bCANDLE: Formal Modelling and Analysis of CAN Control Systems

D.Kendall S.P.Bradley W.D.Henderson
A.P.Robson
Department of Computing, University of Northumbria at Newcastle
Ellison Place, Newcastle upon Tyne, NE1 8ST
david.kendall@unn.ac.uk

Abstract

Embedded control systems appear in many of the manufactured products upon which our society increasingly depends. System developers need better development methods in order to be more confident that the systems which they deliver will behave properly. The need is particularly pressing in the case of distributed, hard real-time control systems for which testing is notoriously difficult. In recent years, much research has been conducted into formal techniques for analysing the quantitative temporal properties of system models. Such work offers the promise of complementing testing in the validation of systems by approaches which include simulation, symbolic monitoring, assertion checking and verfication.

The principal contribution of this paper is the introduction of a modelling language, bCANDLE, whose intended domain comprises embedded control systems in which computing nodes communicate using one or more Controller Area Networks (CAN). bCANDLE is a simple but expressive language which includes value passing broadcast communication, message priorities and an explicit time construct. In giving a formal semantics to bCANDLE in terms of timed transition systems, we present for the first time an abstract, timed formal model of CAN.

1. Introduction

This paper introduces *bCANDLE*, a language for the modelling and analysis of distributed, real-time control systems which communicate using the deterministic broadcast communication protocol, CAN [9]. *bCANDLE* (pronounced 'basic candle') is a simple but expressive process language with a value-passing, broadcast communication primitive, message priorities and an explicit time construct. The language is given a formal semantics in terms of timed transition systems [8].

Broadcast communication is used frequently in the implementation of embedded systems, but has received comparatively little attention from the formal methods community in contrast to point-to-point synchronous communication. Timed transition systems have proved to be very successful models for the analysis of real-time systems [8] and they arise naturally from a variety of formalisms for system description, see for example [1, 6]. However, the construction of a timed transition model of a broadcasting embedded control system poses a number of problems: for example, timed automata are too low-level to allow a manageable description of realistic systems; other formalisms, with a point-to-point, synchronous communication primitive, such as ET-LOTOS, are not convenient for the description of broadcasting systems (see [12] for a convincing justification of this point of view); yet other formalisms which do adopt a broadcast communication primitive, such as Esterel [2], assume the viability of the synchrony hypothesis which is not sustainable for distributed systems. Therefore, we have been motivated to develop an approach to the description of broadcasting systems which adopts a broadcast mechanism as its communication primitive, with the intention of making recent progress in real-time systems analysis as accessible to developers of these systems as it is to others.

We wish to promote the production of formal system models which arise almost as a by-product of a 'natural' development process. In this respect, the approach is similar to [3]and [7]. A model is intended to be a conservative approximation of its associated implementation, i.e. the behaviours of the implementation should be a subset of the behaviours of the model. Given such a conservative approximation, by restricting attention to requirements which are expressed as properties of *all* behaviours of the model, it is sufficient to establish that the model satisfies a requirement in order to conclude that the implementation also satisfies the requirement. This approach is similar in some respects to the timing analysis of Ada programs undertaken by Corbett [5]. However, the work described there is restricted to single processor systems, whereas we are concerned pri-

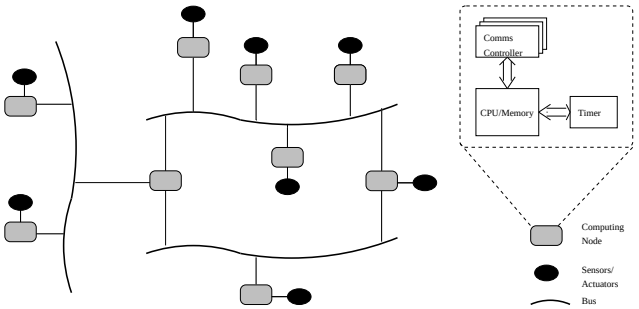

Figure 1. Control system model

marily with distributed systems.

2. Informal control system model

Figure 1 shows a typical organisation for the class of control systems to be studied. Control is distributed over a number of *tasks* which are statically allocated to computing nodes. Several tasks may be allocated to a single node and share the processing unit using some fixed scheduling policy. Tasks communicate by using one or more communication channels (buses) to send and receive broadcast messages. Access to each channel is mediated by a dedicated communication controller. This is the only (logical) mechanism for communication between tasks. Tasks do not share memory. In addition, each computing node may have access to a number of sensors and actuators which form part of the interface to the controlled system. In the case of multi-tasking, it is assumed that sensors and actuators are not shared but that each is accessed by a single task.

3. Formal Preliminaries

3.1. Introduction

The formalization of the class of models described in section 2 is undertaken by the definition of the modelling language, *bCANDLE*. A *bCANDLE* model integrates models of the structure and behaviour of tasks, of the network, and of the global data environment (see [10] for a similar approach). The semantics of a *bCANDLE* model is given as a timed transition system as described in the following sections.

3.2. Time domain

We use a dense time domain, $\mathbb{R}$, which we take to be the non-negative reals. It is convenient to augment the time domain with a value, ∞, which is defined to be strictly greater than any other time value. We write $\mathbb{R}_\infty$ for $\mathbb{R} \cup \{\infty\}$ and assume that the arithmetic operators and relations are extended to $\mathbb{R}_\infty$ in the usual way: for every $t \in \mathbb{R}$, $t < \infty$; and for every $t \in \mathbb{R}_\infty$, $t + \infty = \infty + t = \infty$. We also define a subtraction operation, $- : \mathbb{R}_\infty \times \mathbb{R} \to \mathbb{R}_\infty$, such that $t_1 - t_2 = t$ where $t_1 = t_2 + t$, if $t_1 > t_2$ and $t = 0$ otherwise.

4. Modelling the data environment

It is common to omit the data environment from system models whose purpose is the analysis of concurrent and real-time properties. This leads to smaller models which are consequently easier to analyse. However, the behaviour of real systems is influenced by their data and we need to be able to reason about these effects. Therefore we include a model of (at least part of) the data environment in our initial system models and choose appropriate abstractions later in the analysis when it becomes clearer which, if any, properties of the data environment are relevant to the system properties of interest.

Definition 1 (Data Environment) Let *Var* be a finite set of variable names, V a finite set of data values, Ω a finite set of operation names and Γ a finite set of predicate names. A *data environment* over Var, V, Ω and Γ is a mapping $D : Var \to V$ with the following operations: a lookup operation $D.x$, which for any variable $x \in Var$ denotes the value of x in the map D; an update operation $D[x := v]$, which denotes a data environment D', which is the same as D except that the variable x is associated with the value v in D'; a binary relation $\stackrel{\omega}{\Longrightarrow}$, for each operation symbol $\omega \in \Omega$, such that $D \stackrel{\omega}{\Longrightarrow} D'$ iff the operation associated with ω can be executed in D producing a new state D'. We use the operation label ID to represent the operation which leaves every data environment unchanged: $\forall D, D' \bullet D \stackrel{ID}{\Longrightarrow} D' \Leftrightarrow D = D'$. We also define a binary relation, $\models$, such that for any predicate symbol $\gamma \in \Gamma$, $D \models \gamma$ iff the predicate associated with γ is satisfied in D. We write $D \not\models \gamma$ for $\neg (D \models \gamma)$. We assume the existence of distinguished predicate symbols `true` and `false`, such that $\forall D \bullet D \models$ `true` and $\forall D \bullet D \not\models$ `false`. We use data operations not only to model changes of internal state but also to model interaction with the external environment. So it is convenient to be able to distinguish independent operations from operations which synchronise with the environment. To do so, we define a function, $\bar{} : \Omega \to \Omega \cup \{\tau\}$ such that for all $\omega \in \Omega$ either $\bar{\omega} = \omega$ or $\bar{\omega} = \tau$ where $\tau \notin \Omega$ represents an independent operation.

■

5. Modelling the Network

Definition 2 (Channel) Let I be a finite set of message identifiers and V a finite set of data values. A *channel* $c = (M, \preceq, \delta, q, s)$ over I, V, is a tuple where

- $M \subseteq I \times V$ is the set of messages which can be transmitted on the channel. We use the notation $i.v = m$, where $i \in I$ and $v \in V$, to decompose a message $m \in M$.
- The relation $\preceq : M \leftrightarrow M$ is a total, reflexive, transitive ordering on the channel's messages, which gives the priority at which they are transmitted. For messages $m_1, m_2 \in M$, $m_1 \preceq m_2$ iff the priority of m_1 is at least as high as the priority of m_2. We write $m_1 \prec m_2$ if $m_1 \preceq m_2$ and $m_2 \not\preceq m_1$. If $m_1 \prec m_2$, m_1 will be transmitted before m_2, in the event that both messages simultaneously contend for transmission on their channel. If two messages of the same priority seek transmission simultaneously[1], an arbitrary choice is made as to which is transmitted first.
- δ is a collection of functions $\delta^{\mathbf{lb}}, \delta^{\mathbf{ub}}, \delta^{\mathbf{lB}}, \delta^{\mathbf{uB}} : M \to \mathbb{R}_\infty$ which give the lower and upper bounds on the duration of the pre- and post-acceptance phases for the transmission of a message on the channel.[2] We write **lb** (resp. **ub**, **lB**, **uB**) for $\delta^{\mathbf{lb}}(m)$ (resp. $\delta^{\mathbf{ub}}(m), \delta^{\mathbf{lB}}(m), \delta^{\mathbf{uB}}(m)$) when m is clear from the context.
- q is a priority ordered queue of messages which are pending transmission on the channel. An empty queue is denoted $\langle\rangle$. A queue with highest priority message m and remaining messages q is written $m{:}q$. An insertion operator $\hookleftarrow\!\!\rho\colon \operatorname{seq} M \times M \to \operatorname{seq} M$, which preserves only the most recently inserted message with a given identifier, is defined[3]

$$q \hookleftarrow\!\!\rho\ i.v = \begin{cases} \langle i.v \rangle & , q = \langle\rangle \\ i.v{:}q' & , q = i.\_{:}q' \\ i.v{:}m{:}q' & , q = m{:}q',\ i.v \preceq m \\ m{:}(q' \hookleftarrow\!\!\rho\ i.v) & , q = m{:}q',\ m \prec i.v \end{cases}$$

- s represents the current status of the channel. A channel is either *free* or is transmitting a message. The transmission of a message $m \in M$ requires some time (> 0) to complete (i.e. $\delta^{\mathbf{lb}}(m) + \delta^{\mathbf{lB}}(m) > 0$, for all $m \in M$). The key instant during transmission is the *acceptance* point for m, by which time, listeners to the channel which intend to accept m, must have indicated that intention; m will not be accessible to any listeners which have not indicated their intention to accept it by that time. The acceptance point is preceded (resp. succeeded) by a *pre-acceptance* (resp. *post-acceptance*) phase of a transmission. The pre-acceptance phase extends from the start of transmission to the acceptance point. The post-acceptance phase extends from the acceptance point to the first instant at which the channel next becomes free. During each of these phases, time can progress within the limits given by the bounds on the duration of the phase. We use the notation shown in figure 2 to denote the status of a channel.

[1] This should not occur in practice. If it does, it probably indicates an error condition which should be detected and corrected.

[2] Pre- and post-acceptance phases of message transmission are explained below in the discussion of channel status.

[3] In this and subsequent sections, we make use of $\_$ to denote an arbitrary value taken from whatever set of values is appropriate in its context.

Notation	Channel Status
$\downarrow$	free
$\overset{[l,u]}{\rightsquigarrow} m$	pre-acceptance phase of transmission of message m with bounds $[l,u]$ on time to completion, $0 \le l \le \mathbf{lb}$, $0 \le u \le \mathbf{ub}$
$\uparrow m$	acceptance point in transmission of m
$\overset{[l,u]}{\rightsquigarrow}$	post-acceptance phase of message transmission with bounds $[l,u]$ on time to completion, $0 \le l \le \mathbf{lB}$, $0 \le u \le \mathbf{uB}$

Figure 2. Channel Status Notation: ($m \in M$ **and** $l, u \in \mathbb{R}_\infty$ **)**

For a channel $c = (M, \preceq, \delta, q, s)$, we define the sets $ids_c = \{i \in I \mid \exists v : V \bullet i.v \in M\}$, $values_c = \{v \in V \mid \exists i : I \bullet i.v \in M\}$, and $channels_c = \{(M, \preceq, \delta, q', s') \mid q' \in \operatorname{seq} M \text{ and } s' \text{ is a possible status of } c\}$ ■

5.1. Channel Behaviour

The behaviours of a channel $c = (M, \preceq, \delta, q, s)$ are given by the timed transition system $(S, L, c, \longrightarrow_{\mathrm{C}})$ where: $S = channels_c$ is the set of states; $L = \{\nu\} \cup \{(t) \mid t \in \mathbb{R}\}$ is the set of labels in which ν indicates an internal channel transition and (t) indicates a time transition of duration t; $c \in S$ is the initial state; and $\longrightarrow_{\mathrm{C}} \subseteq S \times L \times S$ is the least relation closed under the rules of figure 3.

Definition 3 (Network) Let K be a finite set of channel identifiers. A *network* N over K is an indexed set of channels, $N = ((M, \preceq, \delta, q, s)_k \mid k \in K)$. For any $k \in K$, we denote the channel indexed by k as N_k.

If N is a network over K, we can further distinguish N by saying that N is a network over K, I, V, where $I =$

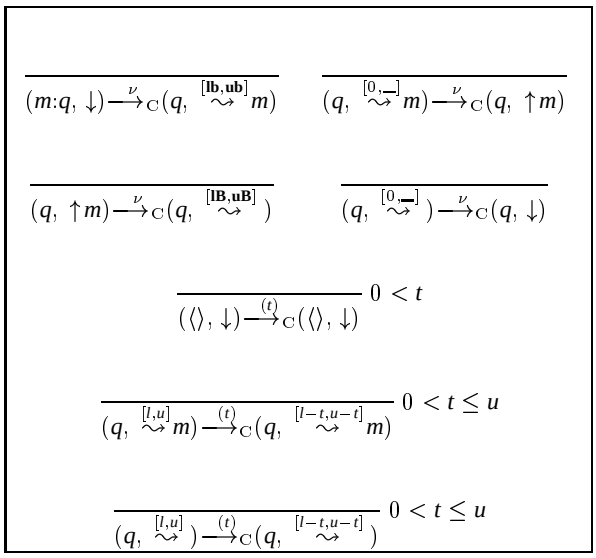

Figure 3. Channel Transitions: The transition rules show only the *dynamic* components of a channel (i.e. the pending message queue and channel status). The presence of the *static* components ($M, \preceq$ and δ) is to be assumed throughout. $t \in \mathbb{R}$ and $l, u \in \mathbb{R}_\infty$.

$\bigcup_{k\in K} ids_{N_k}$ and $V = \bigcup_{k\in K} values_{N_k}$.

Let K be a set of channel identifiers, N a network over K, $k \in K$, $c = N_k$ and $c' \in channels_c$. The operation $N \oplus c'_k$ is defined: $N \oplus c'_k = N \setminus \{c_k\} \cup \{c'_k\}$. ■

5.2. Network Behaviour

Let K be a set of channel identifiers and N a network over K. A channel $c = N_k$ can act independently, by performing any channel transition of which it is capable, to become c', giving a new network N' in which c_k is replaced by c'_k. By contrast, in order for N to perform a time transition, *all* channels in N must be capable of performing the transition together. Figure 4 gives transition rules which allow the extension of the communication model to a network of channels.

6. Modelling Task Behaviour

We use a simple process language to describe the behaviour of tasks. In choosing the operators of the language, we have been concerned to identify a small set which allows us to express naturally the behavioural models in which we are interested, while allowing the definition of a timed transition semantics in a direct manner.

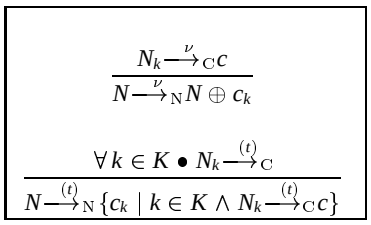

Figure 4. Network Transitions

6.1. Syntax

Given finite sets K of channel identifiers, I of message identifiers, *Var* of data variables, Ω of operation names and Γ of predicate names, the set of process terms over K, I, Var, Ω and Γ is given by the grammar:
$P ::= k!i.x \mid k?i.x \mid [\omega : t_1, t_2] \mid \gamma \texttt{ -> } P \mid P \texttt{ ; } Q \mid P + Q \mid P \texttt{ [> } Q \mid P \texttt{ | } Q \mid \mathrm{rec}\, X.P \mid X$ where $k \in K$, $i \in I$, $x \in Var$, $\omega \in \Omega$, $\gamma \in \Gamma$. P and Q are process terms, X is a process variable and $t_1, t_2 \in \mathbb{R}_\infty$. These terms represent *basic processes* which: enqueue a message for transmission on a channel, accept a message from a channel, perform a computation within a bounded period of time, and evaluate a guard on the data environment; and, *compound processes* formed by: sequential composition, choice, interrupt, and parallel composition. The operators bind from tightest to loosest according to the precedence ordering: `;, ->, +, [>, |`.

Repetitive behaviour is modelled by recursion: $\mathrm{rec}\, X.P$. The free variables of a term are those which are not bound by some recursion. Closed terms are terms without free variables. We denote by $Proc_{\mathbf{Sys}}$ the set of closed terms which do not violate restrictions on the use of recursion: in particular, all use of recursion must be guarded by some non-zero time delay; parallel composition is not allowed inside recursion; and a free process variable must not appear in a sub-expression which constitutes the left operand of an interrupt operator. These restrictions ensure that our models enjoy the properties of finite variability, finite control and representability using a finite number of clocks (see [8] for an explanation of the importance of these properties to verification).

We use a number of syntactic abbreviations: $[t_1] \equiv [ID : t_1]$, $[\omega : t_1] \equiv [\omega : t_1, t_1]$, $[t_1, t_2] \equiv [ID : t_1, t_2]$, $\mathrm{skip} \equiv [0]$, and $\mathrm{idle} \equiv \texttt{false -> skip}$.

We also make use of equational definitions, exploiting the property that processes defined using a set of simultaneous equations have an equivalent description entirely in terms of the recursion operator.

$$\textbf{Snd.1}\ \frac{}{(k!i.x,N,D)\xrightarrow{\tau}(\checkmark,N\oplus(q\leftarrow i.v,s)_k,D)}\ N_k=(q,s)\wedge v=D.x$$

$$\textbf{Snd.2}\ \frac{N\xrightarrow{\nu}_{\mathrm{N}}N'}{(k!i.x,N,D)\xrightarrow{\nu}(k!i.x,N',D)}$$

$$\textbf{Rcv.1}\ \frac{}{(k?i.x,N,D)\xrightarrow{\tau}(\checkmark,N,D[x:=v])}\ N_k=(\_,\uparrow i.v)$$

$$\textbf{Rcv.2}\ \frac{N\xrightarrow{\lambda_{NT}}_{\mathrm{N}}N'}{(k?i.x,N,D)\xrightarrow{\lambda_{NT}}(k?i.x,N',D)}\ N_k\neq(\_,\uparrow i.\_)\vee N_k=N'_k$$

$$\textbf{Comp.1}\ \frac{}{([\omega:0,\_],N,D)\xrightarrow{\bar{\omega}}(\checkmark,N,D')}\ D\overset{\omega}{\Longrightarrow}D'$$

$$\textbf{Comp.2}\ \frac{N\xrightarrow{\nu}_{\mathrm{N}}N'}{([\omega:t_1,t_2],N,D)\xrightarrow{\nu}([\omega:t_1,t_2],N',D)}$$

$$\textbf{Comp.3}\ \frac{N\xrightarrow{(t)}_{\mathrm{N}}N'}{([\omega:t_1,t_2],N,D)\xrightarrow{(t)}([\omega:t_1-t,t_2-t],N',D)}\ t\leq t_2$$

Figure 5. Basic Processes: Communication and Computation

$$\textbf{Gu.1}\ \frac{}{(\gamma\ \text{->}\ P,N,D)\xrightarrow{\tau}(P,N,D)}\ D\models\gamma$$

$$\textbf{Gu.2}\ \frac{N\xrightarrow{\lambda_{NT}}_{\mathrm{N}}N'}{(\gamma\ \text{->}\ P,N,D)\xrightarrow{\lambda_{NT}}(\gamma\ \text{->}\ P,N',D)}\ D\not\models\gamma$$

$$\textbf{Seq}\ \frac{(P,N,D)\xrightarrow{\lambda}(P',N',D')}{(P\ ;\ Q,N,D)\xrightarrow{\lambda}(P'\ ;\ Q,N',D')}$$

$$\textbf{Ch.1}\ \frac{(P,N,D)\xrightarrow{\lambda_P}(P',N',D')}{(P+Q,N,D)\xrightarrow{\lambda_P}(P',N',D')}$$

$$\textbf{Ch.2}\ \frac{(Q,N,D)\xrightarrow{\lambda_P}(Q',N',D')}{(P+Q,N,D)\xrightarrow{\lambda_P}(Q',N',D')}$$

$$\textbf{Int.1}\ \frac{(P,N,D)\xrightarrow{\lambda_P}(P',N',D')}{(P\,[>\,Q,N,D)\xrightarrow{\lambda_P}(P'\,[>\,Q,N',D')}$$

$$\textbf{Int.2}\ \frac{(Q,N,D)\xrightarrow{\lambda_P}(Q',N',D')}{(P\,[>\,Q,N,D)\xrightarrow{\lambda_P}(Q',N',D')}$$

$$\textbf{Par.1}\ \frac{(P,N,D)\xrightarrow{\lambda_P}(P',N',D')}{(P\mid Q,N,D)\xrightarrow{\lambda_P}(P'\mid Q,N',D')}$$

$$\textbf{Par.2}\ \frac{(Q,N,D)\xrightarrow{\lambda_P}(Q',N',D')}{(P\mid Q,N,D)\xrightarrow{\lambda_P}(P\mid Q',N',D')}$$

Figure 6. Compound Processes

$$\frac{(P,N,D)\xrightarrow{\lambda_{NT}}(P',N',D)\quad(Q,N,D)\xrightarrow{\lambda_{NT}}(Q',N',D)}{(P\bullet Q,N,D)\xrightarrow{\lambda_{NT}}(P'\bullet Q',N',D)}$$

Figure 7. Network and Time Transitions – ($\bullet\in\{+,[>,|\}$)

$$\textbf{Rec}\ \frac{(P[\mathrm{rec}\,X.P/X],N,D)\xrightarrow{\lambda}(P',N',D')}{(\mathrm{rec}\,X.P,N,D)\xrightarrow{\lambda}(P',N',D')}$$

$$\textbf{Def}\ \frac{(P,N,D)\xrightarrow{\lambda}(P',N',D')}{(X,N,D)\xrightarrow{\lambda}(P',N',D')}\ X\mathrel{\widehat{=}}P$$

Figure 8. Recursion and Equational Definition

6.2. Semantics

Let K, I, Var, V, Ω and Γ be as described above, with $K' \subseteq K$, $I' \subseteq I$, $Var' \subseteq Var$, $V' \subseteq V$, $\Omega' \subseteq \Omega$ and $\Gamma' \subseteq \Gamma$. The set of *bCANDLE* control systems, $Sys_{\mathbf{bCAN}}$ over K, I, Var, V, Ω and Γ, is the set of triples (P, N, D), where P is a process term over $K', I', Var', \Omega', \Gamma'$; N is a network over K, I, V'; and D is a data environment over Var, V, Ω, Γ.

The semantics of a control system, (P, N, D), is given as the set of timed computations of the timed transition system $(\Sigma, \sigma_{\mathcal{I}}, \mathcal{L}, \longrightarrow)$ where

- $\Sigma \subseteq Sys_{\mathbf{bCAN}}$, is the set of states of the system.
- The initial state, $\sigma_{\mathcal{I}}$, is (P, N, D) where it is required that for all $(M, \preceq, \delta, q, s)_k \in N$, $q = \langle\rangle$ and $s = \downarrow$.
- $\mathcal{L} = \Omega \cup \{\tau, \nu\} \cup \{(t) \mid t \in \mathbb{R}\}$ is the set of transition labels where τ denotes an internal transition initiated by a task and ν an internal transition initiated by the network.
- $\longrightarrow \subseteq (\Sigma \times \mathcal{L} \times \Sigma)$ is the least relation which is closed under the structured operational rules of figures 5,6,7 and 8.

The operational rules make use of generic labels λ_P, λ_{NT} and λ which range over the sets $\{\tau\} \cup \Omega$, $\{\nu\} \cup \{(t) \mid$

Term	Substitute
$\checkmark$	idle
$\checkmark$; P	P
$\checkmark \mid P$ or $P \mid \checkmark$	P
$\checkmark \lceil > P$	$\checkmark$

Figure 9. Termination substitutions

$t \in \mathbb{R}\}$ and $\mathcal{L}$, respectively.; t ranges over $\mathbb{R}$ and t_1, t_2 range over $\mathbb{R}_\infty$. We make use of a distinguished process name, $\checkmark$, which indicates termination. It is used only in giving the semantics and is not available to a user of the language. In applying the operational rules, $\checkmark$ is eliminated from a derived system description by rewriting the process term in which it occurs using the substitutions given in figure 9; the substitution rules are applied until all occurrences of $\checkmark$ are eliminated from the derived term. For example, assume $D \stackrel{\omega}{\Longrightarrow} D'$,then $([\omega : 0, 3]\,;\ \mathrm{idle}, N, D) \stackrel{\bar{\omega}}{\longrightarrow} (\checkmark\ ;\ \mathrm{idle}, N, D')$(by **Seq** and **Comp.1**), which, by the substitutions of figure 9 becomes (idle, N, D'), as expected.

7. Example: Manufacturing Production Cell

We illustrate the use of *bCANDLE* for the construction of a timed transition model of a CAN-based system, using the example of a simple manufacturing production cell [4]. Figure 10 is a graphical representation of the system. A production process, outside the cell, continually deposits items at position 1 of the producer conveyor belt. The producer belt controller drives the belt to move items from position 1 to position 2. The robot controller captures each item at position 2, rotates and processes the item. After processing, the robot rotates again and attempts to deposit the item on the consumer conveyor belt at position 3. The consumer belt controller operates the belt to move items from position 3 to position 4 where they are removed by some external consumption process. We assume an implementation of the manufacturing cell which uses three distributed tasks executing in parallel and communicating via a CAN (see figure 11). The tasks interact with the environment using a variety of sensors and actuators. Environmental interaction is modelled and implemented by simple data operations (`CheckPosition2, BeltOn, DepositItem` etc.) which are analysed independently to obtain bounds on performance. The notation `[CheckPosition2]` etc. is used throughout to abbreviate `[CheckPosition2:t1,t2]` where `t1` and `t2` are the lower and upper bounds, respectively, on the execution time of the code which implements `CheckPosition2`. Other computations (e.g. `[pre_timer]`, `[jump]` etc.)

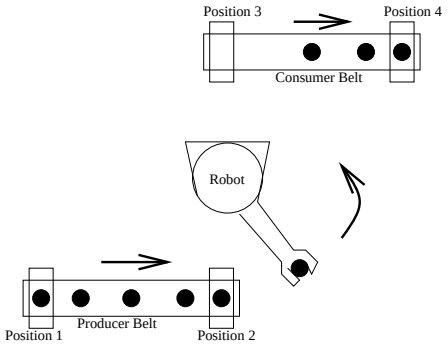

Figure 10. Simple Manufacturing Cell

leave the data environment unchanged and are assumed to take some positive amount of time to execute. Each task maintains a number of boolean variables (e.g. `p1, belton`) to model the state of the external environment; these variables are updated by the execution of one of the data operations (e.g. `CheckPosition1`, `BeltOff`). The variable `__exit_RELEASE` is updated by `[PRE_EXIT_RELEASE]` and `[POST_EXIT_RELEASE]` and is tested by the predicate `exit_RELEASE`. It allows the modelling of a simple loop with an exit condition. Tasks communicate using a single communication channel which carries two types of message, position 2 and position 3 status messages, distinguished by the message identifiers `pos2` and `pos3`. The static attributes of the channel are given in the network section of the system model. Initially, the channel queue is assumed to be empty and the status to be *free*.

The timed transition semantics of *bCANDLE* makes available a wide variety of approaches to analysis of the model. A simulator has been constructed which allows user directed exploration of system behaviour. Translation of a *bCANDLE* model to a timed automaton allows the use of KRONOS [6] or UPPAAL [11] for more detailed investigation.

8. Conclusions

This paper describes work which has been undertaken as part of an ongoing project to provide a tool-supported engineering environment for the development of distributed embedded control systems. We aim to build systems in such a way that it is possible to extract abstract models which preserve important features of the quantitative properties of their implementations. *bCANDLE* is an appropriate language for the expression of such models, which are amenable to a variety of well-developed, tool-supported analysis techniques [8, 6, 11]. We believe that CAN will be an important component in many small/medium scale

```
PBelt | Robot | CBelt
where
PBelt =
   [pre_timer] ;
   ([CheckPosition2];
    [pre_snd]; k!pos2.p2; [post_snd];
    [eval_guard1];
    (guard1 -> [BeltOff] + not_guard1 -> [branch]);
    [CheckPosition1];
    [eval_guard2];
    (guard2 -> [BeltOn] + not_guard2 -> [branch]);
    idle) [>
   [approx_PRODPERIOD]; [post_timer]; [jump]; PBelt

Robot =
  [pre_rcv]; k?pos2.p2; [post_rcv];
  [eval_guard3];
  (guard3 ->
    [CaptureProcessRotate];
    (Release [> exit_RELEASE ->[POST_EXIT_RELEASE]);
    [RotateC_180]
  + notguard3 -> [branch]); [jump]; Robot

Release =
  [pre_rcv]; k?pos3.p3; [post_rcv];
  [eval_guard4] ;
  (guard4 -> [Deposit]; [PRE_EXIT_RELEASE]; idle
  + notguard4 -> [branch]);
  [jump]; Release

CBelt =
  [pre_timer];
  ([CheckPosition4]; [eval_guard5];
   (guard5 -> [BeltOff] + notguard5 -> [branch]);
   [CheckPosition3];
   [pre_snd]; k!pos3.p3; [post_snd];
   [eval_guard6];
   (guard6 -> [BeltOn] + not_guard6 -> [branch]);
   idle) [>
  [approx_CONSPERIOD]; [post_timer]; [jump]; CBelt

/*  Abbreviations
guard1 == PBelt.p2 and PBelt.belton
guard2 == PBelt.p1 and not (PBelt.p2 or PBelt.belton)
guard3 == Robot.p2     guard4 == not Robot.p3
guard5 == CBelt.p4 and CBelt.belton
guard6 == CBelt.p3 and not (CBelt.p4 or CBelt.belton)

not_guardn == not guardn
*/

network
  /*          pri  dlb  dub  dlB  duB         */
  k = (pos2:    1,  43,  53,  10,  12;
       pos3:    2,  43,  53,  10,  12)

data
  PBelt.p1 = false; PBelt.p2 = false;
  Robot.p2 = false; Robot.p3 = false;
  CBelt.p3 = false; CBelt.p4 = false;
  PBelt.belton = false; CBelt.belton = false;
  __exit_RELEASE = false
```

Figure 11. *bCANDLE* **model of Manufacturing Production Cell**

embedded control systems, because of its properties of robustness and predictability. Therefore we have adopted an approach which enables a straight forward construction of tractable system models of CAN-based systems. The major obstacle, as always, remains the management of the state explosion problem. We are currently seeking to apply symbolic and partial order techniques to the simplification of our system models.

References

[1] R. Alur, C. Courcoubetis, and D. Dill. Automata for modelling real-time systems. In *Proc. 17th ICALP*, volume 443, pages 322–335, 1990.

[2] G. Berry and G. Gonthier. The ESTEREL synchronous programming language: design, semantics, implementation. *Science of Computer Programming*, 19:87–152, 1992.

[3] S. Bradley, W. D. Henderson, D. Kendall, and A. P. Robson. Designing and implementing correct real-time systems. In H. Langmaack, W.-P. de Roever, and J. Vytopil, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems FTRTFT '94, Lubeck, Lecture Notes in Computer Science 863*, pages 228–246. Springer-Verlag, September 1994.

[4] M. Brockmeyer, F. Jahanian, C. Heitmeyer, and B. Labaw. An approach to monitoring and assertion-checking of real-time specifications. In *Proceedings of the 4th IEEE Workshop in Parallel and Distributed Real-time Systems*, 1996.

[5] J. Corbett. Timing analysis of Ada tasking programs. *IEEE Transactions on Software Engineering*, 22(7):461–483, July 1996.

[6] C. Daws, A. Olivero, and S. Yovine. Verifying ET-LOTOS programs with KRONOS. In *Proc. Int. Conf. on Formal Description Techniques VII (FORTE'94)*, pages 227–242, 1994.

[7] R. Gerber and I. Lee. A layered approach to automating the verification of real-time systems. *IEEE Transactions on Software Engineering*, 18(9):768–784, September 1992.

[8] T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[9] ISO/DIS 11898: Road Vehicles – interchange of digital information – Controller Area Network (CAN) for high speed communication, 1992.

[10] Y. Kesten and A. Pnueli. Timed and hybrid statecharts and their textual representation. In J. Vytopil, editor, *Formal Techniques in Real Time and Fault Tolerant Systems*, volume 571 of *Lecture Notes in Computer Science*, pages 591–620. Springer Verlag, January 1992.

[11] K. Larsen, P. Pettersson, and Wang Yi. UPPAAL in a Nutshell. *Springer International Journal on Software Tools for Technology Transfer*, October 1997.

[12] K. Prasad. A calculus of broadcasting systems. *Science of Computer Programming*, 25, 1995.