Constraint Nets: A Semantic Model for Real-Time Embedded Systems

by
Ying Zhang
and
Alan K. Mackworth

Technical Report 92-10
May 1992

Department of Computer Science
University of British Columbia
Rm 333 - 6356 Agricultural Road
Vancouver, B.C.
CANADA V6T 1Z2

Constraint Nets: A Semantic Model for Real-Time Embedded Systems

by

Ying Zhang and Alan K. Mackworth

Technical Report 92-10
May 1992

Department of Computer Science
The University of British Columbia
Vancouver, B. C. V6T 1Z2
Canada

email: zhang@cs.ubc.ca, mack@cs.ubc.ca

Constraint Nets: A Semantic Model for Real-Time Embedded Systems

Ying Zhang and Alan K. Mackworth

Department of Computer Science
University of British Columbia
Vancouver, B.C., Canada
E-mail: zhang@cs.ubc.ca, mack@cs.ubc.ca
phone: (604)822-3731

Abstract

We take a real-time embedded system to be the control system of a plant in an open environment, where the control is realized by computation in digital or analog form. The key characteristic of a real-time embedded system is that computation is interleaved or in parallel with actions of the plant and events in the environment. Various models for real-time embedded systems have been proposed in recent years, most of which are extensions of existing concurrency models with delays or time bounds on transitions. In this paper, we present a different approach to modeling real-time systems. We take the overall system as a dynamic system, in which time or event structures are considered as an intrinsic dimension. Our model, called the Constraint Net model (CN), is capable of expressing dynamic behaviors in real-time embedded systems. It captures the most general structure of dynamic systems so that systems with discrete as well as dense time and asynchronous as well as synchronous event structures can be modeled in a unified framework. It models the dynamics of the environment as well as the dynamics of the plant and the dynamics of the computation and control. It provides multiple levels of abstraction so that a system can be modeled and developed hierarchically. By explicitly representing locality, CN can be used to explore true concurrency in distributed systems. With its rigorous formalization, CN provides a programming semantics for the design of real-time embedded systems. It also serves as a foundation for specification, verification, analysis and simulation of the complete dynamic system.

1 Motivation and Introduction

Real-time embedded systems are reactive as well as purposive systems, closely coupled with their plants and environments; they must deal with inconsistent, incomplete and delayed information from various sources. Such systems are usually complex, hierarchical and physically distributed. Design of control systems for plants with high degree of freedoms and multiple sensors has become a growing challenge. Work on real-time embedded systems is usually based on one of two frameworks [5]: semantic models or scheduling theory. Even though these two are closely related, in this paper we shall concentrate on formal semantic models, as well as specification and verification of the overall system.

Much work has been done on introducing real-time concepts into formal models in recent years. The real-time representation follows one of two approaches. The first is to define events or transitions on time, i.e. each event or transition is associated with a non-negative real number, and delays or durations are also represented by non-negative real numbers. For example, Merritt et. al. [8] augmented the input-output automaton model with a notion of time that allows reasoning about timed behaviors. Various extensions of Timed CCS (Calculus of Communicating Systems) have been developed to model the relative speed of processes [9] and asynchronous behaviors. Timed CSP (Communicating Sequential Processes) was based on failure/stability models [11]. The Timed Petri Net model [2] was introduced to specify and verify real-time systems. In all of these models, either time or delay is augmented with transitions, events or processes. The second approach is to define time on events, i.e. defining time instants on a pair of adjacent events [5], such that the structure of time instants is isomorphic to the set of non-negative real numbers.

Our philosophy for developing the Constraint Net model (CN) is: instead of adding a model of time onto an existing model of concurrency, a model of dynamic systems should be developed in the first place [10]. The traditional model of dense (resp. discrete) time dynamic systems is a set of differential (resp. difference) algebraic equations. However, most advanced control systems nowadays are developed as distributed and asynchronous processes in digital computer networks, in addition to analog circuits. There is a strong need to generalize the model of dynamic systems so that multiple time structures and multiple data types can be represented in a unified framework and the dynamic interactions between various components can be analyzed. The major influences on the Constraint Net model are the Operator Net model and the Temporal Automaton model.

The Operator Net model [1], abstracted from Lucid [13] or the dataflow model, is defined on continuous algebras using fixpoint theory. The most attractive feature of this model is its independence of any particular algebra. Given a continuous algebra which specifies data types and basic operations, a sequence (continuous) algebra is obtained on which an operator net can be defined. This idea is incorporated into the development of the Constraint Net model (CN). LUSTRE [3], a development based on Lucid, is a real-time programming language, in which sequences are interpreted as time steps. In addition, LUSTRE introduces synchronous clocks, so that any expression is evaluated at its clock's sampling rate.

The Temporal Automaton model [7] is a step towards modeling causal functions in multiple time domains. The Temporal Automaton model provides explicit representation of process time, symmetric representation of a machine and its environment, aggregation of individual machines to form a machine at a coarser level of granularity. However, there remain untackled problems in modeling continuous change and event control.

As in the Operator Net model, CN is defined on continuous algebras using fixpoint theory. Like LUSTRE, CN introduces reference time structures and clocks, but the reference time may be dense and the clocks are asynchronous [12]. As in the Temporal Automaton model, transductions are introduced as an abstraction of causal functions, but the definition of transductions is generalized to include continuous and event-driven transitions. In our definition, a transduction is a state-determined transformational process from a tuple of inputs to an output. Furthermore, environments and machines are represented in a unified framework. Like both operator nets and temporal automata, constraint nets provide composite structure and multiple levels of abstraction. Unlike both operator nets and temporal automata, constraint nets introduce "locations" so that distributed memories are explicitly modeled. In contrast to most concurrency models which are inherently non-deterministic, CN is a deterministic model, while non-determinism can be modeled by open constraint nets. In summary, the major contributions of the Constraint Net model are: (1) by introducing asynchronous clocks, CN models various time structures and coordination among components with different time structures; (2) developed on abstract algebras, CN supports abstract data types and functions; (3) with a rigorous formalization, CN provides a programming semantics for the design of real-time embedded systems; (4) by modeling plants and environments as well as control, CN serves as a foundation for specification,

verification, analysis and simulation of the complete dynamic system.

This paper is organized as follows. Section 2 presents the syntax of constraint nets and a running example that is used throughout the paper. Section 3 reviews the preliminaries on Σ-algebras and Σ-domains. Section 4 introduces the algebraic formalization of dynamics. Section 5 gives the semantics using fixpoint theory of continuous algebras. Section 6 discusses behavioral specification and verification of constraint nets. Section 7 concludes this paper and proposes future research directions.

2 The Structure of Constraint Nets

In this section, we present the syntax of constraint nets and characterize the composite structure and modularity of the model.

2.1 Syntax

A *constraint net* is a triple $CN \equiv \langle Lc, Td, Cn \rangle$, where Lc is a set of *locations*, each of which is associated with a *sort*; Td is a set of *transductions*, each of which is associated with a tuple of *input ports* and an *output port*, of certain sorts; Cn is a set of directed *connections* between locations and ports of transductions of the same sort. Clocks are a special kind of location and events can be generated by transductions. Topologically, a constraint net is a bipartite graph where locations are represented by circles, transductions are represented by boxes and connections are represented by arcs, each from a port of a transduction to a location or vice versa, with the restriction that (1) there is at most one connection pointing to each location, (2) each port of a transduction connects to a unique location and (3) no location is isolated. A location is an *input* iff there is no connection pointing to it and it is an *output* otherwise. For a constraint net CN, the set of input locations is denoted by $I(CN)$, the set of output locations is denoted by $O(CN)$. A constraint net is *closed* iff there are no input locations and it is *open* otherwise.

Example 2.1 *An asynchronous event controller [12] is designed to coordinate asynchronous events in a distributed system. Consider a simple 1-buffered producer-consumer (synchronized communication) problem. On request, the producer will produce a product when the old product is consumed and the consumer will consume a product when there is a new product. In Figure 1, we use clock $C1$ to control the producer and clock $C2$ to control the consumer. $R1$ is an input event location for requesting a new*

product. R2 is an input event location for requesting the use of the product. Two negated Muller-C elements (NCs) are used to synchronize events. It is an open system since R1 and R2 are input locations. C1 generates a new event iff there is a request event at R1 and the buffer is empty. C2 generates a new event iff there is a request event at R2 and the buffer is full. There is only one buffer in this example; however, this structure can be extended to any number of buffers.

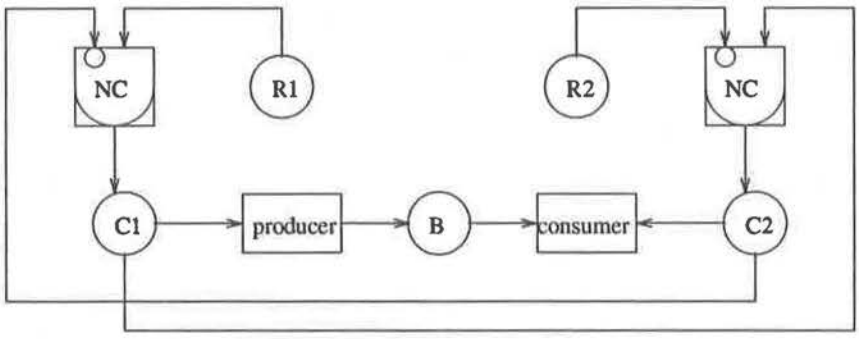

Figure 1: The constraint net for a event controller (where ∘ denotes the second input port of NC)

2.2 Modules

A *module* is a pair $\langle CN, O\rangle$ where CN is a constraint net and $O \subseteq O(CN)$ is a subset of the output locations of CN; O and $I(CN)$ define the *interface* of the module. Complex modules can be hierarchically constructed from simple ones.

There are several operations that can be applied to modules to obtain a new module. The first is *composition*, which combines two modules into one whose interface is the union of the two interfaces. The second is *coalescence*, which coalesces a group of locations in the interface of a module, with the restriction that at most one of the locations is an output location, into an individual location. The third is *hiding*, which deletes a set of output locations from the interface.

- *composition*: Let $CN_1 \equiv \langle Lc_1, Td_1, Cn_1\rangle$ and $CN_2 \equiv \langle Lc_2, Td_2, Cn_2\rangle$ where $Lc_1 \cap Lc_2 = \emptyset$ and $Td_1 \cap Td_2 = \emptyset$, then $\langle CN_1, O_1\rangle \parallel \langle CN_2, O_2\rangle =_{def} \langle CN, O\rangle$ where $CN \equiv \langle Lc_1 \cup Lc_2, Td_1 \cup Td_2, Cn_1 \cup Cn_2\rangle$ and $O \equiv O_1 \cup O_2$.

- *coalescence*: Let E be an equivalence relation on the set of locations of the interface of a constraint net such that (1) all the locations in the same equivalence class have the same sort and (2) there is at most one output location in each equivalence class; then, $\langle CN, O\rangle/E =_{def} \langle CN', O\rangle$ where $CN' \equiv \langle Lc/\overline{E}, Td, Cn/\overline{E}\rangle$, $Lc/\overline{E}$ and $Cn/\overline{E}$ denote the corresponding quotients and $\overline{E}$ is the extension of E, $\overline{E} = E \cup \{\langle l, l\rangle | l \in Lc \backslash (I(CN) \cup O)\}$.

- *hiding*: $\langle CN, O\rangle \backslash O' =_{def} \langle CN, O \backslash O'\rangle$.

3 Preliminaries: Σ-Algebras and Σ-Domains

We briefly recall in this section some mathematical preliminaries about partially ordered sets, continuous functions and fixpoint theory, essentially summarized from [4] with some extensions.

- Σ-*algebra*: $\Sigma \equiv \langle S, F\rangle$ is a signature where S is a set of *sorts* and F is a set of *function symbols* such that F is equipped with a mapping *type*: $F \rightarrow S^* \times S$. A Σ-*algebra* A is a pair $\langle \{A_s\}_{s\in S}, \{f^A\}_{f\in F}\rangle$ with a nonempty carrier set A_s for each $s \in S$ and a total function $f^A : A_{s_1} \times \ldots \times A_{s_n} \rightarrow A_s$ for each $f : s_1, \ldots, s_n \rightarrow s \in F$.

- *partial order*: $\langle A, \leq_A\rangle$ is a partial order. The product of two partial orders is a partial order defined as: $\langle a_1, a_2\rangle \leq_A \langle a_1', a_2'\rangle$ iff $a_1 \leq_{A_1} a_1'$ and $a_2 \leq_{A_2} a_2'$. The set of functions, whose range is a partial order, is a partial order defined as: for two functions f_1, f_2 with the same domain X and range A, $f_1 \leq_{X,A} f_2$ iff $\forall x \in X, f_1(x) \leq_A f_2(x)$. The least element in A, if it exists, is denoted by $\bot_A$. A *flat* partial order, written $\overline{A}$, is a set A augmented with a least element $\bot$, such that $a \leq_{\overline{A}} a'$ implies $a = a'$ or $a = \bot$.

- *complete partial order*: Let D be a subset of A and $\bigvee_A D$ be the least upper bound (*lub*) of D in A, when it exists. D is a *directed* subset of A iff it is nonempty and for every pair of elements d_1, d_2 in D the set $\{d_1, d_2\}$ has an upper bound which is also in D. The partial order $\langle A, \leq_A\rangle$ is a *complete partial order (cpo)* iff: (1) it contains a least element $\bot_A$ and (2) every directed subset of A has a *lub* in A.

- *continuous function*: Let $\langle A, \leq_A\rangle$, $\langle A', \leq_{A'}\rangle$ be two *cpos* and $f : A \rightarrow A'$ be a function. Then f is *continuous* iff for every directed set $D \subseteq A$, (1) $f(D) \equiv \{f(d) | d \in D\}$ is directed and (2)

$f(\bigvee_A D) = \bigvee_{A'} f(D)$. It is easy to see that continuous functions are closed under composition. An element $a \in A$ is a *fixpoint* of f iff $a = f(a)$. The *least fixpoint* of f is denoted by $\mu.f$.

Proposition 3.1 [4] *If $f : A \to A$ is a continuous function on a cpo A, f has a least fixpoint.*

By extending this proposition, with the same proof structure, we have:

Proposition 3.2 *If $f : I \times X \to X$ is a continuous function, then there exists a unique continuous function $\mu.f : I \to X$, such that for all $i \in I$, $(\mu.f)(i)$ is the least fixpoint of $\lambda x.f(i,x)$.*

- *compactness*: Let A be a *cpo*. An element $a \in A$ is *compact* iff given D, a directed subset of A, whenever $a \leq_A \bigvee_A D$, there exists some $d \in D$ such that $a \leq_A d$. A is an *algebraic cpo* iff for every a in A, $a = \bigvee_A \{d | d \leq_A a, d \text{ is compact}\}$. It is easy to show that (1) any flat partial order is an algebraic *cpo*, (2) the product of two algebraic *cpo*s is an algebraic *cpo*, and (3) the set of all functions whose range is an algebraic *cpo* is an algebraic *cpo*.

- Σ-*domain*: Let Σ be a signature which contains a distinguished nullary function symbol Ω_s for each sort s. A Σ-*domain* is a triple $\langle \{A_s\}_{s \in S}, \{\leq_{A_s}\}_{s \in S}, \{f^A\}_{f \in F} \rangle$ where $\langle \{A_s\}_{s \in S}, \{\leq_{A_s}\}_{s \in S} \rangle$ is an S-sorted algebraic *cpo*, f^A is continuous for each $f \in F$ and Ω_s^A is $\bot_{A_s}$. A Σ-*domain homomorphism* $h : A \to B$, where A and B are two Σ-domains, is a family of continuous mappings $\{h_s : A_s \to B_s\}_{s \in S}$ such that for each $f : s_1, \ldots, s_n \to s \in F$ and each $a_1 \in A_{s_1}, \ldots, a_n \in A_{s_n}$, $h_s(f^A(a_1, \ldots, a_n)) = f^B(h_{s_1}(a_1), \ldots, h_{s_n}(a_n))$. It is a Σ-*domain isomorphism* if it has an inverse.

- *completion*: If $\langle A, \leq_A \rangle$ be a partial order, a mapping $in : A \to 2^A$ is defined as $in(a) = \{x | x \leq_A a\}$. The *completion* of A is $A^\infty =_{def} \langle \{in(a) | a \in A\}, \subseteq \rangle$. Completions of sets have good properties.

Proposition 3.3 *If $\langle A, \leq_A \rangle$ has a least element, then A^∞ is an algebraic cpo and every element of A^∞ is compact.*

4 An Algebraic Theory of Dynamics

In this section, we start to define the semantics of locations and transductions of constraint nets, first by introducing time or event structures, and then by developing an algebraic theory of dynamics.

4.1 Time Structures and Dynamic Systems

Generalizing [7], we define a *time structure* as a pair $\langle \mathcal{T}, m \rangle$ where $\langle \mathcal{T}, \leq_{\mathcal{T}} \rangle$ with $\perp_{\mathcal{T}}$ as the least element is a total order, and $m : \mathcal{T}^\infty \rightarrow \mathcal{R}^+$ is a measurement function, where $\mathcal{T}^\infty$ is the completion of $\mathcal{T}$ and $\mathcal{R}^+$ is the set of non-negative real numbers with the normal ordering, which satisfies (1) $m(\perp_{\mathcal{T}^\infty}) = 0$ and (2) if $t_1 \subset t_2$, then $m(t_1) < m(t_2)$. For a time structure $\langle \mathcal{T}, m \rangle$, $\mathcal{T}^\infty$ is called the *time domain.* We use the time domain $\mathcal{T}^\infty$ instead of $\mathcal{T}$ because it has better mathematical properties: for all $t \in \mathcal{T}^\infty$, the unique predecessor $pre(t)$ can be defined:

$$pre(t) =_{def} \begin{cases} \perp_{\mathcal{T}^\infty} & \text{if } t = \perp_{\mathcal{T}^\infty} \\ \bigvee\{t' | t' \in \mathcal{T}^\infty, t' \subset t\} & \text{otherwise} \end{cases}$$

It is easy to show that for all $t \in \mathcal{T}^\infty$, $pre(t) \in \mathcal{T}^\infty$ and if $t \neq \perp_{\mathcal{T}^\infty}$, then $pre(t) \subset t$. A time structure $\langle \mathcal{T}, m \rangle$ may be related to another time structure $\langle \mathcal{T}_r, m_r \rangle$ by a reference time mapping h where (1) $h : \mathcal{T}^\infty \rightarrow \mathcal{T}_r^\infty$ is a continuous function and strictly monotonic, i.e. $t \subset t'$ implies $h(t) \subset h(t')$ and (2) the least element is preserved, i.e. $h(\perp_{\mathcal{T}^\infty}) = \perp_{\mathcal{T}_r^\infty}$. $\mathcal{T}_r$ is a *reference time* of $\mathcal{T}$, and $\mathcal{T}$ is a *sampled time* of $\mathcal{T}_r$. $\mathcal{T}$ is *accurate* w.r.t. $\mathcal{T}_r$ iff $\forall t \in \mathcal{T}^\infty, m(t) = m_r(h(t))$.

A *variable trace* is a mapping from a time domain to a variable domain which is an algebraic *cpo*, $v_T^A : T^\infty \rightarrow A$. v_T^A is *nonintermittent* iff $v_T^A(t) = \perp_A$ implies $\forall t' \supset t, v_T^A(t') = \perp_A$ and it is *intermittent* otherwise. A location in constraint nets denotes a variable trace and a clock denotes a nonintermittent trace. A *total variable* is the set of all variable traces, and a *nonintermittent variable* is the set of all nonintermittent variable traces. A *variable* is either a total or a nonintermittent variable.

Proposition 4.1 *A variable is an algebraic cpo.*

A *transduction* is a mapping from a tuple of input variables to an output variable, $F_{\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_n} : \mathcal{V}_{\mathcal{T}_1}^{A_1} \times \ldots \times \mathcal{V}_{\mathcal{T}_n}^{A_n} \rightarrow \mathcal{V}_{\mathcal{T}_0}^{A_0}$ where $\mathcal{V}_{\mathcal{T}_i}^{A_i}$ is a variable with time structure $\mathcal{T}_i$ and variable domain A_i, which satisfies the causal relationship between its inputs and the output, i.e. if the inputs are the same up to a certain time point, the outputs will be the same up to that time. Formally, if $\mathcal{T}_r$ is a reference

time structure for all $\mathcal{T}_i$ with the reference mapping h_i, then for any pair of input variables v, v', and $t_0 \in \mathcal{T}_0^\infty$: $\forall i \forall t_i,\ h_i(t_i) \subseteq h_0(t_0) \rightarrow v_i(t_i) = v_i'(t_i)$ implies

$$F_{\mathcal{T}_0,\mathcal{T}_1,\ldots,\mathcal{T}_n}(v_1,\ldots,v_n)(t_0) = F_{\mathcal{T}_0,\mathcal{T}_1,\ldots,\mathcal{T}_n}(v_1',\ldots,v_n')(t_0)$$

For example, integration is a transduction. We will see that any finite automata defines a transduction from input variables to state variables. Clearly, transductions are closed under composition. A transduction is *continuous* if it is a continuous function. A transduction is a *transliteration* if it is a *pointwise* extension of some function f. Formally, the transliteration $f_{\mathcal{T}_0,\mathcal{T}_1,\ldots,\mathcal{T}_n} : \mathcal{V}_1 \times \ldots \times \mathcal{V}_n \rightarrow \mathcal{V}_0$,

$$f_{\mathcal{T}_0,\mathcal{T}_1,\ldots,\mathcal{T}_n}(v_1,\ldots,v_n) =_{def} \lambda t_0.f(v_1(\bigvee\{t_i|h_i(t_i) \subseteq h_0(t_0)\}),\ldots,v_n(\bigvee\{t_n|h_n(t_n) \subseteq h_0(t_0)\}))$$

It is easy to see that the following proposition holds.

Proposition 4.2 *If f is a continuous function, then $f_{\mathcal{T}_0,\mathcal{T}_1,\ldots,\mathcal{T}_n}$ is a continuous transduction.*

The following concepts are defined in order to relate variables and transductions with different time structures. If v is a variable trace with time structure $\mathcal{T}$, and $\mathcal{T}_r$ is a reference time structure of $\mathcal{T}$, then the *interpolated variable trace* of v onto $\mathcal{T}_r$ is a variable trace $\overline{v}$, $\overline{v} =_{def} id_{\mathcal{T}_r,\mathcal{T}}(v)$ where id is an identity function. Similarly, if v is a variable trace with time structure $\mathcal{T}_r$, the *sampled variable trace* of v onto $\mathcal{T}$ is a variable $\underline{v}$, $\underline{v} =_{def} id_{\mathcal{T},\mathcal{T}_r}(v)$. If $F_\mathcal{T}$ is a transduction whose variables have the same time structure, the *interpolated transduction* from $\mathcal{T}$ to $\mathcal{T}_r$ is defined as $\overline{F_\mathcal{T}}(v_1,\ldots,v_n) =_{def} \overline{F_\mathcal{T}(\underline{v_1},\ldots,\underline{v_n})}$. Similarly, the *sampled transduction* from $\mathcal{T}_r$ to $\mathcal{T}$ can be defined as $\underline{F_{\mathcal{T}_r}}(v_1,\ldots,v_n) =_{def} \underline{F_{\mathcal{T}_r}(\overline{v_1},\ldots,\overline{v_n})}$. Since identity functions are continuous, we have the following proposition.

Proposition 4.3 *If $F_\mathcal{T}$ is a continuous transduction, then $\overline{F_\mathcal{T}}$ and $\underline{F_\mathcal{T}}$ are continuous transductions.*

4.2 Unit delays and transducers

We shall see that composition of unit delays and transliterations can define complex transductions. Let $\mathcal{V}_\mathcal{T}^A$ be a variable. A *unit* delay is a transduction from $\mathcal{V}_\mathcal{T}^A$ to $\mathcal{V}_\mathcal{T}^A$. Formally, let $init \in A$ be the output value at the start time point, a unit delay in the time structure $\mathcal{T}$ is:

$$\delta_\mathcal{T}^A(init)(v) =_{def} \lambda t. \begin{cases} init & \text{if } t = \perp_{\mathcal{T}^\infty} \\ v(pre(t)) & \text{otherwise} \end{cases}$$

Proposition 4.4 *Unit delays are continuous transductions.*

A *state transducer* is a quadruple $\langle I, Q, q_0, f\rangle$ where I, the set of inputs, and Q, the set of states, are algebraic *cpos*, $q_0 \in Q$ is the initial state and $f : I \times Q \to Q$, the *state transition function*, is continuous. A state transducer is *finite* iff $|Q|$ is finite. A state transducer defines a continuous transduction from input variables to state variables. Formally, let $\mathcal{T}$ be a time structure:

Proposition 4.5 *A state transducer $\langle I, Q, q_0, f\rangle$ defines a transduction from $\mathcal{V}_{\mathcal{T}}^I$ to $\mathcal{V}_{\mathcal{T}}^Q$.*

Proof: Let $f_{\mathcal{T}}$ be the pointwise extension of f, $f_{\mathcal{T}} : \mathcal{V}_{\mathcal{T}}^I \times \mathcal{V}_{\mathcal{T}}^Q \to \mathcal{V}_{\mathcal{T}}^Q$. We have,

$$S_{\mathcal{T}}(v_i, v_q) \equiv \delta_{\mathcal{T}}^Q(q_0)(f_{\mathcal{T}}(v_i, v_q))$$

$S_{\mathcal{T}}$ is a continuous transduction since both $f_{\mathcal{T}}$ and $\delta_{\mathcal{T}}^Q(q_0)$ are continuous transductions. According to Proposition 3.2 there is a continuous function which is the least fixpoint of $S_{\mathcal{T}}$, $\mu.S_{\mathcal{T}} : \mathcal{V}_{\mathcal{T}}^I \to \mathcal{V}_{\mathcal{T}}^Q$. Clearly $\mu.S_{\mathcal{T}}$ is a transduction. □

Let $\overline{B} = \{0, 1, \perp\}$ be a flat *cpo*. An *event transduction* is a transduction whose output is a nonintermittent variable with domain isomorphic to $\overline{B}$.

Example 4.1 *The negated Muller-C element used in Example 2.1 is the basic "and" logic element in event synchronization [12]. A negated Muller-C element is a state transducer: $\langle \overline{B} \times \overline{B}, \overline{B}, 0, f\rangle$ where*

$$f(\langle i_1, i_2\rangle, q) =_{def} \begin{cases} \perp & \text{if } q = \perp \text{ or } i_1 = \perp \text{ or } i_2 = \perp \\ i_1 & i_1 \neq i_2 \\ q & \text{otherwise} \end{cases}$$

So this defines an event transduction $NC : \mathcal{V}_{\mathcal{T}}^{\overline{B}} \times \mathcal{V}_{\mathcal{T}}^{\overline{B}} \to \mathcal{V}_{\mathcal{T}}^{\overline{B}}$, given any time structure $\mathcal{T}$.

A *transducer* is a tuple $\langle I, Q, q_0, f, O, f^o\rangle$ where $\langle I, Q, q_0, f\rangle$ is a state transducer, O, the set of outputs, is an algebraic *cpo* and $f^o : I \times Q \to O$, the *output function*, is continuous. A transducer defines a continuous transduction from input variables to output variables, for any given time structure $\mathcal{T}$. Formally, $F_{\mathcal{T}} : \mathcal{V}_{\mathcal{T}}^I \to \mathcal{V}_{\mathcal{T}}^O$, $F_{\mathcal{T}} \equiv \lambda v_i . f_{\mathcal{T}}^o(v_i, (\mu.S_{\mathcal{T}})(v_i))$.

4.3 Clock traces and Σ-dynamics

Let A_b be any continuous boolean algebra (ternary algebra) with its carrier set isomorphic to $\overline{B}$. A *clock trace* is a nonintermittent variable trace $c_{\mathcal{T}_r}^{A_b}$, which generates an accurate sampled time structure $\mathcal{T}_c$ of

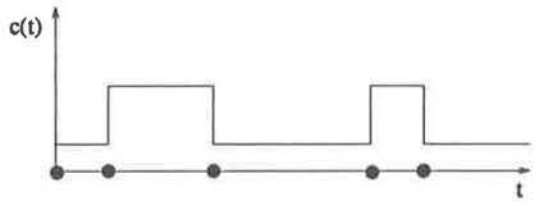

Figure 2: A clock trace: each dot depicts a time point of $\mathcal{T}_c$, $\mathcal{T}_c \subseteq \mathcal{T}_r$

$\mathcal{T}_r$. Formally, $\mathcal{T}_c \subseteq \mathcal{T}_r$ is defined as: $\mathcal{T}_c =_{def} \{\perp_{\mathcal{T}_r}\} \cup \{t \in \mathcal{T}_r | c(in(t)) \neq \perp_{A_b}, c(in(t)) \neq c(pre(in(t)))\}$, i.e. each transition defines a time point (see Figure 2), and $\forall t \in \mathcal{T}_c,\ m_c(in(t)) = m(in(t))$. Clock traces can be produced by event transductions.

A *clock variable* $\mathcal{C}_{\mathcal{T}_r}$ is the set of all clock traces on reference time structure $\mathcal{T}_r$. Let $F_{\mathcal{T}_c} : \mathcal{V}^I_{\mathcal{T}_c} \rightarrow \mathcal{V}^O_{\mathcal{T}_c}$ be a transduction with time structure $\mathcal{T}_c$ produced by clock trace c. We define a transduction with a clock variable on the reference time structure $\mathcal{T}_r$ as $F^c_{\mathcal{T}_r} : \mathcal{C}_{\mathcal{T}_r} \times \mathcal{V}^I_{\mathcal{T}_r} \rightarrow \mathcal{V}^O_{\mathcal{T}_r}$,

$$F^c_{\mathcal{T}_r}(c, v_i) =_{def} \lambda t. \begin{cases} \overline{F_{\mathcal{T}_c}}(v_i)(t) & \text{if } c(t) \neq \perp_{A_b} \\ \perp_O & \text{otherwise} \end{cases}$$

Proposition 4.6 *If $F_{\mathcal{T}_c}$ is a continuous transduction, then $F^c_{\mathcal{T}_r}$ is a continuous transduction.*

Proof: It is easy to see that $F^c_{\mathcal{T}_r}$ is both left and right continuous if $F_{\mathcal{T}_c}$ is continuous, therefore it is continuous [4]. □

Finally, with preliminaries established, we can characterize the domain structure for constraint nets. A Σ-*dynamics* is defined on a Σ-domain and a reference time structure. Let $\Sigma \equiv \langle S, F \rangle$ be a signature and $b \in S$ be a sort denoting boolean. If A is a Σ-domain, a Σ-dynamics $\mathcal{D}(\mathcal{T}_r, A)$ is a triple $\langle \mathcal{V}, \leq_{\mathcal{V}}, \mathcal{F} \rangle$ where

- $\mathcal{V} \equiv \{\mathcal{V}^{A_s}_{\mathcal{T}_r}\}_{s \in S} \cup \mathcal{C}_{\mathcal{T}_r}$ where $\mathcal{V}^{A_s}_{\mathcal{T}_r}$ is a total variable and $\mathcal{C}_{\mathcal{T}_r}$ is a clock variable;
- $\leq_{\mathcal{V}} \equiv \{\leq_{\mathcal{V}^{A_s}_{\mathcal{T}_r}}\}_{s \in S}$ is the set of partial orders on variables, $v_1 \leq_{\mathcal{V}^{A_s}_{\mathcal{T}_r}} v_2$ iff $\forall t \in \mathcal{T}_r, v_1(t) \leq_{A_s} v_2(t)$;
- $\mathcal{F} \equiv \mathcal{F}_{\mathcal{T}_r} \cup \mathcal{F}^c_{\mathcal{T}_r}$ where $\mathcal{F}_{\mathcal{T}_r} \equiv \{f^A_{\mathcal{T}_r}\}_{f \in F} \cup \{\delta^{A_s}_{\mathcal{T}_r}(init)\}_{s \in S, init \in A_s}$ is the set of transliterations and unit delays and $\mathcal{F}^c_{\mathcal{T}_r} \equiv \{F^c | F \in \mathcal{F}_{\mathcal{T}_r}\}$ is the set of transductions with clock variables.

Theorem 4.1 *Given a Σ-dynamics $\mathcal{D}(\mathcal{T}_r, A)$ consisting of a triple $\langle \mathcal{V}, \leq_{\mathcal{V}}, \mathcal{F} \rangle$ then (1) $\langle \mathcal{V}, \leq_{\mathcal{V}} \rangle$ is a multi-sorted algebraic cpo, and (2) each transduction in $\mathcal{F}$ is continuous.*

5 The Semantics of Constraint Nets

Now we present a denotational semantics for constraint nets based on fixpoint theory. Let $\mathcal{D}(\mathcal{T}_r, A)$ be a Σ-dynamics consisting of a triple $\langle \mathcal{V}, \leq_{\mathcal{V}}, \mathcal{F} \rangle$ and CN be a constraint net consisting of a triple $\langle Lc, Td, Cn \rangle$. Semantically, each location in Lc with sort s denotes a variable trace $v \in \mathcal{V}^{A,\bullet}_{\mathcal{T}_r}$ and each clock denotes a clock trace $c \in \mathcal{C}_{\mathcal{T}_r}$. Each transduction in Td is a composition of transductions in $\mathcal{F}$ with the same time structure (with at most one clock variable). Each connection relates an input/output variable of a transduction with a location. Therefore, the semantic representation of a constraint net is a set of equations, where each left-hand side is an individual output location and each right-hand side is an expression composed of transductions and locations: $\vec{o} = \vec{F}_{\mathcal{T}_r}(\vec{i}, \vec{o})$ where $\vec{o}$ is the tuple of output locations, $\vec{i}$ is the tuple of input locations and $\vec{F}_{\mathcal{T}_r}$ is the tuple of transductions. Since each transduction is continuous, $\vec{F}_{\mathcal{T}_r}$ is a continuous transduction from $\vec{\mathcal{V}}_i \times \vec{\mathcal{V}}_o$ to $\vec{\mathcal{V}}_o$. According to Proposition 3.2 (fixpoint theory), there is a continuous function $\mu.\vec{F}_{\mathcal{T}_r}$ which is the least fixpoint of this equation. We call $\vec{o} = \mu.\vec{F}_{\mathcal{T}_r}(\vec{i})$ the set of *semantic equations* of the constraint net, where $\mu.\vec{F}_{\mathcal{T}_r}$ is a tuple of continuous transductions. The semantics of a constraint net CN, $[\![CN]\!]$, is defined as: $[\![CN]\!](o_k) = \mu.\vec{F}_{\mathcal{T}_r k}$. Note that $[\![CN]\!]$ also denotes the *trajectory* of the dynamic system being modeled.

Example 5.1 *Consider the subnet in Figure 1 consisting of locations R1, R2, C1 and C2, and the two negated Muller-C elements. We have*

$$C1 = NC(R1, C2), \quad C2 = NC(C1, R2) \tag{1}$$

The semantic equations of this subnet are $C1 = c1(R1, R2), \quad C2 = c2(R1, R2)$ *where* $(c1, c2)$ *is the fixpoint of Equation 1.*

For a complex system with many components, the semantics of the whole system can be obtained by the semantics of its components.

- *composition*: If $\langle CN, O \rangle \equiv \langle CN_1, O_1 \rangle \parallel \langle CN_2, O_2 \rangle$, then $[\![CN]\!]$ is obtained as follows: if $l \in O(CN_1)$, then $[\![CN]\!](l) =_{def} [\![CN_1]\!](l)$; if $l \in O(CN_2)$, then $[\![CN]\!](l) =_{def} [\![CN_2]\!](l)$.
- *coalescence*: If $\langle CN, O \rangle \equiv \langle CN', O \rangle / E$, then $[\![CN]\!]$ is obtained as follows: let $\vec{o} = \vec{F}(\vec{i})$ be the set of semantic equations of CN', the set of equations for CN is: $\vec{o} = \vec{F}([\vec{i}])$ where $[\vec{i}]_k = [i_k]$ denotes

the quotient from the equivalence relation of $\overline{E}$. If $\mu.\vec{F}$ is the least fixpoint of the equations, then $[\![CN]\!](o_k) = \mu.\vec{F}_k$.

Example 5.2 *Consider again the net in Figure 1. Let $B = producer(Clock)$ be the semantic equation for "producer". By coalescing Clock with $C1$ we have*

$$B = producer(C1), \quad C1 = c1(R1, R2)$$

And the semantic equations for this subnet are: $B = producer(c1(R1, R2)), \quad C1 = c1(R1, R2)$.

6 Behavioral Specification and Verification

We discuss in this section the behavioral specification and verification of constraint nets. Let $\Sigma \equiv \langle S, F\rangle$ be the signature of a constraint net. There may be another signature $\Sigma^+ \equiv \langle S^+, F^+\rangle$ which is an augmented signature of Σ, i.e. $S \subseteq S^+$ and $F \subseteq F^+$. Intuitively, S^+ may include qualitative or more abstract data types of S, and F^+ may include the functions that relate sorts in S with those in S^+.

There are various levels of specifications for constraint nets:

- *algebraic specification*: a set of equalities and/or inequalities between terms in Σ^+-algebra,
- *implementation specification*: the set of equations of a constraint net in Σ-dynamics,
- *requirement specification*: the set of relations between input and output locations in Σ^+-dynamics; usually, the requirement specification is more abstract than the implementation specification.

Temporal logic has been used as a specification and verification tool for concurrent programs [6]. A temporal logic on Σ-algebras is developed for our purposes. A temporal formula is built from elementary formulas using boolean operators and temporal operators:

$$P = p \mid \neg P \mid P_1 \rightarrow P_2 \mid \Box P \mid \Diamond P \mid \bigcirc P \mid P_1 \, U \, P_2$$

A frame $\mathcal{F}_r$ is a pair $\langle A, I\rangle$ where A is a Σ-domain, I is an interpretation which maps each predicate p in P to true or false, given the values of its arguments. A model $\mathcal{M}$ is a triple $\langle \mathcal{F}_r, \mathcal{T}_r, \sigma\rangle$ where $\mathcal{F}_r$ is a frame, $\mathcal{T}_r$ is a reference time, σ is a mapping from locations to variable traces. Let $\models_t$ be the relation between model $\mathcal{M}$ and formula P at time point $t \in \mathcal{T}_r^\infty$:

- $\mathcal{M} \models_t p$: $p[\sigma(l)(t)/l,\ \sigma(l)(pre(t))/pre(l)]$;
- $\mathcal{M} \models_t \Box P$: $\forall t' \supseteq t, \mathcal{M} \models_{t'} P$;
- $\mathcal{M} \models_t \Diamond P$: $\exists t' \supseteq t, \mathcal{M} \models_{t'} P$;
- $\mathcal{M} \models_t \bigcirc P$: $\forall t' \in succ(t), \mathcal{M} \models_{t'} P$;
- $\mathcal{M} \models P_1\, U\, P_2$: $\exists t', t' \supseteq t$ and $\mathcal{M} \models_{t'} P_2$ and $\forall t'', t \subseteq t'' \subset t', \mathcal{M} \models_{t''} P_1$.

$\mathcal{M} \models P$ iff $\mathcal{M} \models_{\perp_{\mathcal{T}^\infty}} P$.

The logical specification is a powerful language for specify various qualitative behaviors of a real-time embedded system. Some important properties of systems are:

1. *asymptotic stability*: if P is a stable property, $\Diamond\Box P$;
2. *persistence*: if P is a goal to be tracked, $\Box\Diamond P$;
3. *safety*: if P is a dangerous behavior, $\Box(\neg P)$;
4. *response*: if E is an event specification, R is an response specification, $\Box(E \rightarrow \Diamond R)$;
5. *progress*: if E_i is an event, R_i is a possible response of E_i, $\Box(E_1 \wedge \ldots \wedge E_n \rightarrow R_1 \vee \ldots \vee R_n)$.

Verification is a process of demonstrating that the implementation specification implies the requirement specification. There are two approaches; both can be used in our framework. The first is model checking. Let CN be a constraint net with semantic equation $\vec{o} = \vec{F}(\vec{i})$, and $\mathcal{R}(O, I)$ be the requirement specification. CN *satisfies* $\mathcal{R}$, written $CN \models \mathcal{R}$, iff $\forall \vec{i}, \mathcal{R}(\vec{F}(\vec{i}), \vec{i})$. The second is theorem proving. Let $\mathcal{I}$ be the implementation specification and $\mathcal{R}$ be the requirement specification, then show that $\vdash \mathcal{I} \rightarrow \mathcal{R}$.

Example 6.1 *Consider the asynchronous event controller in Figure 1. The algebraic specification includes the specification of boolean algebra. The implementation specification, written in temporal logic formula is:*

$$(C1 = 0) \bigwedge \Box(R1 = C2 \rightarrow \bigcirc(C1 = pre(C1)) \bigwedge \Box(R1 \neq C2 \rightarrow \bigcirc(C1 = pre(R1))$$

$$(C2 = 0) \bigwedge \Box(C1 = R2 \rightarrow \bigcirc(C2 = pre(C2)) \bigwedge \Box(C1 \neq R2 \rightarrow \bigcirc(C2 = pre(C1))$$

Let $e(x)$, where x is a clock, denote $x \neq pre(x)$, i.e. an event occurs at x, and $ne(x)$ denote the number of events that have occurred at x. The requirement specification is:

- safety property: the size of the buffer is one, i.e. $\mathcal{S} \equiv \Box(0 \leq ne(C1) - ne(C2) \leq 1)$
- response: a request will be handled as soon as possible, i.e.

$$\mathcal{R}_1 \equiv \Box(e(R1) \rightarrow (\bigcirc(e(C1)) \vee \Box(e(C2) \rightarrow \bigcirc e(C1))))$$

$$\mathcal{R}_2 \equiv \Box(e(R2) \rightarrow (\bigcirc(e(C2)) \vee \Box(e(C1) \rightarrow \bigcirc e(C2))))$$

However, in order to satisfy this specification, the following interface protocol must be satisfied:

- initialization: $\mathcal{I}_0 \equiv (R1 = 0) \wedge (R2 = 1)$
- synchronization: $R1$ shall not create another request until $C1$ generates an event, $R2$ shall not create another request until $C2$ generates an event, i.e.

$$\mathcal{E} \equiv \Box(e(R1) \rightarrow R1 \neq C1) \bigwedge \Box(e(R2) \rightarrow R2 = C2)$$

As a result the requirement specification for an asynchronous event controller is: $\mathcal{I}_0 \wedge \mathcal{E} \rightarrow \mathcal{S} \wedge \mathcal{R}_1 \wedge \mathcal{R}_2$. And we can prove that the implementation specification implies the requirement specification.

7 Conclusion and Future Work

We have presented a semantic model, Constraint Nets, for real-time embedded systems based on algebraic theory. With its rigorous formalization, the Constraint Net model serves as a foundation for specification, verification, analysis and simulation of the complete dynamic system. We have been able to model robotic behaviors with constraint nets which are simulated by logical concurrent objects [14]. We plan to develop further a visual programming and simulation environment, known as ALERTS (A Laboratory for Embedded Real-Time Systems), based on the Constraint Net model. Using ALERTS, a system can be designed hierarchically, and simulated or verified incrementally.

References

[1] E.A. Ashcroft. Dataflow and eduction: Data-driven and demand-driven distributed computation. In J. W. deBakker, W.P. deRoever, and G. Rozenberg, editors, *Current Trends in Concurrency*, number 224 in Lecture Notes on Computer Science. Springer-Verlag, 1986.

[2] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time petri nets. *IEEE Transactions on Software Engineering*, 17(3), March 1991.

[3] P. Caspi, D. Pilaud, N. Halbwachs, and J. A. Plaice. Lustre: A declarative language for programming synchronous systems. In *ACM Proceeding of Principles of Programming Languages*, 1987.

[4] M. Hennessy. *Algebraic Theory of Processes.* The MIT Press, 1988.

[5] M. Joseph. Time and real-time in programs. In C.E. Veni Madhavan, editor, *Foundations of Software Technology and Theoretical Computer Science*, number 405 in Lecture Notes on Computer Science, pages 312 – 324. Springer-Verlag, 1989.

[6] L. Lamport. The temporal logic of actions. Technical Report 79, Digital Systems Research Center, Palo Alto, California, December 1991.

[7] J. Lavigon and Y. Shoham. Temporal automata. Technical Report STAN-CS-90-1325, Robotics Laboratory, Computer Science Department, Stanford University, Stanford, CA 94305, 1990.

[8] M. Merritt, F. Modugno, and M.R. Tuttle. Time-constrained automata. In J.C.M. Baeten and J.F. Groote, editors, *CONCUR-91*, number 527 in Lecture Notes on Computer Science, pages 393 – 407. Springer-Verlag, 1991.

[9] F. Moller and C. Tofts. A temporal calculus of communicating systems. In J.C.M. Baeten and J.W. Klop, editors, *CONCUR-90*, number 458 in Lecture Notes on Computer Science, pages 401 – 415. Springer-Verlag, 1990.

[10] C.A. Petri. "Forgotten topics" of net theory. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Applications and Relationships to Other Models of Concurrency*, number 255 in Lecture Notes on Computer Science, pages 500 – 514. Springer-Verlag, 1986.

[11] G.M. Reed and W. Roscoe. A timed model for communicating sequential processes. In Laurent Kott, editor, *Automata, languages and programming*, number 226 in Lecture Notes on Computer Science, pages 314 – 323. Springer-Verlag, 1986.

[12] I.E. Sutherland. Micropipeline. *Communication of ACM*, 32(6), June 1989.

[13] W.W. Wadge and E.A. Ashcroft. *Lucid, the dataflow programming language.* Academic Press, 1985.

[14] Y. Zhang and A.K. Mackworth. Modeling behavioral dynamics in discrete robotic systems with logical concurrent objects. In S.G. Tzafestas and J.C. Gentina, editors, *Robotics and Flexible Manufacturing Systems.* Elsevier Science Publishers B.V., 1992. (in press).