Automatic Compilation of Concurrent Hybrid Factories from Product Assembly Specifications

Eric Klavins

Advanced Technology Laboratories
Department of Electrical Engineering and Computer Science
University of Michigan
1101 Beal Avenue
Ann Arbor, MI 48109-2110, USA
klavins@eecs.umich.edu

Abstract. We address the problem of designing a distributed, hybrid factory given a description of an assembly process and a palette of controllers for basic assembly operations. In particular, we present a method that, starting with a product assembly graph (PAG), allows us to "compile" a factory description, consisting of a geometry and a hybrid, dynamical system representing the motions of robots on that geometry. This method is based on a formalism, which we have described in previous work, that allows us to manage the details of low level, continuous control of robot actuation and high level, logical control of various couplings of robot behaviors. The factory description is intended to be an aid in the design of an actual factory, if not directly implementable itself.

1 Introduction

Large distributed networks of robots and computers form the basis of modern manufacturing systems. These systems should be rapidly reconfigurable, to adjust to design changes in the products they assemble or to changes in the market. Furthermore, they must be easily programmable. These goals, however, are seldom achieved in practice because of the complexity that hundreds of interconnected, concurrently operating robots necessarily incurs. The programming process can be ad hoc and frequently results in a large fraction of the control code being "exception handler code". This cost is felt in terms of expensive programming projects, incompletely understood factory behavior, and a delay in the introduction of new products to the market.

In previous work, [11], [10], we described a formalism for representing and composing concurrent robotic systems which we believe addresses some of the problems in designing distributed, dynamic factories. Specifically, we introduced the notion of a *Threaded Petri Net* (TPN), which combines low level motion control of individual robots or small groups of robots with high level logic control to manage how couplings between robots change over time in the factory. We also introduced a way of composing TPNs to create larger TPNs and demonstrated several properties of TPNs and our composition rules. Although we believe these tools will prove applicable to a broad range of automation settings, our notion of assembly is more immediately inspired by the high flexibility, low volume setting targeted by the "Minifactory" of Rizzi et al. [17], wherein decentralized general

purpose robotic agents accomplish all the factory's parts transport and assembly operations in fluidly choreographed transactions. For example, a complex subassembly task requiring four or six coordinated degrees of freedom can only transpire in such a Minifactory when some subgroup of the decentralized robots "agrees" to collaborate closely in forming the specialized "machine" (the higher degree of freedom coordinated mechanism) suited to the specific task at hand. Of course, that alliance must be temporary, since each of the participating agents is required to play analogous but different roles in other machines, both prior and subsequent to the instantiation of the one in question. The TPN formalism provides tools to frame this problem.

In the present paper, we apply our work on TPNs and composition to the task of *automatically* compiling factory descriptions from a standard representation of a product assembly process called a *product assembly graph* or PAG. The factory description that results consists of: an allocation of robots of various types; a geometrical description of the space that these robots inhabit; and a concurrent hybrid dynamic system, represented by a TPN, which directly corresponds to the robot programs. We use results from our previous work to show that the resulting TPN is live and that it successfully implements the process specified by the PAG input.

It must be stressed that we presuppose an infrastructure of tunable and switchable feedback controllers which our compiler merely "puts together", in a safe and correct way, to realize the assembly process. Such a palette of controllers is relatively easy to build for environments well described by generalized damper dynamics [12], but becomes quite challenging when dynamical dexterity is required. For example, in [2], substantial "hand building" affords deployments of controllers whose domains of attraction explicitly include portions of the forward limit sets of their neighbors. Here, we simply assume that these "dynamical systems details" have been worked out via parameterized families of regulators, and represented in a way that allows us to use them with TPNs (see Section 4). We then focus on the logical coordination and scheduling problems that follow. We have, in fact, built such a palette and a compiler for a simple class of PAGs and simulated the resulting factories. Animations of these factories can be viewed at http://www.eecs.umich.edu/~klavins/mf/.

The paper is organized as follows. In Section 2, we review related research. In Section 3, we review TPNs and our composition method. In Section 4, we introduce mathematical models which represent robots, operations (those in the palette of controllers), and factories. In Section 5, we describe the compilation algorithm in detail and prove that it describes live and correct factories. Finally, in Section 6, we discuss a simple implementation of the compiler.

2 Background and Related Work

The research we report on in this paper draws from several areas: preimage backchaining of motion controllers, autonomous robot assembly, and hybrid discrete/continuous systems including Petri Nets. We review each of these areas as they pertain to the present research.

Preimage backchaining was introduced into the motion planning literature in [14] as a method of sequentially composing motion strategies. In [2] this method was extended to dynamically dexterous robot manipulators in work that serves

as the basis of our current research. In [11], we expanded these ideas to include the notion of concurrent composition of behaviors for the case of several robots in a shared workspace based on simple Petri Net composition methods. Similar methods are found in work on the bottom-up synthesis of Petri Nets, especially [13], where simple Petri Nets are combined along paths and invariants of the resulting net are obtained from the constituent nets. In the present paper, we use the properties of our compositional tools design and verify an algorithm that automatically compiles concurrent, hybrid factories.

The approach to assembly in [12], for simple situations, introduces an automatic method for constructing a control law that guides a single robot to assemble a product from its parts based on the notion of an artificial energy landscape wherein the configuration of least energy is the one in which the product is assembled. It is not obvious that this method could be extended to three dimensional systems with orientable parts. In this paper we take the view that the PAG of a product corresponds to a discrete and parallelized version of such a potential function. The individual steps of the assembly may be given by artificial potential field controllers, but the overall logic of the assembly is given by the PAG. This allows us to use multiple robots, as in a high volume factory setting.

Programs such as Archimedes [9] exist which transform the CAD description of a product into a PAG. Little research has been reported concerning translating the PAG directly into a layout and distributed program for a factory, although in the one example we know of, [19], the authors produce elementary conveyer belt layouts. In this paper we introduce a method that we believe will lead to a general procedure for carrying out such a translation.

Hybrid systems combine a discrete state and a continuous state into the same model. A common representation is the *hybrid automaton*, [7]. Many definitions of hybridized Petri Nets, serving various needs, have also been investigated: Continuous and Hybrid Petri Nets [4], Differential Petri Nets [5], and DAE-Petri Nets [1]. The last is most easily seen as an extension of hybrid automata. Our definition of Threaded Petri Net differs from these definitions in several regards. First, we consider a place in a net to be a controlled dynamic system on some subset of the *degrees of freedom* of the system, depending on the marking, and a transition fires when and only when the systems in its preset are in stable equilibrium states. Furthermore, transition firings *redistribute* the degrees of freedom of the system to other dynamic systems in a controlled manner.

3 Definitions and Basic Properties

In this section we introduce the formal ideas that underlie our compiler research. We refer the reader to [10] for the details. We adopt the following definition of a Petri Net, also called a condition/event net, found in [8].

Definition 3.1 *A* **Petri Net** *is a pair* (T, P) *where* T *is a finite set of elements called* **transitions** *and* $P \subseteq 2^T \times 2^T$ *whose elements are called* **places**.

We use standard Petri Net notation. If $\{\{a_1, ..., a_i\}, \{b_1, ..., b_j\}\} \in P$, we write $[a_1, ..., a_i; b_1, ..., b_j] \in P$. If $p = [a_1, ..., a_i; b_1, ..., b_j]$ then $left(p)$ is the set $\{a_1, ..., a_i\}$ and $right(p)$ is the set $\{b_1, ..., b_j\}$. A **marking** of a net (T, P) is a

set $m \subseteq P$. The **flow relation** F of a Petri Net (T, P) is the relation where $(t, p) \in F$ if $t \in left(p)$ and $(p, t) \in F$ if $t \in right(p)$. The **preset** of an element $x \in T \cup P$ is set $\{y \mid y\ F\ x\}$ and is denoted ${}^\bullet x$. The **postset** of x is the set $\{y \mid x\ F\ y\}$ and is denoted $x^\bullet$. See [15] for a detailed introduction.

In a graphical representation of a Petri Net, places are represented by circles and transitions by squares. In our research, a place represents a controlled dynamical subsystem decoupled from the entire system in question. Transitions represent discrete changes in the dynamics of subsystems.

3.1 Threaded Petri Nets

Suppose we have a collection of robots $r_1, ..., r_n$ with configuration spaces $\mathcal{C}(r_1)$, ..., $\mathcal{C}(r_n)$ whose continuous state can be given by $\mathbf{x} = (x_1, ..., x_n) \in \mathcal{C}(r_1) \times ... \times \mathcal{C}(r_n)$ and whose global dynamics is simply $\dot{\mathbf{x}} = \mathbf{u}$. The dynamics of components of $\mathbf{x}$ are almost independent of each other. However, robots do interact for short periods of time, as for example during a parts mating operation, so that the dynamics of certain components of $\mathbf{x}$ may occasionally be tightly coupled.

To describe how couplings change and which dynamics are operating on which components of $\mathbf{x}$, we introduce the *Threaded Petri Net*, or TPN. Places correspond to control modes which we will have chosen from a palette of such modes. Thus, for each place p there is a system given by $\dot{\mathbf{y}} = F_p(\mathbf{y})$ where $\mathbf{y}$ is the vector concatenation of l_p vectors (components of $\mathbf{x}$) and F_p is chosen from the palette of controllers that we assume is already constructed. The mode has domain of attraction $\mathcal{D}_p$ and goal set $\mathcal{G}_p$. Formally,

Definition 3.2 *A* **Threaded Petri Net** *(TPN) consists of*

1. *a set T of transitions;*
2. *a set $P \subseteq 2^T \times 2^T$ of places;*
3. *for each $p \in P$, size, dynamics, domain and goal l_p, F_p, $\mathcal{D}_p$ and $\mathcal{G}_p$;*
4. *for each $e \in T$ a bijective function*

$$d_e : \bigcup_{p \in {}^\bullet e} \{p\} \times \{1, ..., l_p\} \to \bigcup_{q \in e^\bullet} \{q\} \times \{1, ..., l_q\}$$

called the **redistribution function** *of e;*

subject to the condition that for each $e \in T$,

$$\sum_{p \in {}^\bullet e} l_p = \sum_{q \in e^\bullet} l_q$$

(so that it is possible for d_e to be bijective).

Note that the difference between a TPN and a condition/event net is not only the additional information associated with each place. We have also added the redistribution functions, d_e for each $e \in E$, which define what happens to each component of $\mathbf{x}$ as mode changes occur. Graphically, a TPN is depicted as is a simple Petri Net, except that the redistribution functions are shown by curves through the net. See Figure 1 for example.

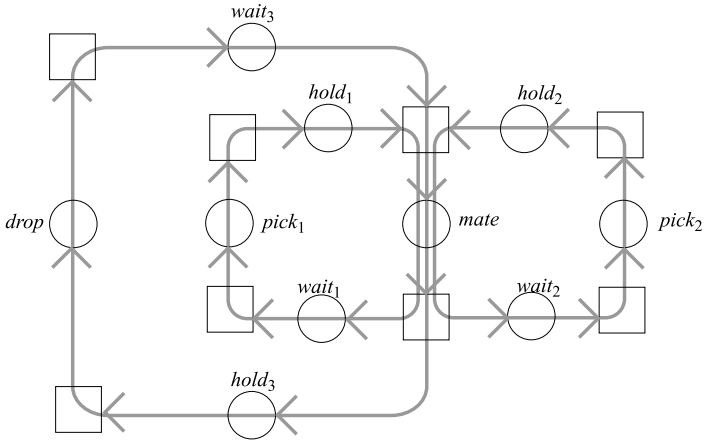

Fig. 1. An example of a Threaded Petri Net which describes the dynamics of three robots in a parts mating procedure.

Definition 3.3 *A* **marking** *is a pair* (m, f_m) *where* $m \subseteq P$ *and*

$$f_m : \bigcup_{p \in m} \{p\} \times \{1, ..., l_p\} \to \{1, ..., n\}$$

which specifies which degrees of freedom of the system each mode is operating on. A **legal** *marking is one where* f_m *is bijective. We will be concerned only with legal markings in what follows.*

A legal marking (m, f_m) of a TPN says, for each $p \in m$, which components of $\mathbf{x}$ F_p is acting on *and* what the dynamics of *each* component of $\mathbf{x}$ are. Thus, we can say how the state of the system is changing given a particular marking (m, f_m). Given $j \in N$, suppose that $f_m^{-1}(j) = (p, i)$. That is, under the marking (m, f_m) the jth component of $\mathbf{x}$ is changing according to the ith component of the mode dynamics of p:

$$\dot{x}_j = \pi_i \circ F_p(x_{f_m(p,1)}, ..., x_{f_m(p,l_p)})$$

where π_i gives the ith component of the function F_p. This is valid until some mode changes, which leads us to a definition of *how* events are triggered.

Definition 3.4 *Let* (m, f_m) *be a legal marking.* $e \in T$ *is* m**-enabled** *with respect to* $\mathbf{x} \in \mathbb{R}^n$ *if*

1. ${}^\bullet e \subseteq m$ *and* $e^\bullet \cap m = \emptyset$;
2. *for each* $p \in {}^\bullet e$, $(x_{f_m(p,1)}, ..., x_{f_m(p,l_p)}) \in \mathcal{G}_p$;
3. *for each* $q \in e^\bullet$, $(x_{f_m \circ d_e^{-1}(q,1)}, ..., x_{f_m \circ d_e^{-1}(q,1)}) \in \mathcal{D}_q$.

Notice that condition (1) is just the usual definition of m-enabled for condition/event nets. The second two conditions impose the restriction that the dynamic systems in the preset of the enabled event must be in goal states and the systems in the postset must all be prepared.

A set of events $G \subseteq E$ is called **detached** if whenever e_1 and e_2 are distinct events in G, ${}^\bullet e_1 \cap {}^\bullet e_2 = e_1{}^\bullet \cap e_2{}^\bullet = \emptyset$. Suppose we have a marking (m, f_m). The **follower marking** $(m', f_{m'})$ with respect to $G \subseteq E$ is calculated as follows. As with condition/event nets $m' = (m - {}^\bullet G) \cup G^\bullet$. $f_{m'}$ is the function given by

$$f_{m'}(p,j) = \left\{ \begin{array}{l} f_m(p,j) \;\; if \;\; p \in m - {}^\bullet G \\ f_m \circ d_e^{-1}(p,j) \;\; otherwise \end{array} \right\}$$

where e is the single event in $p^\bullet \cap G$. We write $(f_m, m) \rightarrow^G (f_{m'}, m')$ when $(f_{m'}, m')$ is the follower marking of (f, m) with respect to G. Since legal markings (m, f_m) are such that f_m is bijective, we can be sure that *every* component of $\mathbf{x}$ is accounted for when the system is in the set of modes given by m. It can be shown that if (f_m, m) is a legal marking and if $(f_m, m) \rightarrow^G (f_{m'}, m')$, then $(f_{m'}, m')$ is a legal marking as well.

3.2 Composing Threaded Petri Nets

As mentioned, we intend to compose TPNs into factories. We present a simple type of composition to complete this section. It is based on the idea of a cyclic subprocess, which we call a **gear**, and which we use as the basic building block of our nets. A gear represents the simplest thing a robot in a factory can do, besides remain idle: cycle repeatedly through some set of behaviors.

Definition 3.5 *A* k**-gear** *is a net* (T, P) *where* $T = \{t_0, ..., t_{k-1}\}$ *and* $P = \{[t_i; t_{i+1}] \mid i \in \mathbb{Z}/k\}$. $m \subseteq P$ *is a* **legal marking** *for a* k*-gear if* $|m| = 1$.

(We ignore the dynamics and redistribution functions for now.) A gear for a robot models the program of a single robot. Certain places of a gear must be synchronized with the gears of other robots. Thus, we *compose* gears as follows.

Definition 3.6 *A* **gear net** *is defined recursively:*

1. *A gear is a gear net.*
2. *If* (T, P) *is a gear net and* (S, Q) *is a gear then* $(T \cup S, P \cup Q)$ *is a gear net as long as the following conditions hold:*
   (a) *let* $(T_1, P_1), ..., (T_k, P_k)$ *be the set of gears in* $(T \cup S, P \cup Q)$ *which intersect* (S, Q). *Then* $\bigcap_{i=1}^k P_i = \{[a; b]\}$ *and* $\bigcap_{i=1}^k T_i = \{a, b\}$ *for some transitions* a *and* b;
   (b) *there exists a transition* $c \in S - T$ *such that* $[c; a] \in Q$.

A **legal marking** *for a gear net is one in which each gear in the net is marked exactly once.*

Since all places in a gear net are of the form $[x; y]$, gear nets are a kind of *marked graph*, a class of nets which have been extensively studied. (See [3], for example.) Conditions (a) and (b) require that gears be added with a "standard interface".

We can show the following properties about gear nets.

Theorem 3.1 *(Liveness) Gear nets are deadlock free under legal markings.*

Theorem 3.2 *(Reversibility) Gear nets are reversible given any legal initial marking.*

Thus we are assured that systems we build up from gear nets are live, logically conflict free, and cyclic processes.

4 Representation

Next we describe how to represent the building blocks of factories – products, robots, workspaces and controllers – in a way that is amenable to compilation.

4.1 The Product Assembly Graph

A **product assembly graph** or PAG, is represented as a tree whose leaves represent parts and whose internal nodes represent operations on subtrees which yield subassemblies. For a given set of operations and part types we can define a simple class of PAGs as follows. Suppose that we have part types $part_1, ..., part_k$ and operations $\mathcal{O}_1, ..., \mathcal{O}_j$ where $\mathcal{O}_i$ is an operation which takes m_i subassemblies and produces a single subassembly. Then the class of PAGs is given by:

1. $part_1, ..., part_k$ are all PAGs;
2. for each $i \in \{1, ..., j\}$, if $P_1, ..., P_{m_j}$ are all PAGs then $\mathcal{O}_i(P_1, ..., P_{m_j})$ are PAGs as well.

Clearly, this defines a very simplified class of PAGs. In practice, each operation can take only certain types of subtrees (those representing subassemblies appropriate to the operation), operations are parameterized, and so on. However, we believe that this is a first approximation to the kind of PAGs that we will encounter in practice.

For a given PAG P, we give a unique label to each node P' in P, called $Label(P')$. This identifies the subassembly that is result of the operation.

4.2 Robot Types and Workspaces

We suppose that there is some set of *robot types* at our disposal which we denote by $\mathcal{T} = \{T_1, T_2, ...\}$. Each type T has an "ideal" workspace $\mathcal{W}(T) \subseteq \mathbb{R}^3$ (compact and connected) and a configuration space $\mathcal{C}(T)$. $\mathcal{W}(T)$ describes the geometry of the set of all positions the robot may take – in general, a solid in $\mathbb{R}^3$. $\mathcal{C}(T)$ represents the degrees of freedom of the robot. An example robot type in the Minifactory is the courier, a two degree of freedom planar robot with a workspace that is a rectangular solid $[x_{min}, x_{max}] \times [y_{min}, y_{max}] \times [0, h]$ where the x and y terms represent the limits of movement on a factory platen and h is the height of the robot. The configuration space of a courier is just $\mathbb{R}^2$.

An instantiation of a robot will be denoted by an identifier r with type $Type(r) \in \mathcal{T}$, workspace $\mathcal{W}(r) \simeq \mathcal{W}(Type(r))$, and configuration space $\mathcal{C}(r) = \mathcal{C}(Type(r))$. As we build factories in the compilation procedure defined below, we instantiate new robots and add them to a set R of robot identifiers. We suppose

that their ideal workspaces are copies of the ideal workspaces of their types and that for any two distinct instantiated robots r_1 and r_2, we have $\mathcal{W}(r_1) \cap \mathcal{W}(r_2) = \emptyset$. We represent the way in which robots are located with respect to each other by forming an identification (quotient) topology on the union of the workspaces of the robots. This *does not* represent the actual layout of the factory because the resulting geometry may not embed in $\mathbb{R}^3$ without some "stretching". We comment on the layout procedure in Section 5.

A Robot can carry a subassembly, which may be an atomic part or the result of some operation on some number of parts. Which subassembly, if any, a robot is carrying is the discrete state of the robot. It is given by $Label(P)$ for some node P of the PAG that is being compiled. The distinguished label *nopart* will be used to denote the state of a robot not carrying any subassembly.

4.3 Templates for Controllers

In order to use controllers with our assembly compiler, they must be represented in a standard way. Here we describe a template for representing controllers. This template consists of: a description of the robots needed; the index of the robot, called the "carrier", that will hold the result of the operation once the it is complete; a way of combining the workspaces of the robots into a workspace for the operation; a *parts transform pair*; and a control law over the configuration space. The carrier robot and its workspace are used to join the workspaces of controllers as the PAG is traversed during compilation. We have the following definition:

Definition 4.1 *An* **operation template** *is a tuple*

$$\mathcal{O} = (R, j, \sim, \langle \mathbf{a}; \mathbf{b} \rangle, \mathbf{F})$$

where

1. $R = \langle T_1, ..., T_k \rangle$ *is an ordered set of types of robots, with* $k = |R|$*;*
2. $j \in \{1, ..., k\}$ *is the index of the robot that will carry the result;*
3. $\sim$ *is an equivalence relation.* $\mathcal{W} = (\bigcup_{i=1}^{i=k} \mathcal{W}(r_i))_{/\sim}$ *is the resulting workspace;*
4. $\langle \mathbf{a}; \mathbf{b} \rangle = \langle a_1, ..., a_k; b_1, ..., b_k \rangle$ *is the* **parts transform pair** *denoting how the labels of the parts each robot is carrying change as a result of the controller reaching its equilibrium state;*
5. F *is a vector field on* $\mathcal{C}$ *describing the controlled dynamic system corresponding to the controller with domain* $\mathcal{D}_F$ *and goal* $\mathcal{G}_F$*.* $\mathcal{C}$ *is defined by* $\prod_{T \in R} \mathcal{C}(T) - \Delta$ *where* Δ *is the set of configurations which correspond to two robots touching or being in the same place according to* $\sim$*.*

The operations of interest take some number of subassemblies and perform an operation that produces one new subassembly. Thus, $b_j \neq nopart$ while $b_i = nopart$ for $i \neq j$.

An **instantiation** of a template is an assignment of robot identifiers to R and is written $\mathcal{O}(r_1, ..., r_k)$. The Threaded Petri Net fragment corresponding to this instantiation is denoted $N_{\mathcal{O}}(r_1, ..., r_k)$ and is depicted, in its general form, in Figure 2.

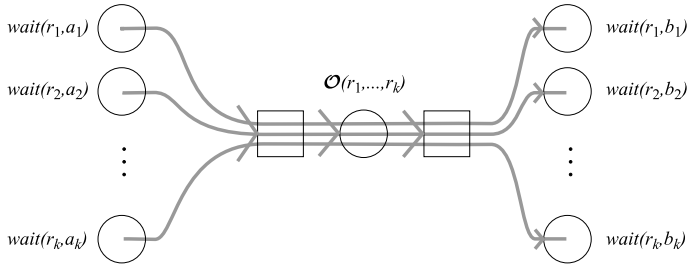

Fig. 2. The Threaded Petri Net associated with the instantiation $\mathcal{O}(r_1, ..., r_k)$

4.4 Factories

We define a factory to be a workspace, a set of robots in the workspace, and a TPN describing the dynamics of the factory. This structure will be built up as the compilation procedure progresses. It will start with a single robot whose task it is to receive the final subassembly from the highest operation in the PAG.

A **factory**, therefore, is a triple $\mathcal{F} = (R, \sim, N)$ where R is a set of robot identifiers, $\sim$ is an equivalence relationship on the union of the workspaces of the robots which describes how the robots are placed in the factory, and N is a TPN which describes the hybrid dynamics of the factory.

5 The Compilation Algorithm

In this section we describe the general form of the compilation procedure for a given class of PAGs. We assume that each operation is already described via a template and that templates for the operations for picking up parts (from parts feeders or trays) and dropping off the final subassembly part are also given. Assume that the type of robot that receives the final subassembly is $OuputType$. The input to the algorithm is a PAG $P = \mathcal{O}(P_1, ..., P_k)$. The function Compile initializes the factory structure with a robot and workspace for the final subassembly $DropOff$ operation and then calls the main function CompileNode.

Compile(P)
  $r \leftarrow Instantiate(OutputType)$
  $R \leftarrow \{r\}$
  $\sim \leftarrow \{(x, x) \mid x \in W(r)\}$
  $N \leftarrow N_0$
  CompileNode(P, r)
End

Here, N is initialized to N_0 which is the Fragment depicted in Figure 3.
The subroutine CompileNode first adds to the factory the robots and workspaces required for the operation $\mathcal{O}$ and then applies itself to each of the subtrees P_1 through P_k. Assume that $P = \mathcal{O}_i(P_1, ..., P_{m_i})$ where$\mathcal{O}_i = (R_i, j_i, \sim_i, \langle \mathbf{a}_i; \mathbf{b}_i \rangle, F_i)$ with $R_i = \langle T_1, ..., T_k \rangle$

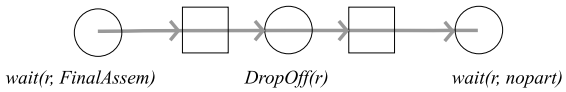

Fig. 3. The Threaded Petri Net N_0 used to initialize the factory before compilation. $FinalAssem$ is the label of the root of the input PAG.

```
CompileNode ( P, carrier )
    Allocate robot identifiers r_l where Type(r_l) = T_l for each l ≠ j
    r_j ← carrier
    R ← R ∪ {r_1, ..., r_k}
    ∼ ← (∼) ∪ (∼_i)
    N ← N ∪ N_{O_i}(r_1, ..., r_k)
    For each l ∈ {1, ..., k}
        If a_l ≠ nopart Then
            Choose P ∈ {P_1, ..., P_{m_i}} such that Label(P) = a_i
            CompileNode(P, r_l)
        EndIf
    EndFor
End
```

The CompileNode routine first allocates the new robot identifiers needed for the operation. The factory robots are updated to include these robots as well as the carrier. The equivalence relation is updated as well and becomes a relation over the union of workspaces of all the newly allocated robots as well as the robots that were already in R. Then the TPN that describes the dynamics of the factory is updated to include the fragment for the operation. Finally, for each robot identifier r which, according to the part transition pair $\langle \mathbf{a}; \mathbf{b} \rangle$, should be arriving at the current operation with a part a, CompileNode calls itself on the subtree corresponding to a with r as the new carrier robot. Notice that the recursion eventually bottoms out since the part nodes (leaves) of the PAG have no children.

5.1 Properties of the Resulting Factory

We can show that the factory is a gear net and that its dynamics are correct. We first make use of the following lemma.

Lemma 5.1 *Let $(T_1, P_1), ..., (T_k, P_k)$ be gear nets and suppose that for each $i \in \{1, ..., k\}$ we have that $(\{a_i, b_i\}, \{[a : b]\}) \subseteq (T_i, P_i)$ is the intersection of some number of gears in (T_i, P_i). Then, the net obtained by identifying each $(\{a_i, b_i\}, \{[a : b]\})$ is also a gear net.*

This result can be used to show that the algorithm above produces gear nets. The proof is inductive on the form of the PAG input. Roughly, we show that PAGs consisting of a single part produce single gears and that assuming that the algorithm compiles gear nets for the subtrees $P_1, ..., P_k$ of the tree $P = \mathcal{O}(P_1, ..., P_k)$, we show that it compiles P correctly into a gear net as well.

Theorem 5.1 *Let N be the TPN resulting from applying the above algorithm to the PAG P. Then N is a gear net.*

Since gear nets are live and reversible and because they are deterministic it is also straightforward to show that under any legal initial conditions (we usually consider the situation where each robot is running $wait(r_i, nopart)$ as the initial marking), that the output robot runs the controller for the $DropOff$ operation infinitely many times in any run. Formally,

Theorem 5.2 *Suppose that $m_0, m_1, \ldots$ is a sequence of markings obtained from a run of the gear net N produced from PAG P. Then there exist infinitely many markings m in the sequence such that $DropOff(r) \in m$ where the robot r is the one instantiated in the initialization routine* Compile.

The workspace that results from compiling a PAG, $\mathcal{W} = (\bigcup_{r \in R} \mathcal{W}(r))/\sim$, does not represent the layout of the factory. In general, $\mathcal{W}$ needs to be "stretched" to be properly embedded in $\mathbb{R}^3$, if it is even possible to do so. At present, we do not have a complete procedure for producing this layout, however, we have an idea of how it will be carried out in practice. Certain workspace types are amenable to stretching in certain directions. For example, the workspace of a planar robot may be extended to be longer or wider but not taller. Thus, there is an allowable family of embeddings $\mathcal{F}$ from $\mathcal{W}$ into $\mathbb{R}^3$ which must be explored. Once one is found, say $f \in \mathcal{F}$, the controllers for the low level operations are composed with f to produce dynamics on the image of f. In the next section, we illustrate this procedure in a simple implementation.

6 The DotFactory: An Example

We have explored the compilation procedure with a simple family of PAGs and a class of "toy" factories called "DotFactories". In the simplest of our investigations, we assume that there is only one part type, $atomic()$, and two operations $mate(\cdot,\cdot)$, $weld(\cdot)$. The robots we consider are all of the same type T_{dot} with workspaces that are copies of the unit interval $[0,1] \in \mathbb{R}$ and configuration spaces $[0,1]$ (guidepaths). The physics are simplified: a robot may control its velocity directly ($\dot{x} = u$); parts move with the robots nearest to them; and part transfers happen instantaneously as long as the robots involved are close together. Robots have width r.

An example template is given next, for the $mate$ operation. $mate = (R, j, \sim, \langle \mathbf{a}; \mathbf{b} \rangle, \mathbf{F})$ where

1. $R = \langle T_{dot}, T_{dot}, T_{dot} \rangle$;
2. $j = 3$;
3. $\sim = \{A_1 = A_2 = B_3\}$ where we assume that robot i will have as its workspace the interval $[A_i, B_i] \in \mathbb{R}$;
4. $\langle \mathbf{a}; \mathbf{b} \rangle = \langle LAB_1, LAB_2, nopart; nopart, nopart, LAB_3 \rangle$;
5. $\mathbf{F}$ is a control law over $\mathcal{W}_{mate} = (\bigcup_{i=1}^{3} [A_i, B_i])/\sim$ with $\mathcal{D} = \mathcal{W}_{mate}$ and $\mathcal{G} = B_\epsilon(A_1 + 2r, A_2 + 2r, B_3)$ (a small open ball around the goal point).

We omit a description of the details of F. In the actual implementation, F is derived from a *navigation function* [16] – a method that is quite suitable to the present situation. Similar templates are given for the *weld*, *atomic* and *dropoff* operations.

The input PAG is represented syntactically as in the following example input file:

```
6 parts; // the number of subassemblies
root = sub3; // the finished product
sub3 = mate ( sub2, part3 ); // how to make the subassemblies
sub2 = weld ( sub1 );
sub1 = mate ( part1, part2 );
part1 = atomic();  // these are the actual parts
part2 = atomic();
part3 = atomic()
```

The TPN that is compiled from this PAG describes programs and low level control for six robots in a workspace composed of guidepaths. Since the PAG is a tree, the compiler constructs workspaces that are, topologically, trees as well so that the layout procedure is obvious. The programs for each robot, essentially gears, can be read off directly from this TPN. For example, the gear for the robot, call it r_3, that receives the result of subassembly 1, fixes it to be welded, and then mates it with part 3 is

Loop:
  If $state_3 = nopart$
    Run $\dot{x}_3 = wait$ Until $state_5 = part1 \wedge state_6 = part2$
    Run $\dot{x}_3 = \pi_3 \circ mate(x_5, x_6, x_3)$ Until $x_3 = sub1$
    Break
  If $state_3 = sub1$
    Run $\dot{x}_3 = hold$ Until $state_4 = nopart$
    Run $\dot{x}_3 = \pi_1 \circ weld(x_3, x_4)$ Until $state_3 = sub2$
    Break
  If $state_3 = sub2$
    Run $\dot{x}_3 = wait$ Until $state_1 = nopart \wedge state_2 = part3$
    Run $\dot{x}_3 = \pi_1 \circ mate(x_3, x_2, x_1)$ Until $state_3 = nopart$
    Break
End Loop

Programs for the other robots are similar. Note that we assume a simple communication system which, in our implementation, is composed of two parts: a *shared memory* where robot i may write its discrete state (the label of the part it is carrying) to memory location i and may read any memory location; and a high speed continuous state sharing link between robots sharing control modes. Because of the distributed nature of the control, the number of continuous states a robot must monitor at any time is less than or equal to the size of the the largest control mode, *independent* of the size of the PAG and the resulting factory. We believe that the method will scale well to significantly larger factories.

Each robot is simulated concurrently at varying operating speeds (chosen randomly) and with varying control speeds. All factories that were compiled performed well under these minor disturbances due to the reactive nature of the low level control method used (borrowed from [16]) and to the robust nature afforded by the gear net structure of the compiled TPN.

We have also investigated robot types with workspaces that are "T-shaped" and shared by another robot. We use the method suggested by Ghrist and Koditschek in [6] for constructing dynamical systems of multiple points on topological graphs. Animations of the factories resulting from several different input PAGs can be viewed at http://www.eecs.umich.edu/~klavins/mf/.

7 Conclusion

We have developed an automatic factory compiler based on our formalism for representing concurrent, hybrid systems. The compiler uses a standard representation of robot workspaces and low level operations and yields a factory geometry, robot task allocation and control programs for each robot. The resulting factory dynamics are shown to be correct using basic properties of our gear net composition method. Our implementation of a simple toy situation suggests that our method yields robust systems and that it scales well.

In the future, we will consider optimizing the compiled net for robot reuse (i.e. reallocating tasks so that one robot alternates between tasks formerly assigned to two robots) and for parallelization of tasks. This leads to TPNs that are not based on gear nets but do have a regular structure, and implies the need for a much more sophisticated layout procedure. The dynamics of the resulting nets must be considered with fairness constraints so that they do not deadlock. We must also address the issues of error recovery and product reworking. We believe that the complexities these issues introduce into our TPNs can be managed by compositional methods similar to those we have already introduced. We are also working on applying these ideas to a factory design tool for a more realistic example which better approaches the Minifactory, mentioned in Section 1 [17].

This research has also lead us is to study the idea of "momentum across transitions" where the dynamical systems corresponding to places are not always controlled to equilibrium states. For example, a robot might toss a ball to another robot which must catch the ball. As the ball approaches the second robot, the transition of that robot into a catching behavior becomes more urgent. We would like to be able to solve this problem not with the explicit use of time as in timed Petri Nets but rather with the intrinsic dynamics of, in this case, a ball in flight. An example of switching between tasks based on urgency can be found in [18] where Rizzi controls a robot to switch between the tasks of bouncing one of two balls on a paddle, effectively juggling them. A systematic approach to this problem may yield factories that are highly dexterous, distributed manipulation systems.

Acknowledgments

The author thanks Professors Bill Rounds and Dan Koditschek for many conversations and advice about this work and Al Rizzi for introducing him to the Minifactory. Eric Klavins is supported in part by the Charles DeVlieg Foundation Fellowship for Manufacturing.

References

1. D. Andreu, J. Pascal, H Pingaud, and R. Valette. Batch process modeling using Petri Nets. In *Proc. of 1994 Intl. Conf. on Systems, man, and Cybernetics*, pages 314–319, October 1994.
2. Robert R. Burridge, Alfred A. Rizzi, and Daniel E. Koditschek. Sequential composition of dynamically dexterous robot behaviors. *International Journal of Robotics Research*, 1998.
3. F. Commoner, A.W. Holt, S. Even, and A. Puneli. Marked directed graphs. *Journal of Computer and System Sciences*, 5:511–523, 1971.
4. R. David and H. Alla. Continuous Petri Nets. In *8th European Workshop on Application and Theory of Petri Nets*, pages 275–294, Saragosse, 1987.
5. I. Demongodin and N. Koussoulas. Differential Petri Nets: Representing continuous systems in a discrete-event world. *IEEE Transactions on Automatic Control*, 43(4):573–579, April 1998.
6. R. Ghrist and D. Koditschek. Safe cooperative robotic motions via dynamics on graphs. In Y. Nakayama, editor, *8th Intl. Symp. on Robotics Research*. Springer Verlag, 1998.
7. T. Henzinger, P.H. Ho, and H. Wong-Toi. HYTECH: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:110–122, 1997.
8. Ryszard Janicki. Nets, sequential compositions and concurrency relations. *Theoretical Computer Science*, 29:87–121, 1984.
9. Stephen G. Kaufman et al. The Archimedes 2 mechanical assembly planning system. In *Proceedings of the 1996 IEEE Conference on Robotics and Automation*, pages 3361–3368, 1996.
10. Eric Klavins and Daniel Koditschek. A formalism for the composition of loosely coupled robot behaviors. Technical report no. CSE-TR-412-99, University of Michigan, 1999.
11. Eric Klavins and Daniel Koditschek. A formalism for the composition of concurrent robot behaviors. In *Proceedings of the IEEE Conference on Robotics and Automation*, 2000.
12. Daniel E. Koditschek. An approach to autonomous robot assembly. *Robotica*, 12:137–155, 1994.
13. B. H. Krogh and C. L. Beck. Synthesis of place/transition nets for simulation and control of manufacturing systems. In *4th IFAC/IFORS Symp. Large Scale Systems*, pages 661–666, Zurich, 1986.
14. Tomás Lozano-Perez, Matthew T. Mason, and Russell H. Taylor. Automatic synthesis of fine-motion strategies for robots. *The International Journal for Robotics Research*, 3(1):3–23, 1984.
15. Wolfgang Reisig. *Petri Nets: An Introduction*. Springer Verlag, 1985.
16. Elon Rimon and Daniel E. Koditschek. Exact robot navigation using artificial potential fields. *IEEE Transactions on Robotics and Automation*, 8(5):501–518, October 1992.
17. A. A. Rizzi, J. Gowdy, and R. L. Hollis. Agile assembly architecture: An agent based approach to modular precision assembly systems. In *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, pages 1511–1516, Albuquerque, NM, April 1997.
18. Alfred A. Rizzi. *Dexterous Robot Manipulation*. PhD thesis, University of Michigan, 1994.
19. Bruce Romney, Cyprien Godard, Michael Goldwasser, and G. Ramkumar. An efficient system for geometric assembly sequence generation and evaluation. In *Proceedings of the 1995 AMSE. Intl. Computers in Engineering Conf.*, pages 699–712, 1995.