A Formalism for the Composition of Loosely Coupled Robot Behaviors

Eric Klavins and **Daniel Koditschek**
Artificial Intelligence Laboratories
Department of Electrical Engineering and Computer Science
University of Michigan
1101 Beal Avenue, Ann Arbor, MI 48109-2110
{klavins,kod}@eecs.umich.edu

Abstract

We address the problem of controlling large distributed robotic systems such as factories. We introduce tools which help us to compose local, hybrid control programs for a class of distributed robotic systems, assuming a palette of controllers for individual tasks is already constructed. These tools, which combine backchaining behaviors with Petri Nets, expand on successful work in sequential composition of robot behaviors. We apply these ideas to the design of a robotic bucket brigade and to simple, distributed assembly tasks.

Contents

1 Introduction

One of the basic limitations in engineering systems is our rudimentary understanding of large scale, complicated, engineered systems such as factories. We do have tools, in control theory for example, for constructing elegant and precise components which react with their environments in simple and provably correct ways, and we also have tools for compiling gigantic logical systems, such as computer operating systems or central processing units. We currently only have a limited understanding of how to compose situated systems, such as robots and other objects, which sense and actuate in the real world directly, with each other into more complicated systems. With such compositional tools we could, in principle, synthesize quite complicated, yet completely understood, dynamic systems from simple systems. The systems in which we will be particularly interested in this paper are automated factories.

Many simulation based design tools exist so that factories can be designed with as little reconfiguration in hardware as possible. The idea is that the factory designer can design and test a virtual factory in software and hopefully discover and solve problems in candidate designs before beginning the costly process of putting together an actual factory, with robots, conveyer belts, power and control cables, and other supporting hardware. In general, the cost of fixing a design flaw is much greater once the factory has been built than before it was constructed. However, the cost of designing the layout of the factory, the logic and control of its robots, and communication and coordination procedures between components, remains almost as high. This is because the process of translating the formal specification of the product, in terms of its geometry and assembly procedures, into a factory which assembles the product remains a craft practiced only by experts in factory layout and control software programming and lacks a complete set of formal methods for designing factories as well as verifying factory designs. These problems have been noted in other systems as well:

> Hand coding functions for maintaining the system's internals traditionally requires the programmer to reason through system wide interactions, along lengthy paths between the sensors, control processor and control actuators. This reasoning requires thinking about the behavior of a hybrid system, composed of complex real-time software constructs, distributed digital hardware and continuous physical processes.
>
> — Brian C. Williams, [30]

Thus, the problem of automating this process is difficult and poorly understood, and the tasks of designing factories and of reconfiguring factories to accommodate product changes are slow. A higher cost of getting a product to market, and more importantly a longer time to market, is the main effect of this lack of automation.

We would like to be able to understand more completely, in order to further automate the factory design process, is a theory of distributed, modular control of robotic systems. Such a theory must include elements of logical control, continuous control, communication and concurrency issues, and a compositional semantics for factory components blended in a way that facilitates the automatic synthesis of factory designs – from layout to control – from the most basic description of the product as possible. We view the synthesis procedure as a sort of *compilation* of a factory from a syntactic description of the product via an *assembly graph*. This requires a solid foundation in composing factory components – actually, controlled hybrid dynamic systems – that respects the common needs of factory design including: decentralized control, modularity, resistance against disturbances both physical and logical, and of course speed and efficiency.

Of course, a completely automatic factory synthesis procedure is a long way off. In order to simplify these basic steps toward our goal, we abstract away to more fundamental research problems. Essentially, the question is this: What are the fundamental tools for compiling distributed, concurrent processes (factories) from syntactic representations of processes (product descriptions)? Or, said another way, how can we construct cooperative systems of robots to perform tasks which have been represented syntactically? Such tools must be concise, formal, provably correct, compositional, and scalable. The resulting distributed process must be correct and robust.

We address the problems of concurrency and composition of behaviors in this paper by introducing a formalism which subsumes the work in sequential composition. We define a way in which simple Petri Nets can contain a more general form backchaining robot behaviors. We call the resulting nets *Threaded Petri Nets*, or TPNs. We also describe a simple net composition method which lends itself well to the kinds of decentralized assembly tasks encountered in manufacturing systems. This method allows us to compose many single robot programs into decentralized, concurrent programs for groups of robots that are guaranteed not to deadlock. Finally, we give several increasingly complicated examples of how this formalism may be used to automatically construct provably correct distributed robotic systems: a robot bucket brigade, a simple assembly arrangement, and the beginnings of a factory compiler.

1.1 Overview

In Section 2 we discuss the context of this research and related work. In Section 3, we introduce our method of composing simple Petri Net cycles, called *gears*, into a class of Petri Nets called *gear nets*. Gears represent single robot behaviors, and gear nets represent the safe composition of many robot behaviors. We prove that our composition method produces live Petri Nets. In Section 4, we add to the Petri Net a way to keep track of certain sequential components which correspond to the actions of robots and the paths of parts in a distributed manipulation setting. In Section 5 we present two examples of how our formalism can be used to model certain simple arrangements of robots. Then, in Section 6, we show how we can build a compiler which synthesizes simple "toy" factories, from product assembly descriptions, which we can prove correct using these tools.

We also include two appendices. In the first, we review some basic notions from control theory. In the second we provide the basic definitions pertaining to Petri Nets and partial order theory. The latter appendix is recommended reading even for those with experience in Petri Nets because of a slight deviance from the usual notation.

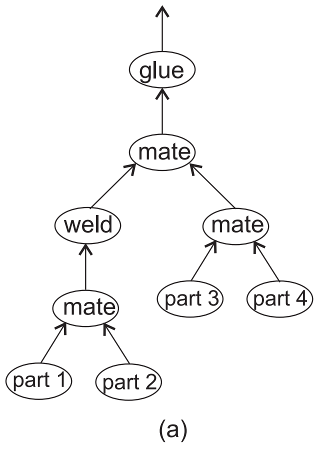

Figure 1: (a) A product assembly graph. (b) A work flow graph where the two lowest mate operations have been identified and the glue operation has been parallelized.

2 Background and Related Work

In this section we review research related to synthesizing factories and we present, briefly, some of the foundations upon which the present work is built.

2.1 The Product Assembly Graph

The *product assembly graph*, or PAG, is the starting point of our problem. Every product has a PAG which encodes how the product is put together. The PAG for a product is actually a tree. The root node represents the assembled product, the leaves represent the raw materials, or atomic parts, and nodes represent operations on subassemblies which produce compound subassemblies. See Figure 1(a) for an example. A closely related concept is the *work flow graph*, WFG, which encodes more details about the methods involved in assembling the product. In a WFG, for example, we may specify that some operation be done in parallel, if it is a lengthy operation. Figure 1(b) shows a WFG obtained from the PAG in in Figure 1(a). In general, the WFG is obtained after careful examination of existing manufacturing methods and subsequent optimizations of the methods. In Section 6, we will discuss how some aspects of the process of optimizing a PAG to obtain a WFG are automated in our formalism.

Programs such as Archimedes [12] exist which take as input a CAD description of a product and produce a PAG for the product. Most of these algorithms operate by virtually disassembling the product, removing the easiest to remove piece first, to obtain an assembly tree. Information about the trajectory of the subassemblies is noted, so that it may be used to construct motion controllers later, by careful attention to the geometry of the product. Beyond that, work by

Wilson, [31], annotates the PAG with information about what tools to use and how they should be controlled, thereby bringing the PAG one step closer to being directly usable by the factory designer. Many other research groups have contributed various optimizations and augmentations of the basic algorithms to this field. The general focus, it seems, has been to provide feedback to the designer of the product about how design changes might lead to manufacturing changes – information that is very important to the marketability of the product. Less research has been done on translating the PAG, or the WFG, directly into a layout and program for a factory, although the STAAT program [28] produces elementary conveyer belt layouts, for instance. Some researchers have programmed workcells (six degree of freedom robot arms, for example), that can interpret some PAGs as programs for assembling the product. These workcells are slow and impractical at present, however, and more importantly, they do not take advantage of the distributed and parallel nature of assembly. A notable step in the direction of concurrency is the thesis of Bruce Romney, [27], in which assembly and fixturing (holding the subassembly in place), are considered as concurrent activities and planned for accordingly. In most implementations, however, a topological sort of the PAG is used to choose a *linear* sequence of assembly steps implying a purely sequential assembly process. In contrast, we will see that the larger, distributed robotic systems available to us will allow us to take advantage of the parallel nature of the assembly graph.

Another approach to synthesizing assembly controllers is given in [16]. For simple situations, there is an automatic method for constructing a control law which 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 at all obvious how this method can be extended to three dimensional systems with orientable parts. In this paper we take the view that the *product assembly graph* (PAG) of a product corresponds to a sort of discrete and parallelized version of such a potential function. The individual steps of the assembly – the nodes in the assembly graph – 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, more like what might be seen in a high volume factory setting.

In this paper we introduce a method for annotating the PAG, as though it were a parse tree, with many sequential processes. Then we will compose the processes according to the structure of the tree to produce a concurrent process which may be directly interpreted as a description of a complete working factory. We intend that this representation of the product be used directly to construct a factory thereby bypassing the time consuming process of programming the individual robots in the factory.

2.2 Petri Nets

One of the most basic tools for analyzing and designing factories, and in general specifying concurrent systems, has been the Petri Net. A good introduction is the book by Wolfgang Reisig, [23]. A summary of the essential definitions can also be found in Appendix II of this paper. Petri Nets are used to model everything from parallel processing computers and distributed systems to baroque

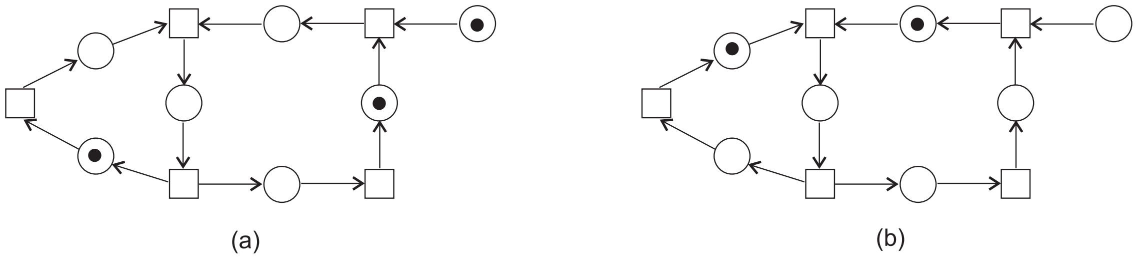

Figure 2: (a) An example of a Petri Net with tokens representing a marking. (b) A possible next marking.

style counterpoint, and the literature on the subject is vast. Furthermore, there are many types of Petri Net, seemingly one for every field to which they have been applied. We will limit ourselves to a very simple kind of Petri Net, the *condition/event net*, which has been studied extensively, as well as a subclass of these nets called *marked graphs*. We will augment the interpretation of a state (or marking) of a net with information about the continuous dynamics of the factory. In particular, when a condition holds in a run of our nets, we will take that to mean that some subset of the robots in the corresponding factory is functioning under a certain controller corresponding to the condition. An example of a condition/event net is shown in Figure 2.

We introduce the most basic definitions in this section and refer the reader to Appendix II for a slightly more detailed introduction. A simple Petri Net (or Condition/Event Net) is a graph consisting of *conditions* (or *places*), which are represented by circles, and *events* (or *transitions*), which are represented by squares. There may be a directed edge only between a condition and an event or between an event and a condition. A *marking* is a set of conditions that are said to hold during that marking. A marking thus gives the state of the system. A transition may *fire* in some marking if the conditions before it are represented in the marking and the conditions after it are not. The result is a new marking where the conditions before the transition are removed and the conditions after it are added to the marking. Figure 2 shows an example of this process.

We adopt a somewhat unconventional notation for Petri Nets which we believe is slightly more concise for our purposes. To the best of our knowledge this notation is introduced in [11]. A Petri Net is given by a pair (T, P) where T is a finite set of transitions and $P \subseteq 2^T \times 2^T$ is the set of places of the net. A place is thus represented by the transitions that come before it and the transitions that come after it. We write $[a_1, ..., a_i; b_1, ..., b_j] \in P$ to denote that $\{\{a_1, ..., a_i\}, \{b_1, ..., b_j\}\} \in P$. In the case that all places are of the form $[a; b]$, the net can be viewed simply as a graph where the transitions are nodes. Such a structure is called a *marked graph* and is studied in [7]. We will use some of what is known about marked graphs in Section 3 where we develop a class of Petri Nets which are composed of cycles, which we call gears.

Much of the research in Petri Nets is about analyzing nets to determine if they are live (every transition can eventually fire) and safe (no undesirable marking is reached). In contrast, we are concerned with building nets up from simple, sequential components, so that the resulting nets are guaranteed to have certain properties such as safety and liveness. People who have looked into composing Petri Nets are Kindler, [15] and Park et al., [21]. Kindler's work focuses on ensuring that the operational semantics of the components of a net compose as the nets do. Park et al. synthesize marked graphs for factories from sequential function charts. Our compositional method is also quite similar to those found in work on bottom-up synthesis of Petri Nets, especially [17] where simple Petri Nets are combined along paths and invariants of the resulting net are obtained from the constituent nets.

2.3 Distributed Manipulation and Cooperative Robotics

The manipulation of an object by more than one robot is an aspect of the fields of distributed manipulation and cooperative robotics. In distributed parts manipulation the emphasis tends to be on large numbers of simple, and usually tiny, actuators, such as MEMS ciliary structures [29] or air jets [3]. The programming of these arrays is usually done with programmable force fields [13] where each actuator in a rectangular array is assigned a direction to "push" so that the resulting array of directions has an equilibrium point in some desired place. Although the emphasis here is on parts feeding and not on assembly, the work is headed in that direction. Switching between force fields in a controlled manner and concurrent control will need to be addressed eventually.

In cooperative robotics, the emphasis is on coordinating the behaviors of a small number of large robots such as mobile robots, or robot arms. Many researchers have investigated this area and we will mention only a few. Khatib and his group at Stanford, for example, have developed techniques for controlling pairs of mobile robots equipped with robotic arms in the context of manipulating and carrying objects [14]. The emphasis is on the dynamics of manipulation and stability. Another interesting line of research is reported in [18] where the authors do on-line control of two robot arms in a shared workspace. The arms must avoid each other and obstacles while removing parts from a conveyer belt. In this paper two types of deadlock are considered – so called *computational deadlock* (which we will call *logical*) and *physical deadlock*. Logical deadlock occurs when one robot is waiting for another which is waiting for another and so on some cycle of mutual waiting and is akin the the kind of deadlock which can occur in an operating system environment. Physical deadlock occurs when a robot or subassembly are physically oriented in such a way that some other robot can not proceed with its task. Of course, if the physical space of the factory is correctly modeled, physical deadlock is really just another kind of logical deadlock wherein physical spaces are considered shared resources. The emphasis in [18] is on coordinating high level plans, however, as it is in much research on coordinated robotics. In the present research, however, we consider the bottom up synthesis of simple behaviors into an already coordinated, reactive, concurrent set of behaviors for the robots involved.

We will require the coordinated behavior of many robots as they go about the task of assembling products. Since communication and computational resources are at a premium in manufacturing settings, our robots will have to switch between partner robots so that control remains local. The local control of small groups along with the switching between groups must be done in such a way as to ensure that the system is live and that the product assembly processes are maintained.

2.4 Control Theory

In our formalism, robots will operate under continuous feedback control with discrete switching between control functions. In fact, we assume that the low level controllers for the devices we use exist already: our focus will be on switching among controllers taken from an already constructed palette of controllers.

The simplest way to specify the dynamics of a robot is with the formula

$$\dot{x} = f(x, u)$$

where $x : \mathbb{R} \to C$ is the state of the robot, C is the set of states the robot may take, $\dot{x}$ is its rate of change (i.e. $\frac{dx}{dt}$) and u is the control input. In reactive control, u is a function of x and so the robot must know its own state via sensors of some kind. Thus, the formula says that the change in x is a function of the state x and the controllable input u. When a robot with state x is cooperating with with another robot with state y, we have a system like

$$(\dot{x}, \dot{y}) = f(x, y, u_x, u_y)$$

where u_x, a function of x and y, is the input controllable by the first robot and u_y, also a function of x and y, is the input controllable by the second robot. In such a system, obviously, there must be some way for each robot to estimate the state of the other robot. We will provide for this kind of state sharing with an ideal communications link between the robots and an ideal sensor system and we will not consider the problem of state estimation in this paper. In the obvious way, this may be extended to the case of any number of robots and any other nonactive objects. Sometimes, to simplify matters, we consider the case where $f(x, u) = u$. In the appendix, we elaborate on these definitions and present other fundamental definitions of the field of dynamic systems and control.

In a factory situation especially, robots must switch between different controls (e.g. different fucntions u of x). Discrete switching between continuous dynamics fall under the general field of hybrid control. There are several formalisms for hybrid control systems. Here we will review one of the most common and then explain how the present research relates to it. A *Hybrid Automaton*, introduced in [1], is a discrete graph with information about continuous dynamics attached to the nodes and edges. Essentailly, to each node v a predicate on the state variable, such as $x \leq 1$ is given and an equation for the rate of change, such as $\dot{x} = x + 1$ is given. If the discrete state of the system is v, then as long as the predicate for v is true, then $\dot{x}$ will correspond to the equation for v. As soon as the predicate on x is false, the state changes to a neighbor of v in the graph whose predicate is true for x. In this manner, a piecewise continuous trajectory of the system can be given and some analysis can be done. Systems such as thermostats and train crossing gates are specified and proved correct, in [10] for example, by combining the above with model theory and modal logic.

In our formalism, we will also be concerned with discrete modes of control – the dynamics in different modes only changes because u changes. Essentially, we will consider a mode to be given by a goal, or attracting equilibrium point (x_{goal} such that $f(x_{goal}, u) = 0$) and a domain of attraction, that is, the set of all states for which u eventually drives the system to x_{goal}. We denote by $\mathcal{G}$ and $\mathcal{D}$ the goal set (usually a small open set around the goal point) and the domain of a controller. These notions are described formally in Appendix I. In Section 5.1 we will give a simple example of how such controllers can be designed. Once the control modes are designed, modes may be strung together in such a way that the goal of one controller is in the domain of the next, as in [6] where different juggling behaviors are composed. One of the contributions of the present paper is the extension of this idea of backchaining controllers to concurrent systems.

2.5 Compositional Control

Preimage backchaining was introduced into the motion planning literature in [19] as a method of sequentially composing motion strategies. In [5] this method was extended to robot juggling in work that serves as the basis of our current research. The idea is to start with a palette of controllers $\Phi_1, ..., \Phi_n$ for a robot. Suppose Φ_k has domain $\mathcal{D}_k$ and goal $\mathcal{G}_k$. Order the palette by setting $\Phi_i \succeq \Phi_j$ (read Φ_i *prepares* Φ_j) whenever $\mathcal{G}_i \subseteq \mathcal{D}_j$. If the palette is suitably designed, then a switching strategy may be obtained which drives the robot to a goal from any initial condition in $\bigcup_{i=1}^{n} \mathcal{D}_i$. In the work on juggling, this technique was used to switch between catching, juggling, palming and tossing behaviors, resulting in a quite sophisticated overall behavior. In this paper we expand these ideas to include the notion of concurrent composition of behaviors for the case of several robots in a shared workspace.

One quality we would like our factory to have is smoothness of motion. We recognize that humans perform tasks with a fluidity that roboticists can only poorly approximate. We believe that the road to building such robots is paved with a theory of composing control laws or more generally, dynamic systems in various ways. We have already discussed the sequential composition, but this is just the beginning. More elusive is the parallel composition of controllers of coupled systems. In the present paper, we consider loosely coupled systems and discrete composition. Although, we do not claim to have a theory of dynamic, smooth concurrent systems, we do believe that we are headed in that direction. In this section, we will review the important thread of research in robot juggling.

In [4], Bühler and Koditschek present a 2-dimensional juggling robot: a 1 degree of freedom arm leaned against a wall capeable of bouncing a ball repeatedly to some specified height. They also experimented with two balls at once. Later Rizzi and Koditschek built a three degree of freedom arm capable of bouncing a ball in three dimensions, [26]. As if this wasn't enough, Rizzi then expanded this work to control two balls at once, [25]. To accopmplish this, Rizzi essentially combined two copies of a one ball juggle in parallel using notions of urgency and ball phase to control the two balls so that they (1) did not hit each other and (2) were attracted toward being 180° out of phase. Although the juggler proved to be experimentally sound with this task, a theoretical treatment remains elusive.

To explain the relavance of the research just described, consder the following situation. Suppose we have two robots, say C_1 and C_2 which carry subassemblies from place to place in a factory and that we have a third robot G whose job it is to put some glue on any subassembly brought to it by C_1 or C_2. Since C_1 and C_2 have other things to do as well, the strategy for G is to attempt to never keep either C_1 or C_2 waiting. We propose that to do this, it must "juggle" between the task of servicing C_1 and C_2. We belive that the present research is the discrete version of this problem.

2.6 The Minifactory

Finally we review what has actually been the inspiration for the present research all along. At the Microdynamic Systems Laboratory at Carnegie Mellon University, a modular, reconfigurable robotic factory system, called the Minifactory, is being developed, [20], [22], along with a considerable software support and simulation system, [9]. The Minifactory project is a collection of modular robots and other components which, in theory at least, may be assembled quickly and programmed almost as quickly into a factory which can assemble some small product, usually electronic and very precisely specified. In practice, many of these details have yet to be worked out.

The main structure in a Minifactory is a set of *platens* which serve as the factory floor. On the platens, robots called *couriers* float on bearings of air and move around in the factory with subassemblies on their backs. These couriers are very precise and can navigate the platens to within a micron's accuracy. Mounted above the platens are various types of manipulators and parts feeders which can insert parts and perform other simple operations on the subassemblies being carried about by the couriers. Each robot is controlled separately by its own computer running a real time operating system and the computers are connected to each other in a sort of parallel or distributed architecture by high speed ethernet switches, also provided by the minifactory architecture. See Figure 3 for a picture of an example minifactory. Note that the couriers replace the conveyer belts in a traditional factory, allowing fewer restrictions on the paths of subassemblies between manipulators.

The simulation software and programming environment for the Minifactory is called the *Architecture for Agile Assembly*, or AAA. AAA provides a fairly complete simulation of the factory and lets the user easily reconfigure, move and edit the programs of robots within the factory. However, at this point, the programming of robots is a difficult, time consuming process. It is based on an object oriented communications system and factory reservation areas as shared objects. Fairly low level knowledge of how the robots are controlled is needed to program them successfully. Current work on AAA includes constructing a palette of controllers from which the programmer may choose low level behaviors and on augmenting the simulation and user interface.

The minifactory idea poses many challenges. First of all, there is no general and formal method by which concurrent robotic systems, which may block each other logically *and* physically as we

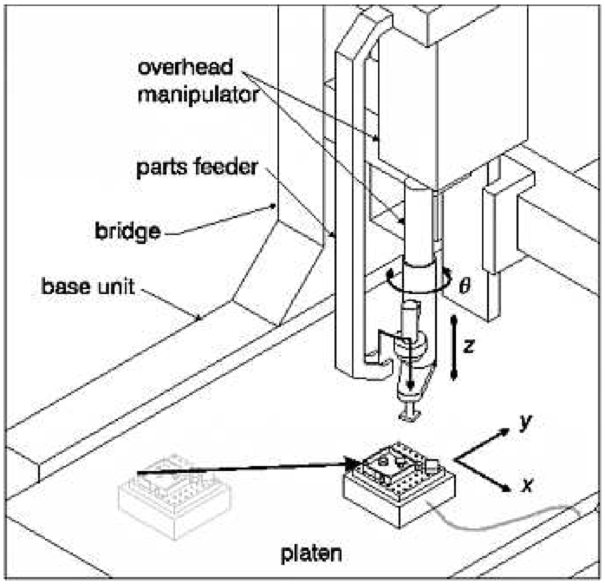

Figure 3: A schematic of a part of a Minifactory taken from [22].

mentioned in Section 2.3, can be constructed. What is required is a provably correct means of constructing robot programs so that the global behavior of the factory is ensured. Second, in order for the Minifactory paradigm to be viable, a much simpler method of obtaining programs and layouts for the robots in the factory, other than the creativity of the specialist, must be found.

3 Gear Nets

In this section we consider how to compose sequential Petri Net components, each representing the behavior of a robot, in such a way that the resulting net represents the combined behavior of the robots. Such a thing is called a *compositional semantics*. It requires an sort of standard interface for combining nets in such a way that the semantics (what the net does) of the resulting net can be obtained in exactly one way from the semantics of the components. Here we give a compositional semantics for a very simple class of Petri Nets called *gear nets*, which are a kind of marked graph. These simple nets form the basis for a more complete structure which we introduce in Section 4. Gear nets only describe the discrete states of the robots involved in the factory and do not include information about the state of any partially assembled products or about the low level dynamics of the factory. First we describe gears, then gear nets and their relation to marked graphs, and finally the properties of gear nets which allow us to compose gears.

The simplest thing a robot in a factory can do, besides remain idle, is to cycle repeatedly

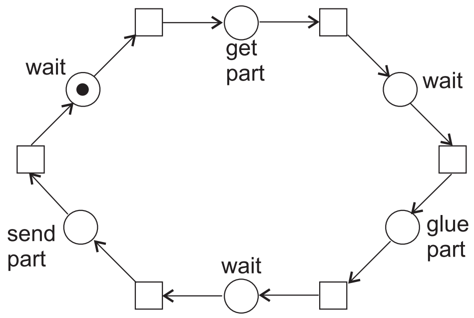

Figure 4: A gear corresponding to a simple, sequential program for a factory robot.

through some set of behaviors. A robot might, for example, (1) pick up a part at a parts feeder, (2) bring the part to a station to be glued to another part, (3) take the result to a manipulator to be added to some other subassembly and then start the sequence again. Thus our most basic Petri Net component is a cycle, which we choose to call a gear, and which represents the sequential program that a robot repeatedly executes during a run of a factory. Formally, we have

Definition 3.1 *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 $|m| = 1$.*

(Recall the definition of a Petri Net given in Appendix II). Figure 4 depicts a gear corresponding to the example just given. The places of a gear correspond to the control mode of the robot. Obviously, given an initial state (control mode), a gear has only one kind of process, namely a linearly ordered one. We point this out with the following property and will make use of it later in this section when we compose the processes of gears to obtain a concurrent process.

Property 3.1 *Any process for a gear is totally ordered.*

Notice that the control modes of the gear shown in Figure 4 tell only what the single robot in question is doing while, in fact, the robot must coordinate with the controller of the glue station and with the manipulator in order to function correctly. Thus, the program of the robot must be synchronized with the programs of other robots. Of course, the programs of other robots are also given by gears. What is needed is a means by which gears are composed, so that any control modes that any robots must execute in synch with each other, are identified. Furthermore, it is important that each robot involved in a control mode wait, before entering the mode, for the other robots involved. With these constraints in mind, we are led to a definition of a gear net as the union of gears. However, we must be careful. Not any union will do. One problem is that arbitrary unions of cycles can introduce spurious cycles into the unions, possibly resulting in deadlock situations. See Figure 5 for an example of this. Therefore, our definition is more careful.

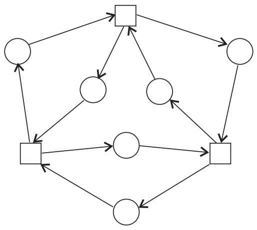

Figure 5: A union of gears that is not a gear net.

Definition 3.2 *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. See Figure 6 for an example of the construction of a gear net in which the inductive nature of the definition is illustrated. Note that a legal marking gives the state of every gear in the gear net. This corresponds to the fact that each robot is in exactly one state in its program. Conditions (a) and (b) require that gears be added with a "standard interface". This ensures that the nets remain deadlock free. Before we prove this, and also justify the added definition of legal marking, we point out some facts about marked graphs noted in [7]. First, define a marking m to be **live** if there is a transition e such that $^{\bullet}e \subseteq m$. We have

1. If (S, Q) is a cycle in a marked graph and $m \rightarrow^{G} m'$ then $|m \cap Q| = |m' \cap Q|$. That is, transition firing does not change the size of markings on cycles.
2. A marking m of a marked graph is live if and only $|m \cap Q| > 0$ for all directed cycles (S, Q) in the graph.
3. If m is live and $m \rightarrow^{G} m'$, then m' is live.

Next we prove that gear nets are live. To do so we need an auxiliary result which states that we do not add spurious cycles as we build up gear nets in Definition 3.2.

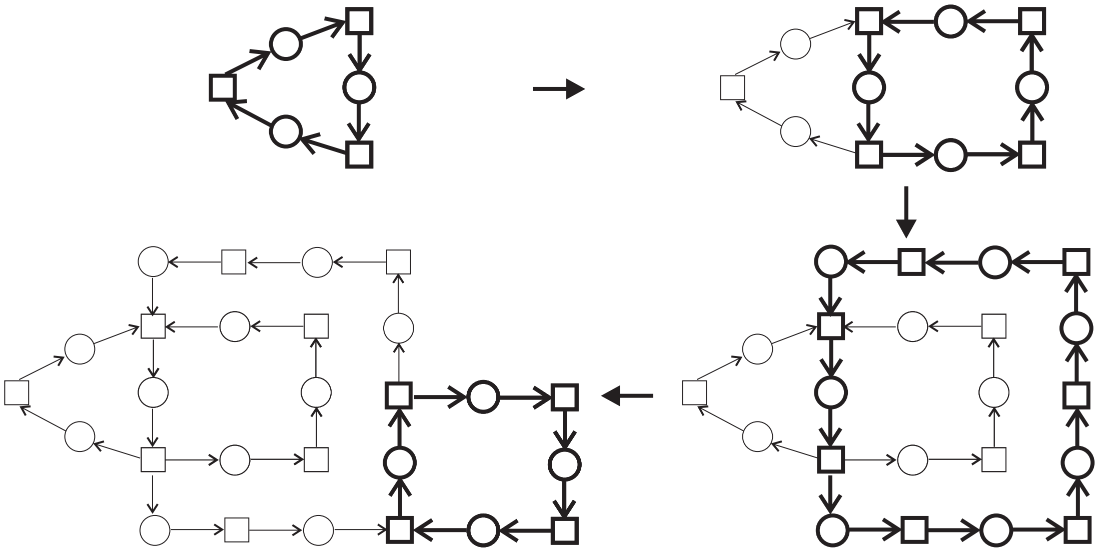

Figure 6: The iterative process of constructing a gear net from simple gears.

Proposition 3.1 *If (T, P) and (S, Q) are as in Definition 3.2 and (S', Q') is a gear contained in $(T \cup S, P \cup Q)$, then either $(S', Q') = (S, Q)$ or (S', Q') is contained entirely within (T, P).*

Proof: Suppose that (S', Q') is not contained entirely in (T, P). Then there is some transition in (S, Q) that is also in (S', Q'). Thus, the transition c which is unique to (S, Q) and which precedes a in the definition must be in (S', Q'). Since $a^{\bullet} = [a; b]$ (otherwise the definition fails), (S', Q') must contain the path c, a, b. Now, after b must come some transition in (S, Q) since the cycle (S', Q') must return to c which is unique to (S, Q). It follows that $(S', Q') = (S, Q)$. $\Box$

Next we have our first main result, from which it follows that gear nets are deadlock free.

Proposition 3.2 *Every gear net has a legal marking.*

Proof: The result is obvious for simple gears. Note that by the previous proposition, we need only consider the gears we add in the definition when we go to the inductive step. Now suppose that (T, P) is a gear net which has a legal marking m and (S, Q) is a gear for which the definition of gear net holds. If $m \cap Q = \emptyset$, then take any $p \in Q - P$. It follows that $m \cup \{p\}$ is a legal marking for $(T \cup S, P \cup Q)$. Otherwise, m must mark (S, Q) exactly once, by condition (1) of the definition, so m is a legal marking of $(T \cup S, P \cup Q)$. $\Box$

These propositions, together with the facts about marked graphs that we presented, are enough to show that gear nets are deadlock free. We know there is one legal marking for any gear net. By fact (1), legal markings lead to legal markings. Fact (2) gives us that gear nets are live under legal markings. Thus, we have:

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

The next Theorem says that gear nets are reversible. This means that any initial (legal) marking we may choose is reachable via detached sets of events from any other legal marking. Formally, say that for a Petri Net (T, P) and a marking m, define the set $R(m)$ to be the set of all markings m' such that there exists a sequence of markings $m_0, ..., m_k$ with $m_0 = m$ and $m_k = m'$ and a sequence of detached sets of events $G_0, ..., G_{k-1}$ such that

$$m_0 \rightarrow^{G_0} ... \rightarrow^{G_{k-1}} m_k \ .$$

Then we have

Theorem 3.2 *(Reversibility) Given an initial marking m_0 of a gear net (T, P), $m_0 \in R(m)$ for all legal markings m of (T, P).*

Proof: We proceed by induction on the form of the gear net. Certainly a gear is reversable given any initial marking. Now suppose that (T, P) and (S, Q) are as in the definition of gear net and that m_0 is the initial (legal) marking of $(T, P) \cup (S, Q)$. Then there are legal markings $m_1, ..., m_k$ of (T, P) and detached sets of events $G_0, ..., G_{k-1}$ such that

$$m_0 \cap P \rightarrow^{G_0} m_1 \rightarrow^{G_1} ... \rightarrow^{G_{k-1}} m_k = m \cap P$$

for any marking m of $(T, P) \cup (S, Q)$. That is, $m_0 \cap P \in R(m \cap P)$. Now we can construct a sequence from this sequence to show that $m_0 \in R(m)$. Suppose that p is the single place in $m_0 \cap Q$. Add it to all markings m_1 through m_j, where G_j contains a (from the definition). If $[c : a]$ (that is, $p \neq [c : a]$) is not in m_j, then we can find a sequence of transitions $t_1, ..., t_l \in S$ to take q to $[c : a]$. Then

$$m_j \cup \{p\} \rightarrow^{t_1} ... \rightarrow^{t_l} m_j \cup \{[c : a]\}$$

"patches up" the sequence. Further such adjustments to the rest of the sequence eventually lead back to m. Thus, $m_0 \in R(m)$. □

Now that we have shown that gear nets are deadlock free and reversible, are we guaranteed that factories so composed are correct? We are as far as the logic of the programs is concerned. However, we will also need to consider the dynamics of the gear net (the processes they admit), so that we can introduce parts lines later in Section 4. It turns out the the kind of processes (semantics) that a gear net admits is directly related to the way the net was composed (syntax). In the rest of this section we elaborate on this. In [15], Kindler presents a compositional semantics for Petri Nets based on the idea that the semantics of a Petri Net is given by the set of closed (complete) processes for that net. Here we present a less sophisticated idea enabled by the fact that our gear nets are so simple. Noting the fact that each gear has a linear process – recall Property 3.1 – we simply "glue" the processes for gears together in a way similar to the way we glued the gears themselves together.

For course, before we can assert that we have a process for a gear net, we must be certain that gear nets are contact free. This is given to us by the definition of a legal marking and by fact (1): if a gear net were to admit contact under some marking m, then there would be a gear (S,Q) $|Q \cap m| > 1$, which is a contradiction.

Now, suppose we have a gear net (T,P) and a gear (S,Q) such that $(T \cup S, P \cup Q)$ is a gear net with intersection $\{a, b, [a;b]\}$. Say that (U_1, K_1, σ_1) is a process for (T,P). Also say that (U_2, K_2, σ_2) is a process for (S,Q) with as many occurrences of $[a;b]$ as (U_1, K_1, σ_1) and minimal element mapping to a place unique to (S,Q). We assume these two processes are disjoint. Then we can construct a process for $(T \cup S, P \cup Q)$ as follows.

Definition 3.3 *For a process (U,K,σ) and an element $a \in U \cup K$, define the* **occurrence** *of a as $occ(a) = |\{\ b \sqsubseteq a\ :\ a \neq b \wedge \sigma(b) = \sigma(a)\ \}|$.*

Definition 3.4 *For any $i, j \in \{1,2\}$, any $a \in U_i \cup K_i$, and any $b \in U_j \cup K_j$, say that $a \sim b$ whenever $\sigma_i(a) = \sigma_j(b)$ and $occ(a) = occ(b)$.*

We can then look at the equivalence classes of $\sim$ as a process for $(T \cup S, P \cup Q)$. Formally, denote by $[x]$ the equivalence class of x with respect to $\sim$. We construct a net (U,K,σ) as follows

$$\begin{aligned} U &= (U_1 \cup U_2)/\sim \\ K &= \{[[x];[y]] \mid \exists j \in \{1,2\} \wedge \exists a \in [x] \cap U_j \wedge \exists b \in [y] \cap U_j \wedge [a;b] \in K_j\} \\ \sigma([a]) &= \sigma_i(a) \end{aligned}$$

where i in the definition of σ is chosen such that $a \in U_j \cup K_j$. U is just the set of equivalence classes of $\sim$ restricted to transitions. K is the set of all pairs from U which contain pairs in one of the original processes. Finally, since $\sigma_i(x)$ is the same for all x in any particular equivalence class, we may simply choose one of the representatives from the class and use the σ_i that corresponds to it for the whole class.

The rest of this section is devoted to proving that this is a process for $(T \cup S, P \cup Q)$. First we have a property of the occurrence number of an element of a process.

Property 3.2 *If (U,K,σ) is a process for a Petri Net (T,P) and $[x;y] \in K$, then $occ(x) \leq occ(y)$.*

Proof: Let $A \doteq \{a \sqsubseteq x \mid a \neq x \wedge \sigma(a) = \sigma(x)\}$. Then $occ(x) = |A|$. If $a \in A$ and $[a;b] \in K$, then $\sigma(b) = \sigma(y)$ since processes preserve the flow relation. Now, the set $B \doteq \{b \mid \exists a \in A, [a;b] \in K\}$ is the same size as A. We claim that $occ(y) = |B|$ or $|B|+1$. If there is no b such that $b \sqsubseteq \bigwedge A$, then $B' \doteq \{b \mid b \neq y \wedge \sigma(b) = \sigma(y)\} = B$. Otherwise, $B' = B \cup \{b\}$ where b is the single element such that $b \sqsubseteq \bigwedge A$. $\square$

Next, we show that (U,K,σ) is an occurrence net which amounts to showing it is acyclic.

Lemma 3.1 *If (U, K, σ) is constructed from (U_1, K_1, σ_1) and (U_2, K_2, σ_2) and $\sim$ as above, then (U, K, σ) is an occurrence net.*

Proof: That $| {}^\bullet[p]| \leq 1$ and $|[p]^\bullet| \leq 1$ is obvious from the definition of (U, K, σ). We must prove that (U, K, σ) is acyclic, which we do by contradiction.

To that end, suppose that $[t_0], ..., [t_k]$ are transitions forming a simple cycle. First we will show that it must be the case that $occ([t_1]) = ... = occ([t_k])$. We know there are $x_1, x'_1, x_2, x'_2, ..., x_k, x'_k$ such that for each i, $x_i, x'_i \in [t_i]$ and $[x'_i; x_{i+1}] \in K_1 \cup K_2$, where the subscripts are taken modulo k. Thus,

$$occ(x_1) \leq occ(x_2) \leq ... \leq occ(x_k) \leq occ(x_1)$$

from which it follows that $occ([t_1]) = ... = occ([t_k])$.

Now, because (U_1, K_1, σ_1) and (U_2, K_2, σ_2) are processes, the cycle can not be contained entirely within either, and thus there must be a path $[t_{j_1}], ..., [t_{j_l}]$ in the cycle with $[t_{j_i}] \cap U_2 \neq \emptyset$ for $i \in \{1, ..., l\}$. Furthermore, we can suppose that $\sigma([t_{j_1}]) = \sigma([t_{j_l}]) = b$ (where b is as in the definition of gear net). Now, since the occurrence number of all the elements in the path is the same, it must be that $t_{j_1} \sim t_{j_l}$ and so $[t_{j_1}] = [t_{j_l}]$. It follows that $[t_{j_1}], ..., [t_{j_l}]$ actually forms a cycle when restricted to (U_2, K_2, σ_2) which is a contradiction. □

Now we are ready for the main result:

Theorem 3.3 *(U, K, σ) is a process for $(T \cup S, P \cup Q)$.*

Proof: We have already shown that (U, K, σ) is an occurrence net. We must show the extra conditions on σ.

First, if D is a slice of (U, K, σ), then $\sigma | D$ is injective. To prove this, suppose that $\sigma([x]) = \sigma([y])$ but $[x] \neq [y]$ for some $[x], [y] \in D$. Without loss of generality, we may assume that $[x] \cap K_1 \neq \emptyset$ and $[y] \cap K_2 \neq \emptyset$. But the only way this can be so is if $\sigma([x]) = \sigma([y]) = [a; b]$ where $[a; b]$ is as in the definition of gear net. Now, clearly, all elements which map to $[a; b]$ are linearly ordered (since they are in (U_2, K_2, σ_2) for example), and thus $[x]$ and $[y]$ can not be in the same slice – which is contradiction.

Second, if D is a slice then $\sigma(D)$ is a legal marking. To prove this, notice that there is only one element $[x] \in D$ such that $[x] \cap K_2 \neq \emptyset$, for otherwise D would not be a slice. Now since $\sigma(\{[x]\})$ marks (S, Q) , it follows that $\sigma(D)$ marks (S, Q) exactly once. Now, if $\sigma([x]) = [a; b]$ then $\sigma(D)$ is a legal marking for (T, P) and otherwise $\sigma(D - \{[x]\})$ is a legal marking for (T, P).

Third, $\sigma({}^\bullet[p]) = {}^\bullet\sigma([p])$ and $\sigma([p]^\bullet) = \sigma([p])^\bullet$. This is straightforward and we will show, as an example, the inclusion $\sigma({}^\bullet[p]) \subseteq {}^\bullet\sigma([p])$. Say $\sigma([q]) \in \sigma({}^\bullet[p])$. Then $[q] \in {}^\bullet[p]$. We can choose $i \in \{1, 2\}$ such that there are elements $p', q' \in K_i$ and $p' \in [p]$ and $q' \in [q]$. Thus, $[q'; p'] \in K_i$ which implies that $[\sigma(q'); \sigma(p')] \in P_i$. But then $[\sigma([q]); \sigma([p])] \in P_i$ or equivalently $\sigma([q]) \in {}^\bullet\sigma([p])$.

These three properties along with Lemma 3.1 give the desired result. □

Figure 7 illustrates this theorem by showing a gear net and a gear, their corresponding processes and the composition of each pair.

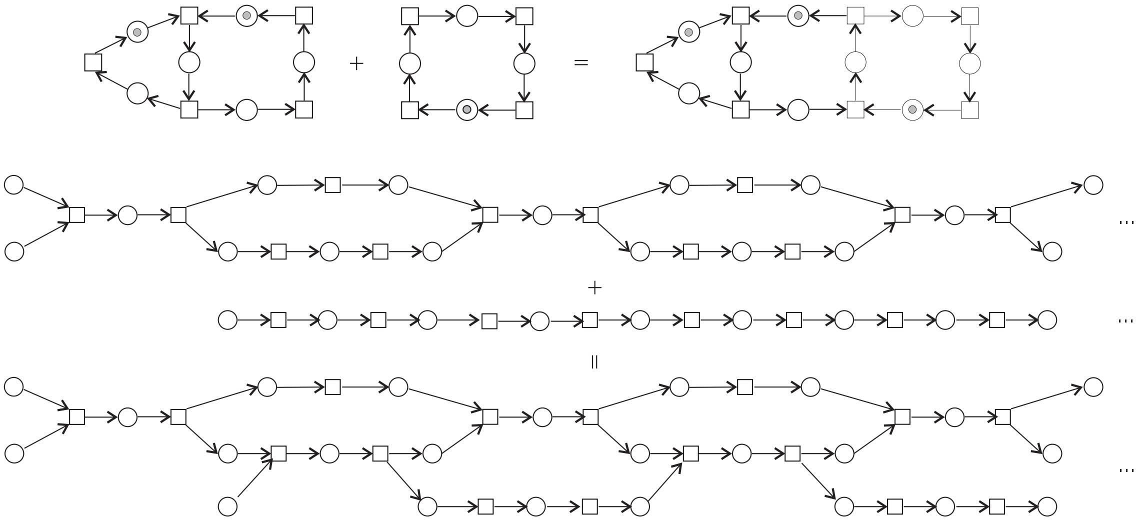

Figure 7: The composition of a gear net and a gear along with the corresponding composition for their processes.

3.1 Sources and Sinks

In this section we add places to simple Petri Nets which serve as *sources* and *sinks* and which, in later sections, we will use to represent *parts feeders* and *output buffers*. A source is a place that is almost always marked and which remains marked even after a marking containing it transitions to another marking. A sink is a place that is never marked, even when it is in the postset of a transition that has just fired. These notions are common in the Petri Net literature, but since we will make slight modifications to them for our use, we present the definitions here.

Definition 3.5 *A* **Petri Net with sources and sinks** *is a Petri Net* (T, P') *where* P' *is a disjoint union of sets* P*,* P_{source}*, and* P_{sink}*. All places in* P_{source} *are of the form* $[\emptyset, t]$ *for some transition* $t \in T$ *and all places in* P_{sink} *are of the form* $[s, \emptyset]$ *for some transition* $s \in T$*.*

As we described, the firing rule for these nets is slightly different.

Definition 3.6 *Let* $m \subseteq P'$ *be a marking for a Petri Net with sources and sinks* (T, P') *and let* $G \subseteq T$ *be a detached set of* m*-enabled events. The* **follower marking** *of* m *with respect to* G *is the marking:*

$$m' = [(m - {}^{\bullet}G) \cup G^{\bullet} \cup (P_{source} - {}^{\bullet}G)] - P_{sink}$$

.

Essentially, this definition is the same as the definition of follower marking (Definition 7.5) except that (1) with the term $(P_{source} - {}^{\bullet}G)$ we add to m' all sources except those which would cause contact and (2) by always subtracting P_{sink}, we ensure that sinks are never part of a marking.

We can use sources and sinks in gear nets to augment them so that they model factories with parts lines by placing sources on those gears which correspond to robots which pick up parts and sinks on those robots which deposit parts in parts feeders. Since sources are always marked, except right after they have caused a transition to fire, they do not affect the liveness of the gear they are attached to nor do they affect the liveness of the gear net. Similarly, sinks do not ever cause contact, so they never prevent any transition from firing that would have fired in their absense. Thus, gear nets with sources and sinks are also deadlock free.

4 Threaded Petri Nets

Although condition/event nets allow us to express certain aspects of the behavior of a factory, they are not enough. Missing are the details of what the robots and subassemblies are doing between transitions of the net – where they are physically and what part is being manipulated by which robot. We use the term *transient machine* to mean a temporary coupling of some number of robots (possibly only one) and parts (possibly none) in a controlled dynamic system. In this section we add, to the basic notion of a net, machinery to keep track of which transient machines are active and when. The resulting structure we call a *Threaded Petri Net* (TPN). From a TPN, we are able to obtain three different views of the factory: the factory centric view, in which we may analyze the global factory dynamics, is given by slices of the hybrid process of the net; the robot-centric view, in which we may analyze the dynamics of an individual robot, is given by sequential components of the net and their linear processes; and the product centric view, in which the trajectory of a subassembly, as it is passed from robot to robot, is traced through the factory process. Our intention is to build certain types of TPNs and use the ideas in this section to prove that they assemble products correctly. We stress that the additions we make here do not affect the liveness of the underlying net, as we will show.

We assume that the entire state of the factory, robot states and subassembly states, is given by a vector $x \in \mathbb{R}^n$. We let $N = \{1, ..., n\}$. For simplicity we assume that the state of each factory entity i is given by a single, one dimensional component of x, say x_i. The first addition we make is to associate with each place in the net a dynamic system, or transient machine, on some subset of the components of x. Thus, given a Petri Net (T, P), we will have for each $p \in P$, a dimension $l_p \leq n$, a function $F_p : \mathbb{R}^{l_p} \to \mathbb{R}^{l_p}$ which gives the dynamics of the transient machine, and the domain and goal of F_p which we will denote $\mathcal{D}_p$ and $\mathcal{G}_p$ respectively (see Appendix I for a more thorough explanation of these terms). Later, more restrictions on F_p will be given. We must have that for any two transient machines which may operate concurrently, the dynamic systems corresponding to the machines must be decoupled. This detail is dependent on the physical model of the robotic system, and need not be considered in this section, which is more general. We use this idea of a place in the following definition:

Definition 4.1 *A* **Threaded Petri Net** *consists of*

1. *A set T of transitions;*
2. *A set $P \subseteq 2^T \times 2^T$ of places;*
3. *For each $p \in P$, dimension, 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***;*

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 degree of freedom as mode changes occur.

Figure 8 represents a TPN for a three robot brigade (see Subsection 5.1) with a parts feeder and an output bin. The underlying net is constructed from a gear net with a parts line added. Grey lines are meant to represent the redistribution function of each transition. By following a line from a place p, through a transition e, and to another place q, the mapping d_e on that line is obtained.

We are now in a position to redefine what a marking is for our new kind of net. In addition to specifying which modes are active, we must say which degrees of freedom in N they are acting on.

Definition 4.2 *A* **marking** *is a pair (m, f_m) where $m \subseteq P$ and*

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

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.*

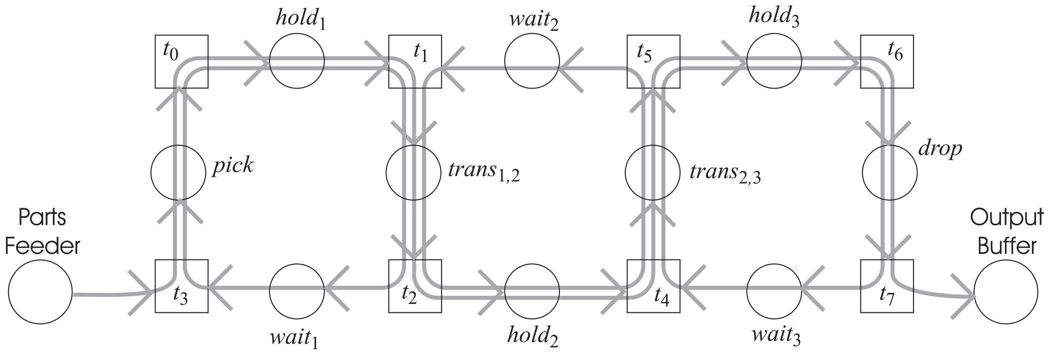

Figure 8: Schematic of a TPN Model of a A Three Robot Brigade (See Subsection 5.1). Grey lines illustrate the redistribution functions of the transitions.

Now we can say how the state of the system is changing given a particular marking (m, f_m). Given $j \in N = \{1, .., n\}$, suppose that $f_m^{-1}(j) = (p, i)$. That is, under the marking (m, f_m) the jth component of x is changing according to the ith dimension of the mode dynamics of p. Then,

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

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

Definition 4.3 *Let (m, f_m) be a legal marking. $e \in T$ is m-***enabled** *with respect to $x \in \mathbb{R}^n$ 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. These two conditions do not affect the logical dynamics of the underlying net – they simply require the designer of the system to ensure that the transient machines constructed for the places have goals and domains so that events *can* become enabled. We will present an example of such a system in Section 5.1. For now, we will just point out that this is our first attempt at distinguishing between *logical* and *physical* deadlock.

Now suppose we have a marking (m, f_m). If we have, with respect to some $x \in \mathbb{R}^n$ a detached set of m-enabled events $G \subseteq E$, the **follower marking** $(m', f_{m'})$ 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 $p^\bullet \cap G$. Since legal markings (m, f_m) are such that f_m is bijective, we can be sure that *every* state variable $x \in \mathbb{R}^n$ is accounted for when the system is in the set of transient machines given by m. We would also like that only legal markings are reachable from given legal markings, so that once the distributed process is underway, there is no point at which some part of x is not acting under a mode of the net. The following property gives us this.

Property 4.1 *Say (m, f_m) is a legal marking, that G is a detached set of events and that G is m-enabled (with respect to some x; it doesn't matter for this property). If $(m, f_m) \rightarrow^G (m', f_{m'})$, then $(m', f_{m'})$ is also a legal marking.*

Proof: We show that the domain and codomain of $f_{m'}$ have the same cardinality and that $f_{m'}$ is surjective from which it follows that $f_{m'}$ is bijective. Note that since f_m is bijective, $|dom(f_m)| = \sum_{p \in m} l_p = |N|$. Now,

$$|dom(f_{m'})| = \sum_{p \in m'} l_p = \sum_{p \in m} l_p - \sum_{p \in {}^\bullet G} l_p + \sum_{p \in G^\bullet} l_p = \sum_{p \in m} l_p + 0$$

since ${}^\bullet G \cap G^\bullet = \emptyset$ and by the final condition in the definition of transient machine net. Thus, the domain and codomain of $f_{m'}$ are the same size. Next we show, directly from the definition of $f_{m'}$, that $f_{m'}$ is surjective. First, suppose that

$$j \in f_m(\bigcup_{p \in m - {}^\bullet G} \{p\} \times \{1, ..., l_p\}).$$

Then $f_m^{-1}(j)$ is the preimage of j under $f_{m'}$. Otherwise, $d_e \circ f_m^{-1}(j)$ is the preimage of j under $f_{m'}$. $\Box$

Finally, we define what a process is for a TPN. We simply add to the definition of process a summary of the redistribution functions.

Definition 4.4 *A* **threaded process** *for a TPN (T, P) with initial marking (m_0, f_{m_0}) is a quadruple (U, K, σ, d) where (U, K, σ) is an occurrence net with dimensions and redistribution functions, $\sigma : U \cup K \rightarrow T \cup P$ and $d : \bigcup_{p \in K} \{p\} \times \{1, ..., l_p\} \rightarrow N$, such that*

1. *(U, K, σ) is a process for (T, P);*

2. *for each slice D of $(U \cup K, \sqsubseteq)$, suppose that*

$$(m_0, f_{m_0}) \rightarrow^{G_1}, ..., \rightarrow^{G_n} (m_{\sigma(D)}, f_{\sigma(D)}).$$

Then the function d' defined by the restriction of d to $\bigcup_{p \in D}\{p\} \times \{1, ..., l_p\}$ is such that $d'(p,j) = f_{\sigma(D)}(\sigma(p), j)$.

Figure 9 shows a threaded process for the net in Figure 8. There are several immediate and satisfying properties of threaded processes. First, slices of $(U \cup K, \sqsubseteq)$ are legal markings. Formally:

Property 4.2 *Let D be a slice and consider the marking $(\sigma(D), f)$ where $f(p,j) = d(q,j)$ whenever $q \in D$ and $\sigma(q) = p$ and $1 \leq j \leq l_p$. Then $(\sigma(D), f)$ is a legal marking.*

Proof: This follows directly from condition (2) of the definition of threaded process. □

The next theorem characterizes the subprocess of a threaded process that corresponds to a particular component x_i of x. We prove that such a subprocess is a convex, linearly ordered subset of a threaded process. This means that the subnet of a TPN corresponding to the component is a sequential subprocess. We will use form of this theorem later to show that parts lines in factories indeed deliver parts from parts feeders to their destinations.

Theorem 4.1 *Let (U, K, σ, d) be a threaded process for a TPN (T, P) and let j be an index in $N = \{1, ..., n\}$. Then*

1. *The set $X_i = \{p \in K \mid d(p,j) = i$ for some $j \in \{1, ..., l_p\}\}$ is totally ordered in $(U \cup K, \sqsubseteq)$ and*

2. *if p_1 and p_2 are elements of X_i such that $p_1 \sqsubseteq p_2$ then whenever $p_1 \sqsubseteq q \sqsubseteq p_2$, we have that $q \in X_i$.*

Proof: First, suppose that $p, q \in K$ are such that $d(p,j) = d(q,k)$ for some k, j and that, to the contrary, $p \,||\, q$. Then there is a slice D of $(U \cup K, \sqsubseteq)$ with $p, q \in D$. Let d' be the restriction of d as in Definition 4.4. Since $d'(p,j) = f_{\sigma(D)}(\sigma(p), j)$ and $d'(q,k) = f_{\sigma(D)}(\sigma(q), k)$ and since $\sigma(p) \neq \sigma(q)$ (otherwise they would be related under $\sqsubseteq$), we have that $f_{\sigma(D)}$ is not a bijective and thus not a legal marking which is a contradiction. Thus, X_i is totally ordered.

Next suppose that $p \in X_i$ is not a maximal element of $(U \cup K, \sqsubseteq)$. We show that there is a $q \in X_i$ such that for some $e \in U$, $e \in p^\bullet$ and $q \in e^\bullet$ from which (2) follows. First let D be a slice with $p \in D$ and let e be the single event in $p^\bullet$ (unique by definition of occurrence net). Now we progress D by firing e. Let D' be the slice $(D - \{p\}) \cup e^\bullet$. Since $d_{\sigma(e)}$ is bijective, it must be that the index i is mapped to some place in $\sigma(D')$, call it $\sigma(q)$. Thus, $q \in e^\bullet$. □

Evidence of the truth of this theorem can be seen in Figure 9 where, for example, we can follow each index from the bottom of the process up along a line in the process. The following corollary expresses this theorem in terms of TPNs only.

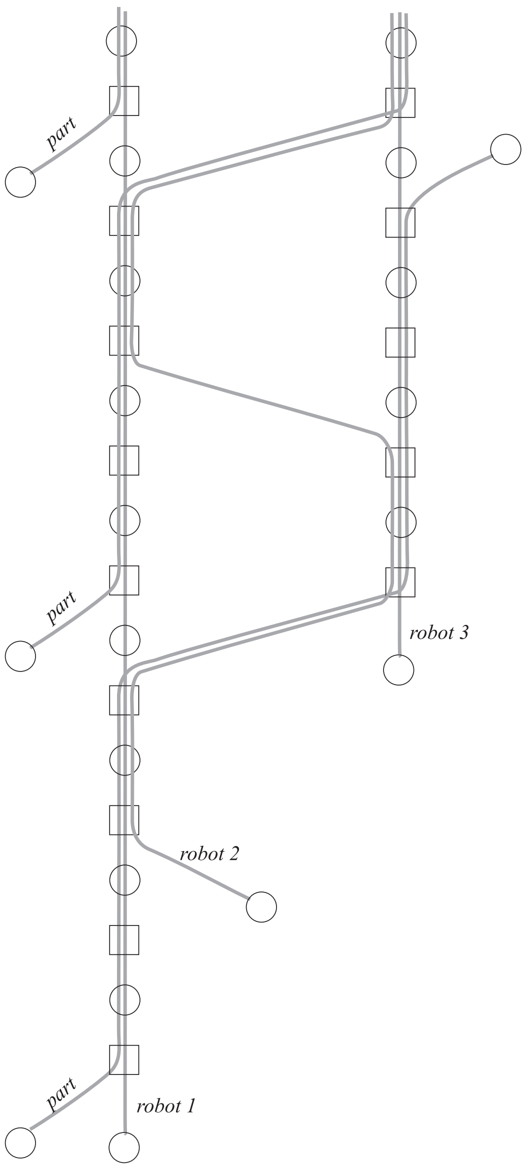

Figure 9: A threaded process for the hybrid net shown in Figure 8

Corollary 4.1 *Suppose that*

$$(m_0, f_{m_0}) \to^{G_1} \ldots \to^{G_k} (m_k, f_{m_k})$$

is a run of a TPN (T, P)*. Given* $i \in N$*, let* $p_j \in P$ *be the place in* m_j *such that* $f_{m_j}(p_j, l) = i$ *for some* l*. Then* $p_j \in {}^{\bullet}p_{j+1}$ *for each* $j \in \{0, \ldots, k-1\}$*.*

5 Examples

In this section we describe two examples which demonstrate the above formalism. The first, the most simple, nontrivial example, is what we call a *bucket brigade.* It includes: task level information, where we specify a palette of controllers, which correspond to places in a TPN; task switching and concurrency; and a simple notion of product flow. Second, we describe a simple assembly process in terms of three robots.

5.1 The Bucket Brigade

Bucket brigades correspond to individual lines in an assembly process and we expect their analysis to contribute to our understanding of more complicated factories. Figure 8 shows the TPN we will use to model the brigade.

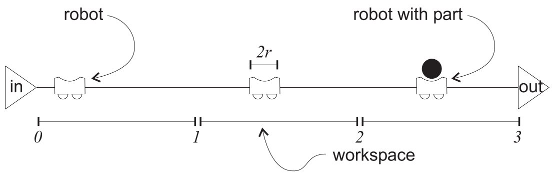

Figure 10: A simple, three robot bucket brigade

A simplistic, three robot bucket brigade consists of three robots, R_1, R_2 and R_3, a parts feeder and an output buffer arranged in a line as in Figure 10. The task is for R_1 to pick up parts, one at a time, transfer them to R_2, which transfers them to R_3. R_3 deposits the parts in an output buffer. Suppose the robots have width $2r$ where $0 < r < 1$. Define the workspace to be the closed interval of the real line $[0, 3]$ and suppose that the robots have continuous state variables, x_1, x_2 and x_3, corresponding to their positions on this line. We assume that the range of each robot is restricted, so that $x_i \in [i-1, i]$ only and also that robots can not inhabit the same place at the same time. That is, the distance between any to robots must be greater than $2r$. Each robot also has a discrete state, corresponding to whether or not it is carrying a part, denoted $b_1, b_2, b_3 \in \{0, 1\}$. Say $b_i = 1$ if R_i is carrying a part and $b_i = 0$ otherwise. Finally, there are some enormous number, n, of parts with state variables $z_1, \ldots, z_n$. Since this example is for illustrative purposes only, the physics are

Table 1: Table of transient machines used in the bucket brigade example

Name	Robots	Dimension of Config. Space	Domain (for robots)	Goal Point	Function
F_{pick}	R_1	2	$[0, 1-r]$	0	R_1 picks up part
F_{drop}	R_3	2	$[2+r, 3]$	3	R_3 drops of part
F_{wait_i}	R_i, $i \in \{1,2,3\}$	1	$[i-1+r, i-r]$	$i-\frac{1}{2}$	R_i waits in safe place
F_{hold_i}	R_i, $i \in \{1,2,3\}$	2	$[i-1+r, i-r]$	$i-\frac{1}{2}$	R_i waits in safe place with part
$F_{trans_{1,2}}$	R_1, R_2	3	$[r,1]\times[1,2-r]$	$(1-r, 1+r)$	R_1 transfers part to R_2
$F_{trans_{2,3}}$	R_2, R_3	3	$[1+r,2]\times[2,3-r]$	$(2-r, 2+r)$	R_2 transfers part to R_3

not entirely realistic: the velocity of the robots is directly controllable ($\dot{x} = u$), parts move with the robots they are near (so that when $b_i = 1$, $z_k = x_i$ for some part with index k), and part transfers happen instantaneously as long as the robots involved are close together.

The first step in constructing the brigade is to specify the palette of controllers. There are nine: F_{pick}, F_{drop}, F_{wait_i} and F_{hold_i} for $i \in \{1,2,3\}$, $F_{trans_{1,2}}$, and $F_{trans_{2,3}}$. The use of these controllers is summarized in the table in Figure 1. Note that controller goals are given by a point x^* in the table but we usually consider them to be a set, $B_\epsilon(x^*) = (x^* - \epsilon, x^* + \epsilon)$ We will describe each of them qualitatively and show, as an example, how one of them might be actually determined using a *navigation function*. The details of the rest should then be clear.

F_{pick} is a two-dimensional controller. One dimension corresponds to the position R_1 and the other to the position of the kth part, which we assume stationary in the parts feeder (so $z_k = 0$). There is a single attracting equilibrium point at 0 where R_1 is next to the parts feeder with a part. The parts feeder operates by proximity. That is, if $b_1 = 0$ and $||x_1|| < \epsilon$ then b_1 will eventually become 1. In the bucket brigade, when $b_1 = 0$, R_1 will run $\dot{x}_1 = F_{pick}(x_1)$ until $b_1 = 1$. Since the controller does not affect the part, $\dot{z}_k = 0$.

F_{drop} is similarly used by R_3 to drop off a part at the output buffer. It is used when $b_3 = 1$ and eventually results in b_3 becoming 0.

F_{wait_i}, for each i, is a one-dimensional controller with attracting equilibrium point $i - \frac{1}{2}$. It is used when R_i needs to wait for another robot, the parts feeder or the output buffer. F_{wait_i} essentially drives R_i to a safe place in the workspace and keeps it there. F_{hold_i} is the same except it is two dimensional and used when the robot is waiting *and* holding a part.

$F_{trans_{1,2}}$ is a three-dimensional controller for R_1 to hand R_2 a part, the position of which we assume to be given by z_k. It has attracting point $(x_1, x_2) = (1-r, 1+r)$ (implying that z_k is attracted to $1-r$). That is, it drives R_1, R_2 and the part to a configuration where the robots are touching. When $(b_1, b_2) = (1, 0)$ and $||x_2 - x_1|| < \epsilon$ for some ϵ, the state instantaneously becomes $(b_1, b_2) = (0, 1)$ and $z_k = x_2$. That is, for R_1 to transfer a part to R_2, the robots need only get very close to each other and the transfer happens. We note, according to our notion of controller, that $F_{trans_{1,2}}$ is being run asynchronously. Half by R_1 and half by R_2. That is, R_1 obtains an estimate of $\hat{x}_1$ of x_1 via its sensors and an estimate $\hat{x}_2$ of x_2 via the communication system and runs the feedback controller

$$\dot{x}_1 = \pi_1 \circ F_{trans_{1,2}}(\hat{x}_1, \hat{x}_2)$$

concurrently and asynchronously while R_2 runs $\pi_2 \circ F_{trans_{1,2}}$. Once again, while the controller is running, the change in the part's position is the same as that of the robot that is holding it. The description of $F_{trans_{2,3}}$ is similar.

Lest the reader be suspicious that these controllers are entirely fictitious, we review a method by which a function suitable for $F_{trans_{1,2}}$ may be constructed using a *navigation function*, [24]. A navigation function is an artificially constructed potential field over a compact domain, with a single minimum point (the goal), and which achieves a uniform maximum at the boundary of the domain. The control law is the negative gradient of the navigation function and is guaranteed to drive the system to the goal point. Since $\dot{z}_k = \dot{x}_1$ while R_1 is holding the kth part, we describe $F_{trans_{1,2}}$ as though it were a two dimensional controller, although technically, it is a three dimensional one. Recall that we wish for $F_{trans_{1,2}}$ to have a stable equilibrium at $(x_1^*, x_2^*) = (1 - r - \epsilon, 1 + r + \epsilon)$ (the ϵ is added so that the goal is not on the boundary of the domain. Thus we first construct a function where this is the case:

$$\gamma(x_1, x_2) = (x_1 - x_1^*)^2 + (x_2 - x_2^*)^2$$

will do. Next, we construct a function that goes to zero along the boundary of the domain).

Recall (Table 1) that the boundary includes $x_1 = r$ and $x_2 = 2 - r$. It also includes the configuration where x_1 and x_2 are closer than $2r$, that is, where the robots are physically touching. The function

$$\beta(x_1, x_2) = (x_1 - r)^2 (x_2 - 2 + r)^2 (||x_2 - x_1||^2 - (2r)^2)^2$$

has this property. We let

$$\hat{\phi}(x_1, x_2) = \frac{\gamma(x_1, x_2)^k}{\beta(x_1, x_2)}$$

where k is chosen large enough so that (x_1^*, x_2^*) is the only minimal point. Finally, the navigation function is

$$\phi = \frac{\hat{\phi}}{1 + \hat{\phi}}$$

We define $F_{trans_{1,2}}(x_1, x_2) = -\nabla\phi(x_1, x_2)$. This reactive control law can be shown to have a single, stable equilibrium point at (x_1^*, x_2^*) and to avoid the boundary of its domain. The vector field that results may be normalized and tuned for use. This technique was used for all the transient machines described above to build a satisfying simulation of arbitrary length bucket brigades and of a class of simple "toy" factories described in Section 5.2.

Once the transient machines are specified, we may construct rules for switching between them for each robot. The first robot repeatedly receives parts from the parts feeder and hands them to the second robot. Thus, R_1 runs the program

```
While True
  If b_1 = 0
    Wait for parts feeder
    Run ẋ_1 = F_pick(x_1) until b_1 = 1
  Else
    Wait for R_2 to be ready
    Run ẋ_1 = π_1 ∘ F_trans_{1,2}(x_1, x_2) until b_1 = 0
  EndIf
End While
```

The second robot receives parts from the first and hands them to the third. So R_2 runs the program

```
While True
  If b_2 = 0
    Wait for R_1 to be ready
    Run ẋ_2 = π_2 ∘ F_trans_{1,2}(x_1, x_2) until b_2 = 1
  Else
    Wait for R_3 to be ready
    Run ẋ_2 = π_1 ∘ F_trans_{2,3}(x_2, x_3) until b_2 = 0
  EndIf
End While
```

Finally, the last robot receives parts from the previous robot and deposits them in the output buffer. So R_3 runs the program

```
While True
  If b_1 = 1
    Wait for output buffer
    Run ẋ_3 = F_drop(x_3) until b_3 = 0
  Else
    Wait for R_2 to be ready
    Run ẋ_3 = π_2 ∘ F_trans_{2,3}(x_2, x_3) until b_3 = 1
  EndIf
End While
```

Notice that any line in a program that says "wait for ..." has the meaning "run F_{wait_i} until the robot (or feeder or buffer) is ready" and presumes some simple communication system that we do not describe in any detail here.

The dynamics of the bucket brigade we have constructed can be modeled by the tools we presented in Sections 3 and 4. Figure 8 shows the resulting TPN. There are three gears, one for each robot. Gear two corresponds to the program of R_2, for example, and consists of transitions $\{t_1, t_2, t_4, t_5\}$ and places $wait_2 = [t_5, t_1], trans_{1,2} = [t_1, t2], hold_2 = [t_2, t_4], trans_{2,3} = [t_4, t_5]$.

As an example, we supply one redistribution function in detail, namely, d_{t_1}. F_{hold_1} expects a robot position and then a part position, in that order. F_{wait_2} expects a robot position. $F_{trans_{1,2}}$

expects the position of the sending robot, the receiving robot and the part in that order. Then we have:

$$\begin{aligned} d_{t_1}(hold_1, 1) &= (trans_{1,2}, 1) \\ d_{t_1}(hold_1, 2) &= (trans_{1,2}, 3) \\ d_{t_1}(wait_2, 1) &= (trans_{1,2}, 2) \end{aligned}$$

The rest of the redistribution functions should now be apparent from Figure 8.

Let $\tilde{p} = [\emptyset, t_3]$ and $\tilde{q} = [t_7, \emptyset]$. The initial marking, (m_0, f_{m_0}) is given by $m_0 = \{wait_1, wait_2, wait_3, \tilde{p}\}$ and

$$\begin{aligned} f_{m_0}(wait_1, 1) &= 1 \\ f_{m_0}(wait_2, 1) &= 2 \\ f_{m_0}(wait_3, 1) &= 3 \\ f_{m_0}(\tilde{p}, 1) &= k \end{aligned}$$

for some k. Note that the addition of parts feeders results in our having to expand and contract the index set N. In this example, the set will always include $\{1, 2, 3\}$ and, depending on what parts are present in the brigade and the parts feeder, may also include some set $\{k_1, ..., k_j\}$ of part indices.

Now we can show two things. First, the bucket brigade never deadlocks by Theorem 3.1. This follows from the fact that its underlying structure is a gear net and because the domains of each place include the goals of the places that precede it. As an example, consider $trans_{1,2}$. Its domain is $\mathcal{D}_{trans_{1,2}} = [r, 1] \times [1, 2-r]$ according to Table 1 (we ignore the part position since it is the same as one of the robots). Now, ${}^\bullet trans_{1,2} = \{hold_1, wait_2\}$ and we have that

$$\mathcal{G}_{hold_1} \times \mathcal{G}_{wait_2} = B_\epsilon(\frac{1}{2}) \times B_\epsilon(\frac{3}{2}) \subseteq \mathcal{D}_{trans_{1,2}}.$$

Second, we can deduce from Theorem 4.1 that the parts move from one end of the brigade to the other. Suppose that at some marking (m_j, f_{m_j}) that $pick \in m_j$ and $f_{m_j}(pick, 2) = k$. That is that R_1 has picked up part k. Then we know there is a sequential sequence of places that control part k. In fact, the sequence is

$$< pick, hold_1, trans_{1,2}, hold_2, trans_{2,3}, hold_3, drop > .$$

Now if we trace the position of the robot carrying the part in the goals of these controllers from *pick* to *hold* (see Figure 1 again), we see that the initial state of the part is $z_k = 0$ and the final state is $z_k = 3$.

There are several things to notice about this example. Control is decentralized as we required in our statement of the problem. Communications is kept low: each robot need only communicate with at most one other robot at a time. And the dimension of the control laws is limited to two: we could, in principle, build bucket brigades with an arbitrary number, n, of robots using the above method, yet we would not have to build anything that is fundamentally different from what we already have. Thus, the method scales.

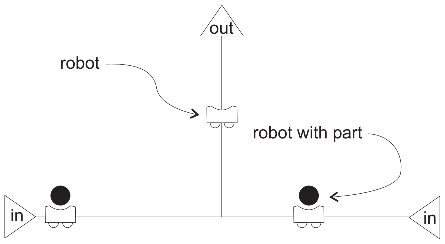

Figure 11: Factory setup for a simple mate operation.

5.2 Assembly

We may also use gear nets and TPNs to model a simple assembly process. Three robots perform this task on a T-shaped guidepath. The first two pick up parts at their respective parts feeders and then, in a synchronized operation with the third robot, attach the two parts together and place the resulting assembly on the third robot. The third robot drops the assembly off in a parts bin. Figure 11 shows how the robots are arranged.

We will not describe this example in as much detail. There are $wait_i$ and $hold_i$ operations as before, two *pick* operations corresponding to the two different parts, one *drop* operation. New to this example is the *mate* operation. It controls two robots, each carrying a part, and a third robot to meet at the intersection of the guidepaths. Once there, the assembly of the two parts occurs, and the result is place on the third robot. The TPN which models this is shown if Figure 12. Note that the two parts lines originate at the parts feeders, meet at the *mate* operation and end up at the output buffer. We could also have identified the two parts lines after the *mate* operation and drawn only one output line for the assembly.

Once again, we can show that the factory does not deadlock using the properties of gearnets. We can also show that the two types of parts progress through the factory using the properties of TPNs.

6 Compiling from the Product Assembly Graph

In the previous section, we demonstrated that the tools we have developed, gear nets and hybrid petri nets, can be used to analyze a simple factory-like situation. In this section, we illustrate how we can expand on those ideas to *automatically* generate gear net based hybrid nets from simplified product assembly graphs.

We represent a class of PAGs recursively as follows. We start with some number of operation types, $OP_1, ..., OP_n$, where OP_i takes k_i subassemblies and returns a single assembly that results from performing some corresponding operation on the subassemblies. The class of PAGs are then defined by:

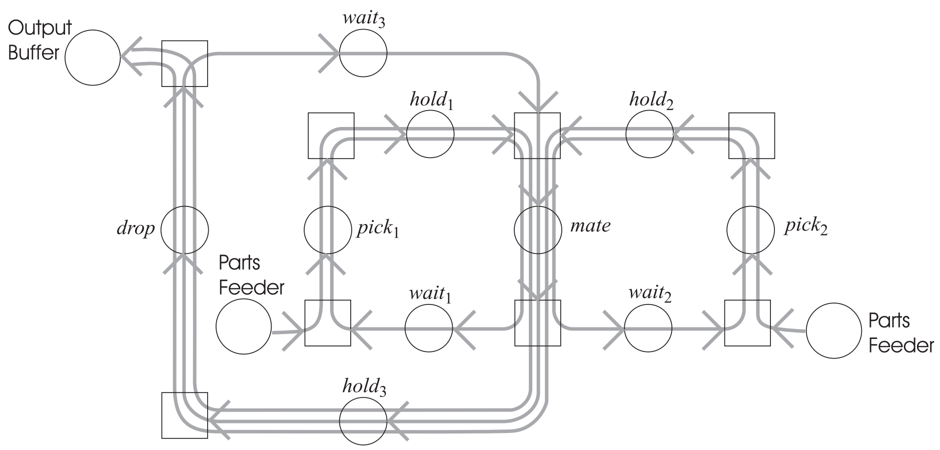

Figure 12: A simple, three robot bucket brigade

1. The object *atomic*() is a PAG representing a simple part.
2. If OP_i is an operation and $sub_1, ..., sub_{k_i}$ are subassemblies, then $OP(sub_1, ..., sub_{k_i})$ is a subassembly

For example, we have developed a "toy" factory which has these operations:

1. $MATE(sub_1, sub_2)$
2. $WELD(sub_1)$

Note that each PAG can be represented as a tree with the nodes representing operations and their children, the subassemblies in the operation.

To each operation, we assume that a template controller, a so called *transient machine*, is already built. We also assume there are transient machines for picking up parts at parts feeders and dropping them off at output buffers. The four templates for PICK, DROP, MATE and WELD are shown in Figure 13. They are represented by Petri Net fragments. The lines going in represent the constituent robots and parts feeders and the lines going out represent the robots and output buffers.

To compile, we annotate the PAG with these Petri Net fragments and then connect them so as to create a gear net. We know that the resulting net is live because it is a gear net. This process is shown in with a simple example in Figure 14. In future work we intend to describe the compilation procedure formally and prove that it produces gear net based TPNs. The layout must also be obtained from thge resulting net. For our "toy" factory, the layout is essentially given by an embedding of the PAG into the plane. We have not investigated more complicated layout procedures.

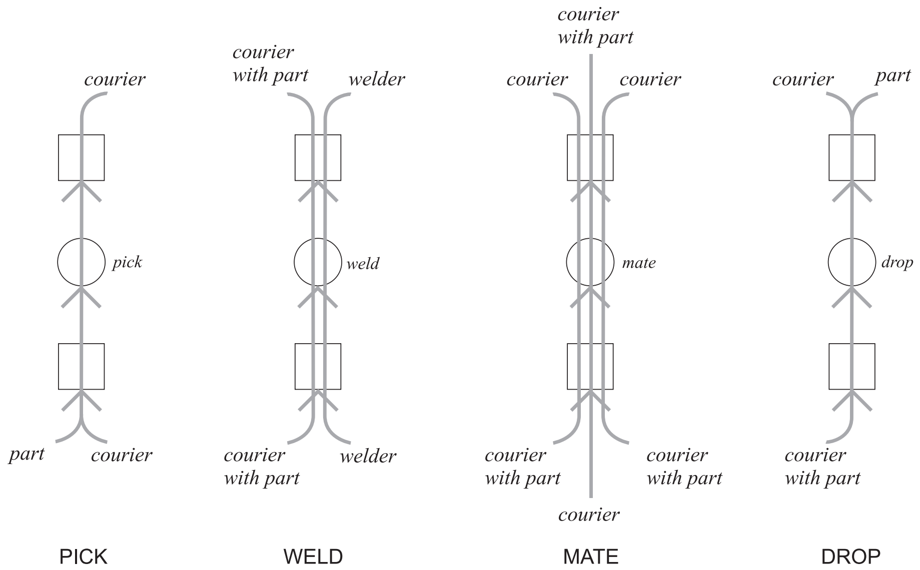

Figure 13: Templates for various "toy" factory operations.

7 Conclusions

Our research aims at developing a method for composing concurrent dynamic systems. Our initial attempt to do so has led us to invent a class of Petri Nets which can contain the notion of a control mode at each place in the net. We have dealt with composition using gear nets, which can be used to describe certain concurrent processes which must occasionally be locally synchronized. We have shown that for simple situations, where a palette of controlled dynamic systems with stable equilibrium points is supplied, that we have partially accomplished such a composition.

We plan to extend this work in several directions. First, we intend to investigate the compilation technique discussed in Section 6. We may add optimizations to the compiler of various types. For example, we may assign the two or more programs in the gear net to one robot which must, therefore, switch between the two programs. The resulting program for that robot is not a gear, but it is a well defined object that we believe has properties which we may use to show the resulting net is live.

Second, we believe it is important to extend this work to apply in situtations where the dynamics across transitions must account for the momentum of the system and for smoothness properties. In the present work, and in fact in much of the work in hybrid systems in general, discrete transitions between dynamic modes are instantaneous. For many systems, this is a convenient way to abstract away from impulse dynamics and the like. However, in control of physical systems which involve robots or manipulation of unactuated objects, such as in some robot juggling or locomotion tasks,

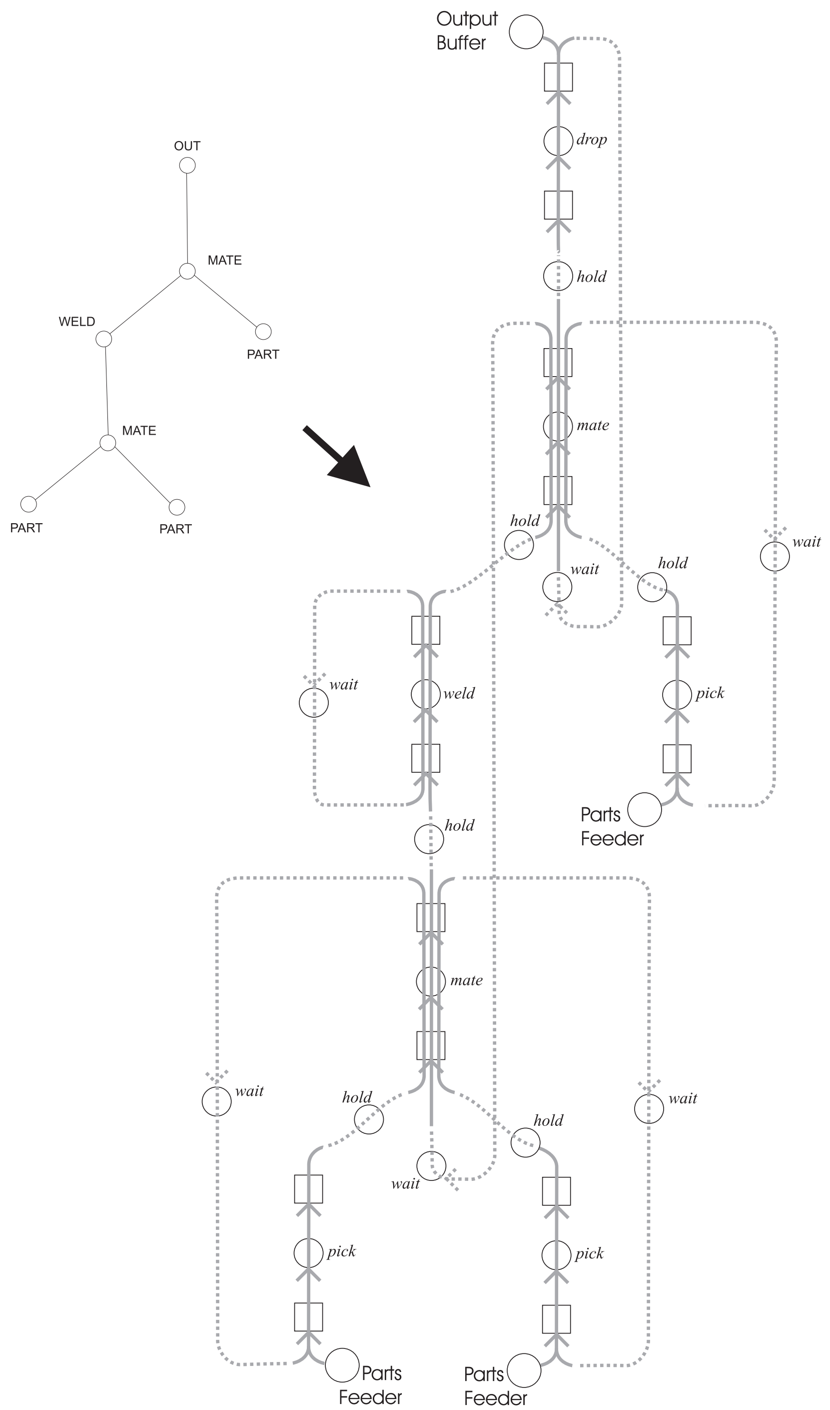

Figure 14: A simple example where a PAG is annoated with gears.

control must be smooth across transitions in order to be realistically implemented with present day actuator technology. It is particularly important in distributed robot systems that we develop a means to acheive this smoothness while still keeping the dimension of the local couplings of robots low.

Appendix I: Dynamic Systems

In this section we very briefly define the terms from dynamic systems that we use in this paper.

We represent the state of a dynamic system at a time $t \in \mathbb{R}$ by an n-dimensional point $x(t)$ in some n-dimensional space such as $\mathbb{R}^n$ or, more generally, an n-dimensional manifold, M. We frequently omit the t and just write x. We denote by $\dot{x}$ the derivative of x with respect to t. That is,

$$\dot{x} = (\frac{dx_1}{dt}, ..., \frac{dx_n}{dt}).$$

Similarly, $\ddot{x}$ is the second derivative of x with respect to t. We can specify a dynamic system with a function, called a vector field, from the space x lives in, M, to the tangent space of M. For example, $F : \mathbb{R}^n \to \mathbb{R}^n$. From F we may obtain a system $\dot{x} = F(x)$.

The solution of the equation $\dot{x} = F(x)$, which may not exist in a closed form, is the function $f : \mathbb{R} \times M \to M$ such that

$$F(x) = \frac{df}{dt}(t, x)|_{t=0}.$$

Starting with a point x_0, the **forward orbit** of x_0 is the set $\{f(t, x_0) \mid t \in \overline{\mathbb{R}^+}\}$.

A **fixed point** of a system is a point x^* such that $F(x^*) = 0$ (or, equivalently, $f(t, x^*) = x^*$ for all t). x^* is a stable fixed point if there is a neighborhood U of M around x^* such that for every $y \in U$, the forward orbit of y contains x^*. The largest U with this property is called the **domain of attraction** of x^* (or the controllable set) and is written $\mathcal{D}$. Often we consider a subset of $\mathcal{D}$ that contains x^* called the goal and written $\mathcal{G}$.

Appendix II: Petri Nets and Partial Orders

In this appendix we give the basic definitions of Petri Nets and partial orders as they pertian to the semantics of Petri Nets. We use the notation of [11] to represent Petri Nets as this is the most convenient and concise we have found for our purposes. We shall show the relationship of this with the more standard definition as well. Note that our definition of Petri Net is actually what is commonly referred to as a *condition/event net*, see [23].

Definition 7.1 *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**. *If $\{\{a_1, ..., a_i\}, \{b_1, ..., b_j\}\} \in P$ we write $[a_1, ..., a_i; b_1, ..., b_j] \in P$.*

Note that places are also called *conditions* and transitions are also called *events*. If $p = [a_1, ..., a_i; b_1, ..., b_j]$ is a place then $left(p) = \{a_1, ..., a_i\}$ and $right(p) = \{b_1, ..., b_j\}$. We usually assume that whenever $t \in T$ there are places $p, q \in P$ such that $t \in right(p)$ and $t \in left(q)$ so that all transitions have actual post conditions. We say that a net (S, Q) is contained in (T, P) if $Q \subseteq P$ (which implies that $S \subseteq T$).

The usual definition of a Petri Net is a triple $(T, P; F)$ where $F \subseteq T \times P \cup P \times T$ is called the flow relation. We may recover the flow relation from (T, P) in our definitions as follows.

Definition 7.2 *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 next definition (and the notation, believe it or not) are straight from the petri net literature.

Definition 7.3 *Given a Petri Net (T, 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 $x^\bullet$.*

Next we describe the dynamics of a Petri Net.

Definition 7.4 *A* **marking** *of a Petri Net (T, P) is a subset m of P.*

Usually when we introduce a Petri Net, we will give a condition on markings by defining what a **legal marking** for the net is. The intention is that the only possible states of the Petri Net are those given by legal markings. Thus, legal markings should be closed under the dynamics of the net.

Definition 7.5 *Say m is a marking of a net (T, P). A transitions $t \in T$ is m-***enabled** *if ${}^\bullet t \subseteq m$ and $t^\bullet \cap m = \emptyset$. If t is m-enabled, we define m' to be the* **follower** *of m under t is*

$$m' = (m - {}^\bullet t) \cup t^\bullet.$$

We write $m \rightarrow^t m'$ in the case that t is m-enabled and m' is the follower of m under t.

Definition 7.6 *Two transitions t_1 and t_2 are* **detached** *if ${}^\bullet t_1 \cap {}^\bullet t_2 = {t_1}^\bullet \cap {t_2}^\bullet = \emptyset$. A set $G \subseteq T$ is detached if its elements are pairwise detached.*

For $G \subseteq T$, we denote by ${}^\bullet G$ and $G^\bullet$ the union of the presets and postsets of events in G respectively. Given a detached set G, we write $m \rightarrow^G m'$ in the case that every $t \in G$ is m-enabled and

$$m' = (m - {}^\bullet G) \cup G^\bullet.$$

Obviously, we can create sequences of markings $\langle m_0, m_1, m_2, ...\rangle$ when there exist detached sets $G_0, G_1, ...$ such that

$$m_0 \rightarrow^{G_0} m_1 \rightarrow^{G_1} m_2 \rightarrow^{G_2} ...$$

thereby obtaining a possible run of the net.

A particular kind of Petri Net we will use in this paper is the **marked graph** which is a Petri Net (T, P) such that $|{}^{\bullet}p| \leq 1$ and $|p^{\bullet}| \leq 1$ for every $p \in P$. In a sense, marked graphs are Petri Nets without nondeterminism, since every condition leads, in one and only one, way to the set of transitions in its posetset. The literature on marked graphs, for example [7], usually calls a marking for a marked graph a function $m : P \rightarrow \mathbb{N}$ (where $\mathbb{N}$ is the natural numbers including zero). We will not have use for integers markings where places are marked other the with 0 or 1 so we use will stay with the notion that a marking is a subset of P.

Finally, we describe the partial order semantics of a net.

Definition 7.7 *An* **occurence net** (U, K) *is a Petri net that is acyclic and where* $|{}^{\bullet}p| \leq 1$ *and* $|p^{\bullet}| \leq 1$.

When a net is acyclic, we can consider it as a certain kind of ordering on places and transitions called a **partial order**, which is a a set, X along with a relation $\sqsubseteq$ where $\sqsubseteq$ is reflexive ($x \sqsubseteq x$ for all $x \in X$), transitive ($x \sqsubseteq y$ and $y \sqsubseteq z$ imply $x \sqsubseteq z$), and antisymmetric ($x \sqsubseteq y$ and $y \sqsubseteq x$ imply $x = y$). If we start with an occurence net (U, K) and set $\sqsubseteq = F^*$, the transitive closure of F, then $(U, K, \sqsubseteq)$ is a partial order. We write $x||y$ to mean that x and y are **uncomparable**, whenever x and y are elements of a partial order such that $x \not\sqsubseteq y$ and $y \not\sqsubseteq x$. If $S \subseteq U \cup K$ is such that for all $x, y \in S$ either $x \sqsubseteq y$ or $y \sqsubseteq x$ then S is called a totally ordered set. If S maximal (i.e. no other element z can be added to S and still preserve total ordering), then S is called a **line** or a **chain**. If $S \subseteq U \cup K$ is such that for all $x, y \in S$, $x||y$, then S is called an *antichain*. If S is maximal with this property then S is called a **cut**. A cut consisting of all places is called a **slice**. Partial orders for Petri Nets are covered briefly in [23] and more thoroughly in [2]. Partial orders are covered in most introductions to discrete math and thoroughly in [8].

Finally, we have the notion of a processes.

Definition 7.8 *A* **process** *for a Petri Net* (T, P) *is a triple* (U, K, σ) *where* (U, K) *is an occurence net and* $\sigma : U \cup K \rightarrow T \cup P$ *such that*

1. *If* D *is a slice of* $(U, K, \sqsubseteq)$, *then* $\sigma|D$ *is injective and* $\sigma(D)$ *is a legal marking and*

2. $\sigma({}^{\bullet}x) = {}^{\bullet}\sigma(x)$ *and* $\sigma(x^{\bullet}) = \sigma(x)^{\bullet}$.

References

[1] R. Alur, C. Courcoubetis, T. Henzinger, and P.H. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems I*, LNCS 736, pages 209–229. Springer-Verlag, 1993.

[2] Eike Best. *Nonsequential Processes: A Petri Net view.* Springer-Verlag, 1988.

[3] D. Biegelsen, W. Jackson, A. Berlin, and P. Cheung. Air jet arrays for precision positional control. In *International conference on micromechanics for information and precision equipment*, Tokyo, 1997.

[4] M. Bühler, D.E. Koditschek, and P.J. Kindlmann. A simple juggling robot: Theory and experimentation. In V. Hayward and O. Khatib, editors, *Experimental Robotics I*, pages 35–73. Springer-Verlag, 1990.

[5] Robert R. Burridge. *Sequential Composition of Dynamically Dexterous Robot Behaviors.* PhD thesis, University of Michigan, 1996.

[6] Robert R. Burridge, Alfred A. Rizzi, and Daniel E. Koditschek. Sequential composition of dynamically dexterous robot behaviors. To appear, 1998.

[7] F. Commoner, A.W. Holt, S. Even, and A. Puneli. Marked directed graphs. *Journal of Computer and System Sciences*, 5:511–523, 1971.

[8] B. A. Davey and H. A. Priestly. *Introduction to Lattices and Order.* Cambridge University Press, 1990.

[9] J. Gowdy and A. A. Rizzi. Programming in the architecture for agile assembly. In *International Conference on Robotics and Automation*, pages 3103–3108, Detroit, MI, 1999. IEEE.

[10] 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.

[11] Ryszard Janicki. Nets, sequential compositions and concurrency relations. *Theoretical Computer Science*, 29:87–121, 1984.

[12] Stephen G. Kaufman et al. The Archimedes 2 mechanical assembly planning system. In *Proceedings of the 1996 IEEE Conference on Roboitcs and Automation*, pages 3361–3368, 1996.

[13] L. Kavraki. Part orientation with programmable vector fields: Two stable equilbria for most parts. In *Proceedings of the IEEE Conference on Robotics and Automation*, Albuquergue, New Mexico, April 1997.

[14] O. Khatib et al. Force strategies for cooperative tasks in multiple mobile manipulation systems. *International Journal of Robotics Research*, 14(1):19–36, feb 1995.

[15] Ekkart Kindler. A compositional partial order semantics for Petri Nets. In P. Azena and G. Balbo, editors, *Applications and Theory of Petri Nets 1997 : Proceedings, LNCS 1248*. Springer-Verlag, 1997.

[16] Daniel E. Koditschek. An approach to autonomous robot assembly. *Robotica*, 12:137–155, 1994.

[17] 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.

[18] Tsai-Yen Li and Jean-Claude Latombe. On-line manipulation planning for two robot arms in a dynamic environment. *International Journal of Robotics Research*, 16(2):144–167, 1997.

[19] 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.

[20] Patrick F. Muir, Alfred A. Rizzi, and Jay Gowdy. Minifactory: A precision assembly system adaptable to the product life cycle. *Architectures, Networks, and Intelligent Systems for Manufacturing Integration (Proceedings of the SPIE)*, pages 74–80, 1997.

[21] Euisu Park, Dawn M. Tilbury, and Pramod P. Khargonakar. A formal implementation of logic controllers for machining systems using petri nets and sequential function charts. *I Don't Know*, I Don't Know.

[22] Arthur E. Quaid and Ralph L. Hollis. Cooperative 2-dof robots for precision assembly. In *International Conference on Robotics and Automation*, Minneapolis, MN, 1996. IEEE.

[23] Wolfgang Reisig. *Petri Nets: An Introduction.* Springer Verlag, 1985.

[24] 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.

[25] Alfred A. Rizzi. *Dexterous Robot Manipulation.* PhD thesis, University of Michigan, 1994.

[26] Alfred A. Rizzi, Louis Whitcomb, and D.E. Koditschek. Distributed real-time control of a spatial robot juggler. *IEEE Computer*, 25(5), 1992.

[27] Bruce Romney. *On the Concurrent Design of Assembly Sequences and Fixtures.* PhD thesis, Stanford University, March 1997.

[28] Bruce Romney, Cyprien Godard, Michael Goldwasser, and G. Ramkumar. An efficient system for geometric assembly seqence generation and evaluation. In *Proceedings of the 1995 AMSE. Intl. Computers in Engineering Conf.*, pages 699–712, 1995.

[29] J. W. Suh et al. CMOS integrated organic ciliary array as a general-purpose micromanipulation tool for small objects. *Journal of Micromechanical Systems*, 1998. Submitted for review.

[30] Brian C. Williams. Model based programming of reactive systems. At an Invited Talk for the AAAI Workshup: *Hybrid Systems and AI: Modeling Analysis and Control of Discrete + Continuous Systems* at Stanford University, 1999.

[31] Randall H. Wilson. Geometric reasoning about assembly tools. *Artificial Intelligence*, 98:237–279, January 1998.