersome unless graphic encodings are exploited. More generally, an important aspect of graphic encodings is their use as meta-framework for which results are proved once and then applied to many instances. For example, concurrent semantics are obtained by applying the general theory of GTS to the particular encoding under consideration. Other examples are congruence results for observational equivalences in the context of tile systems, SHR and bigraphs, or the derivation of LTS from reduction systems in the context of bigraphs.

Table 1 collects in an ordered manner the characteristics of each graphic encoding with respect to certain features. Interactive systems have been left out of the table because they have a different flavor with respect to all other approaches (graphic but not graph-based). The first column tells whether the graphic formalisms allow or not configurations which are not the image of any process. The second and third columns deal with syntactic features of the calculi that can be encoded. They tell which kind of recursion can be more easily encoded (either the replication operator or recursive process definitions) and if non-reactive contexts can be modeled or not. The fourth and fifth columns have semantic flavors. They tell if the resulting operational semantics exploit silent transitions or labeled ones and if concurrent abstract semantics can be easily derived or not. (In the case of bigraphs, we put “silent” in the cell associated to the kind of transitions, because dynamics is usually expressed by reduction, even if suitable LTS can then be derived exploiting the IPO approach.)

To conclude we remind that although the integration of different techniques from the various calculi and their encodings can lead to undoubtful advancements both in theory and in practice, there is still plenty of work to do, because such integration has just started and it has proved to constitute a fertile and challenging field of research for the upcoming years.

	surjective enc.	recursion	non reactive ctxs	transitions	semantics
solo diagrams	yes	replication	avoided	silent	interleaving
term graphs	no	recursive defs	allowed	silent	concurrent
SHR systems	no	recursive defs	allowed	labeled	concurrent
bigraphs	no	-	allowed	silent	interleaving
tile systems	no	recursive defs	allowed	labeled	concurrent

Table 1. Schematic comparison of encodings on specific features.

References

1. M. Abadi and A.D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Inform. and Comput.*, 148(1):1–70, 1999.
2. A. Asperti and S. Guerrini. *The Optimal Implementation of Functional Programming Languages*, volume 45 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1998.
3. P. Baldan, A. Corradini, H. Ehrig, M. Löwe, U. Montanari, and F. Rossi. Concurrent semantics of algebraic graph transformation. In H. Ehrig, H.-J. Kreowski, U. Montanari, and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 3, pages 107–187. World Scientific, 1999.
4. M. Boreale, M. Buscemi, and U. Montanari. D-fusion: a distinctive fusion calculus. In W.-N. Chin, editor, *Proceedings of APLAS 2004, 2nd Asian Symposium on Programming Languages and Systems*, volume 3302 of *Lect. Notes in Comput. Sci.*, pages 296–310. Springer Verlag, 2004.
5. R. Bruni. *Tile Logic for Synchronized Rewriting of Concurrent Systems*. PhD thesis, Computer Science Department, University of Pisa, 1999.
6. R. Bruni, J.L. Fiadeiro, I. Lanese, A. Lopes, and U. Montanari. New insights on architectural connectors. In J.-J. Levy, E.W. Mayr, and J.C. Mitchell, editors, *Proceedings of IFIP TCS 2004, 3rd IFIP International Conference on Theoretical Computer Science*, IFIP Conference Proceedings, pages 367–379. Kluwer Academics, 2004.
7. R. Bruni, F. Honsell, M. Lenisa, and M. Miculan. Comparing higher-order encodings in logical frameworks and tile logic. In U. Montanari, editor, *Proceedings of TOSCA 2001, Final Workshop of the TOSCA Project*, volume 62 of *Elect. Notes in Th. Comput. Sci.*, 2001.
8. R. Bruni and U. Montanari. Cartesian closed double categories, their lambda-notation, and the pi-calculus. In *Proceedings of LICS'99, 14th Annual IEEE Symposium on Logic in Computer Science*, pages 246–265. IEEE Computer Society Press, 1999.
9. L. Cardelli and A.D. Gordon. Mobile ambients. In M. Nivat, editor, *Proceedings of FoSSaCS'98, Foundations of Software Science and Computational Structures*, volume 1378 of *Lect. Notes in Comput. Sci.*, pages 140–155. Springer Verlag, 1998.
10. K. Chandy and J. Misra. *Parallel Program Design - A Foundation*. Addison-Wesley, 1988.
11. A. Corradini and F. Gadducci. An algebraic presentation of term graphs, via gs-monoidal categories. *Applied Categorical Structures*, 7:299–331, 1999.
12. A. Corradini and U. Montanari. An algebraic semantics for structured transition systems and its application to logic programs. *Theoret. Comput. Sci.*, 103:51–106, 1992.
13. R. De Nicola, G.L. Ferrari, and R. Pugliese. KLAIM: a kernel language for agents interaction and mobility. *IEEE Transactions on Software Engineering*, 24(5):315–330, 1998.
14. P. Degano and U. Montanari. A model for distributed systems based on graph rewriting. *Journal of the ACM (JACM)*, 34(2):411–449, 1987.

15. G.L. Ferrari and U. Montanari. Tile formats for located and mobile systems. *Inform. and Comput.*, 156(1-2):173–235, 2000.
16. G.L. Ferrari, U. Montanari, and E. Tuosto. A LTS semantics of ambients via graph synchronization with mobility. In A. Restivo, S. Ronchi Della Rocca, and L. Roversi, editors, *Proceedings of ICTCS 2001, 7th Italian Conference on Theoretical Computer Science*, volume 2202 of *Lect. Notes in Comput. Sci.*, pages 1–16. Springer Verlag, 2001.
17. J.L. Fiadeiro and T. Maibaum. Categorical semantics of parallel program design. *Science of Computer Programming*, 28:111–138, 1997.
18. F. Gadducci. Term graph rewriting for the π-calculus. In A. Ohori, editor, *Proceedings of APLAS 2003, 1st Asian Symposium on Programming Languages and Systems*, volume 2895 of *Lect. Notes in Comput. Sci.*, pages 37–54. Springer Verlag, 2003.
19. F. Gadducci and U. Montanari. The tile model. In G. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*, pages 133–166. MIT Press, 2000.
20. F. Gadducci and U. Montanari. A concurrent graph semantics for mobile ambients. In *Proceedings of MFPS 2001, 17th Conference on the Mathematical Foundations of Programming Semantics*, volume 45 of *Elect. Notes in Th. Comput. Sci.* Elsevier Science, 2001.
21. F. Gadducci and U. Montanari. Comparing logics for rewriting: Rewriting logic, action calculi and tile logic. *Theoret. Comput. Sci.*, 285(2):319–358, 2002.
22. P. Gardner. From process calculi to process frameworks. In C. Palamidessi, editor, *Proceeding of CONCUR 2000, 11th International Conference on Concurrency Theory*, volume 1877 of *Lect. Notes in Comput. Sci.*, pages 69–88. Springer Verlag, 2000.
23. J. Goguen. Categorical foundations for general systems theory. In F. Pichler and R. Trappl, editors, *Advances in Cybernetics and Systems Research*, pages 121–130. Transcripta Books, 1973.
24. M. Hasegawa. *Models of sharing graphs*. PhD thesis, University of Edinburgh, Department of Computer Science, 1997.
25. D. Hirsch, P. Inverardi, and U. Montanari. Reconfiguration of software architecture styles with name mobility. In A. Porto and G.-C. Roman, editors, *Proceedings of Coordination 2000, 4th International Conference on Coordination Languages and Models,*, volume 1906 of *Lect. Notes in Comput. Sci.*, pages 148–163. Springer Verlag, 2000.
26. D. Hirsch and U. Montanari. Synchronized hyperedge replacement with name mobility. In K.G. Larsen and M. Nielsen, editors, *Proceedings of CONCUR 2001, 12th International Conference on Concurrency Theory*, volume 2154 of *Lect. Notes in Comput. Sci.*, pages 121–136. Springer Verlag, 2001.
27. K. Honda and M. Tokoro. An object calculus for asynchronous communication. In P. America, editor, *Proceedings of ECOOP'91, European Conference on Object-Oriented Programming*, volume 512 of *Lect. Notes in Comput. Sci.*, pages 133–147. Springer Verlag, 1991.
28. O.H. Jensen and R. Milner. Bigraphs and transitions. *SIGPLAN Not.*, 38(1):38–49, 2003.
29. O.H. Jensen and R. Milner. Bigraphs and mobile processes (revised). Technical Report TR580, Cambridge Computer Laboratory, 2004.
30. B. König. A graph rewriting semantics for the polyadic π-calculus. In J.D.P. Rolim, A.Z. Broder, A. Corradini, R. Gorrieri, R. Heckel, J. Hromkovic, U. Vaccaro, and J.B. Wells, editors, *Proceedings of ICALP Workshops 2000: GT-VMT 2000, Satellite Workshop on Graph Transformation and Visual Modeling Techniques*, volume 8 of *Proceedings in Informatics*, pages 451–458. Carleton Scientific, 2000.
31. J. Lamping. An algorithm for optimal lambda calculus reduction. In *Proceedings of POPL'90, 17th Annual ACM Symposium on Principles of Programming Languages*, pages 16–30. Association for Computing Machinery, 1990.

32. I. Lanese and U. Montanari. A graphical fusion calculus. In F. Honsell, M. Lenisa, and M. Miculan, editors, *Proceedings of CoMeta 2003, Final Workshop of the CoMeta Project*, Elect. Notes in Th. Comput. Sci., 2003. To appear.
33. I. Lanese and U. Montanari. Synchronization algebras with mobility for graph transformations. In J. Rathke, editor, *Proceedings of FGUC 2004, Satellite Workshop of CONCUR 2004 on Foundations of Global Ubiquitous Computing*, Elect. Notes in Th. Comput. Sci., 2004. To appear.
34. C. Laneve, J. Parrow, and B. Victor. Solo diagrams. In N. Kobayashi and B.C. Pierce, editors, *Proceedings of TACS 2001, 4th International Symposium on Theoretical Aspects of Computer Software*, volume 2215 of *Lect. Notes in Comput. Sci.*, pages 127–144. Springer Verlag, 2001.
35. C. Laneve and B. Victor. Solos in concert. *Math. Struct. in Comput. Sci.*, 13(5):657–683, 2003.
36. K.G. Larsen and L. Xinxin. Compositionality through an operational semantics of contexts. In M.S. Paterson, editor, *Proceedings of ICALP'90, 17th International Colloquium on Automata, Languages and Programming*, volume 443 of *Lect. Notes in Comput. Sci.*, pages 526–539. Springer Verlag, 1990.
37. J.J. Leifer and R. Milner. Deriving bisimulation congruences for reactive systems. In C. Palamidessi, editor, *Proceedings of CONCUR 2000, 11th International Conference on Concurrency Theory*, volume 1877 of *Lect. Notes in Comput. Sci.*, pages 243–258. Springer Verlag, 2000.
38. J.-J. Lévy. Optimal reductions in the lambda-calculus. In J.P. Seldin and J.R. Hindley, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 159–191. Academic Press, 1980.
39. J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoret. Comput. Sci.*, 96:73–155, 1992.
40. R. Milner. *Communication and concurrency*. Prenctice Hall, 1989.
41. R. Milner. Pi-nets: A graphical formalism. In D. Sannella, editor, *Proceedings of ESOP'94, European Symposium on Programming*, volume 788 of *Lect. Notes in Comput. Sci.*, pages 26–42. Springer Verlag, 1994.
42. R. Milner. Bigraphical reactive systems. In K.G. Larsen and M. Nielsen, editors, *Proceedings of CONCUR 2001, 12th International Conference on Concurrency Theory*, volume 2154 of *Lect. Notes in Comput. Sci.*, pages 16–35. Springer Verlag, 2001.
43. R. Milner, J. Parrow, and J. Walker. A calculus of mobile processes, I and II. *Inform. and Comput.*, 100(1):1–40,41–77, 1992.
44. U. Montanari and M. Pistore. Concurrent semantics for the pi-calculus. In *Proceedings of MFPS 1995, 11th Conference on the Mathematical Foundations of Programming Semantics*, volume 1 of *Elect. Notes in Th. Comput. Sci.* Elsevier Science, 1995.
45. J. Parrow. Interaction diagrams. *Nordic Journal of Computing*, 2(2):407–443, 1995.
46. J. Parrow and B. Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *Proceedings of LICS'98, 13th Annual Symposium on Logic in Computer Science*, pages 176–185. IEEE Computer Society Press, 1998.
47. G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, Computer Science Department, 1981.
48. D. Plump. Term graph rewriting. In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 2, pages 3–61. World Scientific, 1999.
49. A. Rensink. Bisimilarity of open terms. *Inform. and Comput.*, 156(1-2):345–385, 2000.
50. D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis, LFCS, University of Edinburgh, 1993. CST-99-93 (also published as ECS-LFCS-93-266).

51. P. Sewell. From rewrite rules to bisimulation congruences. In D. Sangiorgi and R. de Simone, editors, *Proceedings of CONCUR'98, 9th International Conference on Concurrency Theory*, volume 1466 of *Lect. Notes in Comput. Sci.*, pages 269–284. Springer Verlag, 1998.
52. G. Stefanescu. Interactive systems: from a natural language expression, to a mathematical concept. In H.C.M. de Swart, editor, *Proceedings of RELMICS 2001, 6th International Workshop on Relational Methods in Computer Science*, volume 2561 of *Lect. Notes in Comput. Sci.*, pages 197–211. Springer Verlag, 2001.
53. G. Stefanescu. Interactive systems with registers and voices. Draft available at `http://www.comp.nus.edu.sg/~gheorghe/papers/rvsystems.pdf`, 2004.
54. N. Yoshida. Graph notation for concurrent combinators. In T. Ito and A. Yonezawa, editors, *Proceedings of TPPP'94, International Workshop on Theory and Practice of Parallel Programming*, volume 907 of *Lect. Notes in Comput. Sci.*, pages 393–412. Springer Verlag, 1995.