CANDLE: A Tool for Efficient Analysis of CAN Control Systems

David Kendall

Department of Computer Science, University of Durham, Science Laboratories, South Road, Durham, DH1 3LE, UK
david.kendall@durham.ac.uk

Abstract. Construction and analysis of formal system models is increasingly accepted as a valuable technique in the rigorous development of real-time control systems. The effectiveness of modelling with timed automata and analysis via model-checking has been demonstrated often in practice. An obstacle to greater industrial uptake of this approach is the low-level character of the language of timed automata which can make the burden of model construction prohibitively onerous. *CANDLE* is a high-level language and development environment for CAN control systems. It is designed to integrate the production of a CAN system implementation and its formal model. This paper describes the core of an OPEN/CÆSAR-compliant compiler for *CANDLE* which provides an interface to a reachability analyser and established tools such as CADP and OPEN/KRONOS.

1 Introduction

This paper describes the core of an OPEN/CÆSAR-compliant compiler [9] for *CANDLE* [13] which provides an interface to a reachability analyser and established tools such as CADP [7] and OPEN/KRONOS [16]. *CANDLE* is a high-level programming language and development environment for control systems implemented using the Controller Area Network (CAN) communication protocol [11]. It is designed to integrate the production of a CAN system implementation and its formal model. CAN provides multi-master, priority-based bus access based on a CSMA/CR protocol whose deterministic collision resolution policy makes it suitable for use in hard real-time systems. Use of CAN is widespread in the automotive industry and in other sectors where the requirement for high assurance systems provides a strong incentive for the application of formal development methods. The work described in this paper is part of a project which is intended to promote the use of modelling and analysis techniques based on timed automata (TA) [2] in the development of CAN systems. An understanding of timed automata and standard techniques for their analysis is assumed. The reader requiring an introduction is referred to [18].

A *CANDLE* program consists of a high-level description of the behaviour of system processes and the structure of the network via which they communicate. It is not possible to provide details of *CANDLE* here; a detailed presentation appears in [12]. This paper introduces the essential features of the low-level model which is at the core of the compiler. The low-level model comprises three components: a data model, a network model and a control model. These are introduced in the following sections. There follows a discussion of the principles of the compilation techniques and the results of using the compiler on a number of examples. The paper concludes with a summary of the lessons learned so far.

2 The data model

The data model of a *CANDLE* program must be derived from its expression in some suitable external data language. This section does not discuss the details of a particular derivation but introduces only the essential features of a data model.

Let V be a set of data values and Var a finite set of data variables where each variable $x \in Var$ takes its value from some non-empty, finite set of values $\mathsf{type}(x) \subseteq V$. We assume that V contains at least the distinguished value $\perp$, where $\perp \notin \bigcup_{x \in Var} \mathsf{type}(x)$, which is taken to be the "undefined" data value. In modelling the behaviour of a system, the current valuation of the data variables is given by a total function from variables to values. The set *Valuation* of valuations is defined by $Valuation \mathrel{\widehat{=}} Var \rightarrow V$, where for any $\mathsf{val} \in Valuation$ and $x \in Var$ either $\mathsf{val}(x) \in \mathsf{type}(x)$ or $\mathsf{val}(x) = \perp$. During system execution, a valuation of the data variables may be *modified* by executing a data operation or *tested* by a data predicate. The finite sets Ω of operation names and Γ of predicate names are introduced to model data operations and predicates, respectively. A *data environment* D is a tuple $(\mathsf{type}, \mathsf{operation}, \mathsf{predicate}, \mathsf{val})$ where

- $\mathsf{type} : Var \rightarrow 2^V$ is a total function, giving for each variable $x \in Var$, a non-empty, finite set of data values $\mathsf{type}(x)$ ranged over by x;
- $\mathsf{operation} : \Omega \rightarrow 2^{Valuation \times Valuation}$ is a total function giving for each operation name $\omega \in \Omega$ a relation $\mathsf{operation}(\omega)$ which interprets it;
- $\mathsf{predicate} : \Gamma \rightarrow 2^{Valuation}$ is a total function, giving for each predicate name $\gamma \in \Gamma$, a set of valuations $\mathsf{predicate}(\gamma)$ which interprets it;
- $\mathsf{val} : Var \rightarrow V$ is a total function which, for each variable $x \in Var$, gives the current valuation of x, where $\mathsf{val}(x) \in \mathsf{type}(x)$ or $\mathsf{val}(x) = \perp$.

It is assumed that type, $\mathsf{operation}$ and $\mathsf{predicate}$ are fixed for the lifetime of a system but that val may change as the system evolves.

Notation. It is convenient to establish some notational conventions. Let $D = (\mathsf{type}, \mathsf{operation}, \mathsf{predicate}, \mathsf{val})$ be a data environment. Let $x, y \in Var$ be data variables, and $v \in V$ a data value.

- $D.\mathsf{type}$, $D.\mathsf{operation}$, $D.\mathsf{predicate}$ and $D.\mathsf{val}$ denote type, $\mathsf{operation}$, $\mathsf{predicate}$ and val, respectively.
- $D.x$ denotes the value $\mathsf{val}(x)$.
- $D[x := v]$ denotes the data environment $D' = (\mathsf{type}, \mathsf{operation}, \mathsf{predicate}, \mathsf{val}')$ where $\mathsf{val}'(x) = v$ and $\mathsf{val}'(y) = \mathsf{val}(y)$ if y is not syntactically identical to x.

- $D \xrightarrow{\omega}_{\mathsf{d}} D'$ abbreviates the condition

$$(\mathsf{val}, \mathsf{val}') \in \mathsf{operation}(\omega) \wedge D' = (\mathsf{type}, \mathsf{operation}, \mathsf{predicate}, \mathsf{val}')$$

- $D \models \gamma$ abbreviates the condition $\mathsf{val} \in \mathsf{predicate}(\gamma)$.

This framework is general enough to allow the derivation of data models from a variety of practical data languages. For example, a similar approach has been applied in using VDM as an external data language with the process language AORTA [5].

3 The network model

3.1 Structure

A network model is an abstraction of a CAN network [11]. It consists of one or more broadcast channels, each implementing an abstraction of the CAN protocol in which it is assumed that any node is capable of transmitting messages atomically, without errors or failures, to one or more receiving nodes. Messages comprise a message identifier and a data value. The identifier serves both to identify the type of data contained in the message and also to give a priority to the message for use in the arbitration of transmission collisions. Let I be a finite set of message identifiers and V a set of data values. A set of *messages* is any finite subset $M \subseteq I \times V$. A *priority ordering* $\prec$ is a strict, total ordering on message identifiers, where $i \prec i'$ is interpreted to mean that identifier i is of higher priority than identifier i'. In the event of a transmission collision, the message with the highest priority identifier continues transmission to completion while other messages are required to wait. During the transmission of a message on a channel, all nodes connected to the channel act as receivers of the message. At some point in the transmission, each node performs a test based on the message identifier to determine whether it wishes to *accept* the message or ignore it. This *acceptance test* has the effect of creating three phases of message transmission: a *pre-acceptance* phase which extends from the start of transmission to the moment of the acceptance test, an instantaneous *acceptance* phase which is just the moment when the acceptance test is initiated and a *post-acceptance* phase which extends from the acceptance phase to the instant at which the transmission is completed and the channel becomes idle. The *transmission latency* of a message is the time which passes during the pre-acceptance and post-acceptance phases of message transmission. Bounds on transmission latency are given by a function $\delta : M \to \mathbb{N}_\infty \times \mathbb{N}_\infty \times \mathbb{N}_\infty \times \mathbb{N}_\infty$, where $\delta(m) = (l, u, l', u')$ implies that $l \leq u$, $l' \leq u'$ and that the lower and upper bounds on the pre- (resp. post-) acceptance phase of the transmission of m are given by l and u (resp. l' and u'). The derived functions $\delta^{\mathsf{lb}}, \delta^{\mathsf{ub}}, \delta^{\mathsf{lB}}, \delta^{\mathsf{uB}} : M \to \mathbb{N}_\infty$ satisfy

$$\begin{aligned}\forall\, m \in M \,.\, & \delta^{\mathsf{lb}}(m) = l \wedge \delta^{\mathsf{ub}}(m) = u \wedge \delta^{\mathsf{lB}}(m) = l' \wedge \delta^{\mathsf{uB}}(m) = u' \\ & \Leftrightarrow \delta(m) = (l, u, l', u')\end{aligned}$$

Notation. The notation lb (resp. $\mathsf{ub}, \mathsf{lB}, \mathsf{uB}$) will be used as an abbreviation for $\delta^{\mathsf{lb}}(m)$ (resp. $\delta^{\mathsf{ub}}(m), \delta^{\mathsf{lB}}(m), \delta^{\mathsf{uB}}(m)$) when m is clear from the context.

The *transmission status* of a channel identifies whether the channel is free or is transmitting a message and, if transmitting a message, whether it is in the pre-acceptance, acceptance or post-acceptance phase. If a channel is in its pre-acceptance or post-acceptance phase, the bounds on the time to completion of the phase are deemed to be part of its transmission status, since they determine the time at which the channel may next influence the behaviour of a system. The notation shown in Table 1 is used to denote transmission status.

Notation	ASCII	Transmission Status
$\downarrow$	`\/`	FREE, no message is in transmission
$\overset{\mathsf{lb},\mathsf{ub}}{\rightsquigarrow} m$	`--lb,ub->m`	pre-acceptance phase of transmission of message m with bounds $\delta^{\mathsf{lb}}(m), \delta^{\mathsf{ub}}(m)$ on time to completion
$\uparrow m$	`/\m`	acceptance point in transmission of m
$m \overset{\mathsf{lB},\mathsf{uB}}{\rightsquigarrow}$	`m--lB,uB->`	post-acceptance phase of transmission of message m with bounds $\delta^{\mathsf{lB}}(m), \delta^{\mathsf{uB}}(m)$ on time to completion

Table 1. Transmission Status Notation

If it is attempted to transmit a message on a channel which is not free, the message must be stored and offered for transmission again some time after the current transmission has finished. Since messages succeed in their transmission attempts according to their priority, the storing of messages is modelled naturally as a priority ordered queue. If it is necessary to enqueue a message m, whose identifier is the same as that of some other message m' which is already in the message queue, then m replaces m' in the queue and m' is lost forever, i.e. m' is 'overwritten' by m. This represents the behaviour of most implementations of the CAN protocol and has the useful side-effect of ensuring that message queues remain of finite length.

Notation. An empty queue is denoted $\langle\rangle$. A queue with highest priority message m and remaining messages u is written $m{:}u$. The priority-ordered insertion of the message m into the queue u is denoted $u \hookleftarrow m$.

It is now possible to define a *channel* as a tuple $(M, \prec, \delta, s, u)$ where M is the set of messages which can be transmitted by the channel, $\prec$ is the priority ordering of the message identifiers, δ gives the transmission latencies of the messages in M, s is the transmission status and u is the queue of messages currently awaiting transmission. A *network* is collection of channels where each channel is uniquely identified by an identifier taken from some finite set K of *channel identifiers*.

Notation. Let N be a network and $k \in K$ a channel identifier. Then N_k denotes the channel associated with the identifier k in the network N.

3.2 Behaviour

Each channel in a network can act independently by making a discrete change in its transmission status or its message queue. Alternatively, the state of the whole network may be such that it allows time to pass. Since the intention here is to generate a TA which models network behaviour, it is convenient to describe possible discrete changes in network state by giving the inference rules which generate appropriate edges in the TA, and to state time progress conditions as location invariants. With this in mind, a distinct clock variable is associated with each network channel in order to allow the expression of its clock guards and invariants. Let $\mathcal{H}$ be the set of clock variables and h range over $\mathcal{H}$. Let h_u be a distinct clock variable which is used in the statement of *urgency* conditions. Then, the rules for the network edges are given by Figure 1.

$$\textbf{E\_N.1}\quad \frac{N_k = (\downharpoonleft, m : u)^h}{N \xrightarrow[\mathsf{n}]{\mathtt{tt}, k \rightsquigarrow m, \{h_u, h\}} N[k := (\overset{\mathsf{lb},\mathsf{ub}}{\rightsquigarrow} m, u)^h]}$$

$$\textbf{E\_N.2}\quad \frac{N_k = (\overset{t_1,t_2}{\rightsquigarrow} m, u)^h \wedge t_1 \in \mathbb{N}}{N \xrightarrow[\mathsf{n}]{h \geq t_1, k \upharpoonleft m, \{h_u\}} N[k := (\upharpoonleft m, u)^h]}$$

$$\textbf{E\_N.3}\quad \frac{N_k = (\upharpoonleft m, u)^h}{N \xrightarrow[\mathsf{n}]{\mathtt{tt}, m \rightsquigarrow k, \{h_u, h\}} N[k := (m \overset{\mathsf{lB},\mathsf{uB}}{\rightsquigarrow}, u)^h]}$$

$$\textbf{E\_N.4}\quad \frac{N_k = (m \overset{t_1,t_2}{\rightsquigarrow}, u)^h \wedge t_1 \in \mathbb{N}}{N \xrightarrow[\mathsf{n}]{h \geq t_1, k \downharpoonleft, \{h_u\}} N[k := (\downharpoonleft, u)^h]}$$

Fig. 1. Rules for network edges

Notice that the network rules mention only the dynamic components of each network channel, i.e., a channel $(M, \prec, \delta, s, u)$ appears as (s, u); the static components ($M, \prec$ and δ) are unchanging throughout and are assumed when required. The clock variable assigned to a channel is shown as a superscript, as in $(\downharpoonleft, m : u)^h$. The notation $N[k := \eta]$ is used to refer to a network in which channel identifier k is mapped to channel η and all other channels are the same as in network N.

Consider the rule **E_N.1** in Figure 1. It is used to generate an edge for a network N in which some channel k is free and has a highest priority message m in its message queue. In this case, there is an edge to a location in which k is in the pre-acceptance phase of the transmission of m, with bounds $\delta^{\mathsf{lb}}(m), \delta^{\mathsf{ub}}(m)$ on the time to completion of the phase, and m is no longer in the message queue, i.e. the location $N[k := (\overset{\mathsf{lb},\mathsf{ub}}{\rightsquigarrow} m, u)^h]$. The guard of the edge is $\mathtt{tt}$, since a transition via this edge can occur immediately. The label of the edge is $k \rightsquigarrow m$, which identifies a transition via this edge as a change in the status of k to the pre-acceptance phase of transmission of m. The reset set of the edge is $\{h_u, h\}$: the urgent clock h_u is reset by every edge; the channel clock h is reset so that it can be used to measure the pre-acceptance delay in the target location. The rules **E_N.2**, **E_N.3** and **E_N.4** are interpreted similarly.

The network invariant function is given by Figure 2. Consider the definition

$$
\begin{aligned}
I(N) &\mathrel{\widehat{=}} \bigwedge_{k\in K} I(N_k) \\
I(\Downarrow, \langle\rangle)^h &\mathrel{\widehat{=}} \mathsf{tt} \\
I(\Downarrow, m : u)^h &\mathrel{\widehat{=}} h_u \le 0 \\
I(\overset{t_1,t_2}{\rightsquigarrow} m, u)^h &\mathrel{\widehat{=}} \mathsf{if}\ t_2 \in \mathbb{N}\ \mathsf{then}\ h \le t_2\ \mathsf{else}\ \mathsf{tt} \\
I(\uparrow m, u)^h &\mathrel{\widehat{=}} h_u \le 0 \\
I(m \overset{t_1,t_2}{\rightsquigarrow}, u)^h &\mathrel{\widehat{=}} \mathsf{if}\ t_2 \in \mathbb{N}\ \mathsf{then}\ h \le t_2\ \mathsf{else}\ \mathsf{tt}
\end{aligned}
$$

Fig. 2. Invariant function: network

$I(\Downarrow, m : u)^h \mathrel{\widehat{=}} h_u \le 0$. It is used to assert the urgency of a location containing a free channel with a non-empty message queue – transmission of the highest priority queued message must begin immediately. The other invariant conditions are interpreted similarly.

4 The basic process model

There are four kinds of basic process in *CANDLE*:

$k!i.x$ — *non-blocking send*: immediately puts the message $i.v$ into the message queue for channel k, where v is the current value of x, then terminates;

$k?i.x$ — *blocking receive*: idles until k is in the acceptance phase of transmission of a message $i.v$, then immediately assigns v to x and terminates;

$[\omega : t_1, t_2]^h$ — *time-bounded computation*: terminates not earlier than t_1, and not later than t_2, time units after beginning execution – as measured by the clock variable h – atomically transforming the data state at the instant of termination as specified by the operation ω;

$\langle\gamma\rangle$ — *data guard*: idles until the state of the data variables satisfies the predicate γ, then terminates immediately.

The distinguished process $\checkmark$ is used just in giving the semantics of basic processes. It represents the terminated process which is incapable of any discrete action and even blocks the progress of time. As with network behaviour, it is convenient to define basic process behaviour formally by giving rules for the generation of TA edges and location invariants (see Figures 3 and 4).

$$\textbf{E\_Snd}\ \frac{N_k = (s, u) \wedge v = D.x}{(k!i.x, N, D) \xrightarrow{\mathtt{tt}, k!i.v, \{h_u\}} (\checkmark, N[k := (s, u \looparrowleft i.v)], D)}$$

$$\textbf{E\_Rcv}\ \frac{N_k = (\uparrow i.v, \_)}{(k?i.x, N, D) \xrightarrow{\mathtt{tt}, k?i.v, \{h_u\}} (\checkmark, N, D[x := v])}$$

$$\textbf{E\_Comp}\ \frac{D \xrightarrow{\omega}_{\mathsf{d}} D' \wedge t_1 \in \mathbb{N}}{([\omega : t_1, t_2]^h, N, D) \xrightarrow{h \geq t_1, \omega, \{h_u\}} (\checkmark, N, D')}$$

$$\textbf{E\_Gu}\ \frac{D \models \gamma}{(\langle\gamma\rangle, N, D) \xrightarrow{\mathtt{tt}, \gamma, \{h_u\}} (\checkmark, N, D)}$$

Fig. 3. Rules for basic process edges

$$\begin{aligned}
I(k!i.x, D) &\mathrel{\widehat{=}} h_u \leq 0 \\
I(k?i.x, D) &\mathrel{\widehat{=}} \mathtt{tt} \\
I([\omega : t_1, t_2]^h, D) &\mathrel{\widehat{=}} \textsf{if } t_2 \in \mathbb{N} \textsf{ then } h \leq t_2 \textsf{ else } \mathtt{tt} \\
I(\langle\gamma\rangle, D) &\mathrel{\widehat{=}} \textsf{if } D \models \gamma \textsf{ then } h_u \leq 0 \textsf{ else } \mathtt{tt}
\end{aligned}$$

Fig. 4. Invariant function: basic processes

5 The control model

The control of a system model is represented in the *CANDLE* compiler as a kind of Petri net [14]. The net is derived automatically by translation from the *CANDLE* program (see [12] for details of this translation). This approach has been adopted also by Garavel [8] and Yovine [17] in their compilers for LOTOS and ATP, respectively. It has the advantage of giving a compact representation of the control aspects of the system.

The nets which are used in *CANDLE* are similar to the extended nets of [17]. As usual, a net consists of a set of *places* and a set of *transitions*; the convention used here is to denote a set of places by $\mathcal{W}$, W, W', W_1 etc. and a set of transitions by $\Theta, \Theta', \Theta_1$ etc. Two main extensions are introduced whereby each transition is associated with

1. an *attribute* which is used in determining whether the transition is *fireable* in a given system *context* and what are the effects of its firing on its context;
2. a set of places which are said to be *vulnerable* to the firing of the transition. In firing a transition, a token is removed not only from the places in its source set but also from all those places in its vulnerable set.

These extensions are designed to assist in the construction of a compact net representation for *CANDLE* processes. The usual conventions are extended in drawing the diagram of a net: places and transitions are drawn as circles and boxes, respectively; the flow relation is shown using directed edges; additionally, the attribute of a transition is written inside its box and its vulnerable set is written just below.

Figure 5 shows an example net for a simple system for regulating the flow of liquid through a pipe. One process periodically samples a flow sensor and broadcasts its value in a *flow* message. It is assumed that the implementation requires between 85 and 90 μsecs to sample the flow sensor, condition the signal and configure a CAN controller to transmit the *flow* message. A hardware timer, which implements the periodic behaviour of the process, interrupts at intervals of approximately 10 msecs. A second process repeatedly waits to receive a *flow* message, tests the received value to determine whether the flow is *low*, *normal* or *high* and instructs an actuator to adjust a valve accordingly. It takes between 200 to 300 μsecs from receipt of a *flow* message to the configuration of the valve actuator.

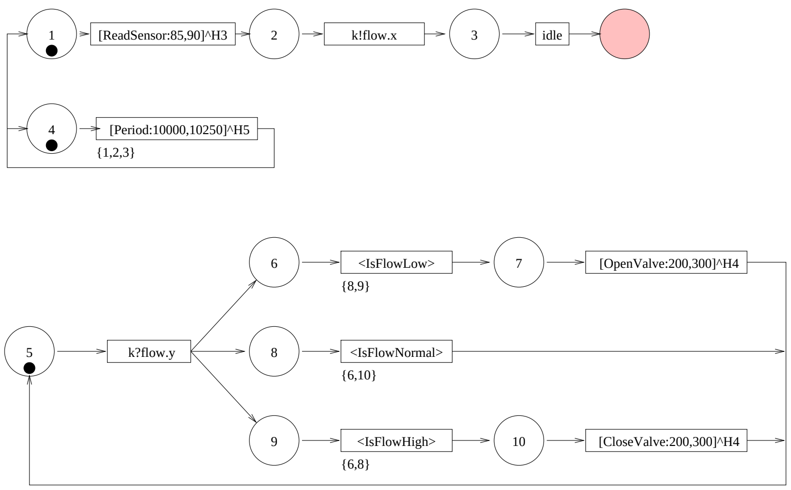

Fig. 5. Flow regulator net

5.1 Definitions and Notation

Transition attributes are just basic processes, as introduced earlier. The set *Attribute* of transition attributes is defined by the grammar

$$\alpha ::= k!i.x \mid k?i.x \mid [\omega : t_1, t_2]^h \mid \langle\gamma\rangle$$

where k, i, x, ω, t_1, t_2, h and γ range over the usual sets. The set of clocks associated with the transition attribute α is denoted $\mathsf{clk}(\alpha)$, where $\mathsf{clk}([\omega : t_1, t_2]^h) \mathrel{\widehat{=}} \{h\}$ and $\mathsf{clk}(\alpha) \mathrel{\widehat{=}} \emptyset$ for $\alpha \in \{k!i.x,\ k?i.x,\ \langle\gamma\rangle\}$. A net $\mathcal{R}$ is a tuple $(\mathcal{W}, \Theta, W^I)$ where

- $\mathcal{W}$ is a finite set of *places*. It is assumed that $\mathcal{W}$ contains a distinguished place ✓ (drawn as a shaded circle) which is used in the representation of the terminal process ✓.
- $\Theta \subseteq \mathcal{W} \times 2^{\mathcal{W}} \times \mathit{Attribute} \times 2^{\mathcal{W}}$ is a set of *transitions*. Let $\theta = (w, W^V, \alpha, W^T) \in \Theta$. The place w is the *trigger* of θ; W^V is the set of places *vulnerable to* θ; α is the *attribute* of θ, denoted $\alpha\theta$; and W^T is the *target* set of θ.
- $W^I \subseteq \mathcal{W}$ is the set of *initial* places.

In the nets which are used here, every place (except ✓) is the trigger of exactly one transition, where the transition triggered by the place w is denoted θ_w. A *marking* is a set $W \subseteq \mathcal{W}$ of places. The marking W^I is the *initial marking*. The set of clocks associated with a marking W is denoted $\mathsf{clk}(W)$, where $\mathsf{clk}(W) \mathrel{\widehat{=}} \bigcup_{w \in W} \mathsf{clk}(\alpha\theta_w)$.

5.2 Behaviour

The behaviour of a net $\mathcal{R}$ is given with respect to a system context comprising a network and a data environment. As before, behaviour is expressed by giving the rules which generate the edges and location invariants in the TA derived from the system description. A location in the derived TA is a tuple (W, N, D) where $W \subseteq \mathcal{W}$ is a marking of $\mathcal{R}$, N is a network and D is a data environment. Intuitively, a system can evolve from one location (W, N, D) to another location (W', N', D') as the result of a transition via a process edge or a network edge.

For a process transition, assume $w \in W$ and that w is the trigger of some transition θ. If the context N, D satisfies the conditions required by the attribute $\alpha\theta$, then a new marking W' is created from W by removing w and any places which are vulnerable to θ, and then including all of the target places of θ. The new context, N', D' is created according to the rule for the attribute $\alpha\theta$, as given in Figure 3.

In the case of a network transition, the system may evolve to a new location in which the network component is modified according to one of the rules in Figure 1. The marking and data environment remain unchanged by the network transition. In order to ensure that the intended broadcast semantics are preserved, the following restriction is imposed:

- a message offer cannot be removed by a network transition if some process is ready to accept the message, i.e., a network transition to the post-acceptance phase of transmission of a message with identifier i on a channel k is inhibited if the current marking contains a place which is the trigger of a transition whose attribute is $k?i.x$.

Now, the process edges of (W, N, D) are given by the rule:

$$\textbf{R.1}\ \frac{\begin{array}{c} w \in W \wedge (w, W^V, \alpha, W^T) \in \Theta \wedge (\alpha, N, D) \xrightarrow{\zeta,\lambda,\mathsf{H}'} (\checkmark, N', D') \wedge \\ W' = W \setminus (\{w\} \cup W^V) \cup W^T \wedge \mathsf{H} = \mathsf{H}' \cup \mathsf{clk}(W^T) \end{array}}{(W, N, D) \xrightarrow{\zeta,\lambda,\mathsf{H}}_{\mathcal{R}} (W', N', D')}$$

and the network edges by the rule:

$$\textbf{R.2}\ \frac{\begin{array}{c} N \xrightarrow{\zeta,\lambda,\mathsf{H}}_{\mathsf{n}} N' \wedge \\ \forall\, k \in K, i \in I \,.\, (\neg\, \mathsf{awaited}(W, k, i) \vee N_k \neq (\uparrow i.\_, \_) \vee N_k = N'_k) \end{array}}{(W, N, D) \xrightarrow{\zeta,\lambda,\mathsf{H}}_{\mathcal{R}} (W, N', D)}$$

where

- the relations $\longrightarrow_{\mathsf{n}}$ and $\longrightarrow$ are as defined in Figures 1 and 3, respectively.
- $\mathsf{awaited}(W, k, i)$ holds iff, in the marking W, it is possible to receive from channel k a message with identifier i. Formally,
$$\mathsf{awaited}(W, k, i) \mathrel{\widehat{=}} \{w \in W \mid \alpha\theta_w = k?i.\_\} \neq \emptyset$$

The invariant of a location (W, N, D) is denoted $I(W, N, D)$ which is defined by

$$I(W, N, D) \mathrel{\widehat{=}} I(W, D) \wedge I(N)$$
$$I(W, D) \mathrel{\widehat{=}} \bigwedge_{w \in W} I(\alpha\theta_w, D)$$

where $I(N)$ and $I(\alpha, D)$ are as in Figures 2 and 4, respectively.

6 Principles of analysis

Given a system model as a net $\mathcal{R} = (\mathcal{W}, \Theta, W^I)$, a network N and a data environment D, the relation $\longrightarrow_{\mathcal{R}}$ and the invariant function I implicitly define a TA which represents the intended system behaviour. The TA can be made explicit quite simply: start from the initial location $q^{\mathcal{I}} \mathrel{\widehat{=}} (W^I, N, D)$ and visit all locations which are reachable under the relation $\longrightarrow_{\mathcal{R}}$, using I to calculate the invariant of each reached location. This approach has been implemented and used to verify simple CAN systems using KRONOS [3] (see [12] for details). However, it suffers from a number of disadvantages, the most serious being that it allows the generation of redundant locations and edges and can produce extremely large TA. Similar approaches to translation from process languages to TA have appeared in the literature and all suffer the same problem [4, 10, 17]. A more interesting technique is to avoid explicit generation of a TA and to proceed directly to the generation of the simulation graph [18] of the implicitly defined TA. A node in the simulation graph is a pair (q, ζ) where $q = (W, N, D)$ is a location and ζ is a clock zone. In calculating graph nodes, the usual operations on clock zones are used: $\nearrow\zeta$, $\zeta[\mathsf{H} := 0]$, $\mathsf{close}_c(\zeta)$ and $\zeta \cap \zeta'$, denoting forward projection, clock reset, c-closure and intersection, respectively. The operation

$\mathsf{suc}^q_\tau(\zeta) \mathrel{\widehat{=}} \nearrow\zeta \cap I(q)$ defines the clock zone which can be reached from ζ as time progresses while control remains at location q. The operation $\mathsf{suc}_e(\zeta) \mathrel{\widehat{=}} (\zeta \cap \zeta')[\mathsf{H} := 0]$ defines the clock zone which can be reached from ζ by following the edge e where the guard and reset set of e are ζ' and H, respectively. Now, the initial graph node is given by $(q^{\mathcal{I}}, \mathsf{suc}^{q^{\mathcal{I}}}_\tau(\mathsf{zero}))$ and there is a successor node (q', ζ') from any node (q, ζ) iff an edge $e = (q, \zeta'', \lambda, \mathsf{H}, q')$ can be exhibited from the inference $q \xrightarrow{\zeta'',\lambda,\mathsf{H}}_{\mathcal{R}} q'$, and $\zeta' = \mathsf{close}_c(\mathsf{suc}^{q'}_\tau(\mathsf{suc}_e(\zeta))) \neq \emptyset$, where c is the largest constant appearing in a computation $[\omega : t_1, t_2]$ of $\mathcal{R}$ or a transmission latency function δ of N.

In order to improve memory usage during 'on-the-fly' generation of the simulation graph, it is very helpful to apply the well-known clock activity abstraction [6]. This requires that it be possible to compute the active clocks of locations 'on-the-fly'. Fortunately, this is very simple for *CANDLE* locations. It is shown in [12] that the only clocks which are *active* in any *CANDLE* location q are just those clocks which are *tested* in q, i.e., which occur in an outgoing edge of q or in its invariant condition. Given a net $\mathcal{R}$ and a location $q = (W, N, D)$, it is trivial to determine the set $\mathsf{tclk}(q)$ of tested clocks of q from the structure of the net and the components of the location (see [12]). In this case, it is simple to apply the clock activity abstraction by computing each successor (q', ζ') of a node (q, ζ) as described above and restricting the representation of ζ' to the clocks in $\mathsf{tclk}(q')$. The inclusion and convex hull abstractions of [6] can also be applied 'on-the-fly', although this remains to be implemented.

An exploration module has been implemented which allows simple reachability analysis of the simulation graph. Further analysis is possible using the CADP tools [7] and a connection to profounder [16] for checking TBA-emptiness should be straightforward.

7 Experiments

In order to test the effectiveness of this approach to analysis, a compiler has been implemented which takes as its input a *CANDLE* program together with a set of C type definitions and functions implementing the program data model. The compiler translates the *CANDLE* program to a net from which it generates a C program which implements the OPEN/CÆSAR graph API [9] as its output. The OPEN/CÆSAR program provides functions for generating the initial simulation graph node and iterating over the simulation graph successors of a node by directly implementing the ideas described above, using both standard and variable-dimension DBMs [16, 18] to represent clock zones. The simulation graphs of a number of examples have been generated, both with and without clock activity reduction, including

flow the system from Figure 5,
sboiler a simplified version of the steam boiler problem [1], and
disbmutex a 3 process version of the centralised algorithm for distributed mutual exclusion from [15].

For the purposes of comparison, an attempt has been made to generate an explicit TA for the same examples. The results are summarised in Table 2.

	TA	NOACT		ACT	
	#locs	#clocks	#states	#clocks	#states
flow	2256	5	84	3	55
sboiler	573683	12	74859	5	2029
disbmutex	–	6	>1065000	6	223604

Table 2. Experimental results

The TA column gives the total number of locations in the generated TA, the NOACT column gives the maximum number of clocks required and the total number of states in the simulation graph generated without clock activity reduction; the ACT column gives the corresponding data when clock activity reduction is applied. It can be seen, as expected, that direct generation of the simulation graph using clock activity reduction performs much better in each case. For the disbmutex program, it was not possible to generate the explicit TA and the generation of the simulation graph without activity reduction was halted after 1,065,000 states.

8 Conclusions

Several lessons have been learned as a result of our experiences so far with the *CANDLE* compiler. The general approach of translation from a high-level language through a number of intermediate stages to C source code implementing the OPEN/CÆSAR API has proved to lead to an effective decomposition of the translation problem, providing opportunities for optimisation at every stage and resulting in programs which can be connected easily with a variety of analysis tools. Our experience is that this approach clearly outperforms methods in which a TA is constructed explicitly. The applicability and utility of clock activity reduction for *CANDLE* programs is apparent. Future work will be aimed at producing a systematic evaluation of further techniques from the literature in terms of their impact on the control of state explosion on a range of CAN control systems.

References

1. J.-R. Abrial, E. Börger, and H. Langmaack, editors. *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*, volume 1165 of *Lecture Notes in Computer Science*. Springer Verlag, 1996.
2. R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–236, 1994. Preliminary version appears in Proceedings of 17th ICALP, 1990, LNCS 443.

3. M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: a model-checking tool for real-time systems. In A. Hu and M. Vardi, editors, *Proceedings of the 10th International Conference on Computer Aided Verification (CAV'98)*, volume 1427 of *Lecture Notes in Computer Science*, pages 546–550. Springer Verlag, 1998.
4. S. Bradley, W. Henderson, D. Kendall, and A. Robson. Validation, verification and implementation of timed protocols using AORTA. In P. Dembinski, editor, *Proceedings of the Fifteenth International Symposium on Protocol Specification, Testing and Verification*, pages 205–220. Chapman and Hall, June 1995.
5. S. Bradley, W. Henderson, D. Kendall, and A. Robson. A formal design language for real-time systems with data. *Science of Computer Programming*, 40(1):3–29, May 2001.
6. C. Daws and S. Tripakis. Model checking of real-time reachability properties using abstractions. In B. Steffen, editor, *Proceedings of 4th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98)*, volume 1384 of *Lecture Notes in Computer Science*, pages 313–329. Springer Verlag, 1998.
7. J. Fernandez, H. Garavel, A. Kerbrat, L. Mounier, R. Mateescu, and M. Sighireanu. CADP a protocol validation and verification toolbox. In *Proceedings of the 8th International Conference on Computer Aided Verification (CAV'96)*, volume 1102 of *Lecture Notes in Computer Science*, pages 437–440. Springer Verlag, 1996.
8. H. Garavel. *Compilation et Vérification de Programmes LOTOS*. PhD thesis, Institut National Polytechnique de Grenoble, July 1992.
9. H. Garavel. OPEN/CÆSAR: An open software architecture for verification, simulation and testing. In B. Steffen, editor, *Proceedings of 4th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98)*, volume 1384 of *Lecture Notes in Computer Science*, pages 68–84. Springer Verlag, March 1998.
10. C. Hernalsteen. *Specification, Validation and Verification of Real-Time Systems in ET-LOTOS*. PhD thesis, Université Libre de Bruxelles, August 1998.
11. ISO/DIS 11898: Road Vehicles – interchange of digital information – Controller Area Network (CAN) for high speed communication, 1992.
12. D. Kendall. *Formal Modelling and Analysis of Broadcasting Embedded Control Systems*. PhD thesis, Department of Computing Science, University of Newcastle upon Tyne, UK, September 2001. (Forthcoming).
13. D. Kendall, S. Bradley, W. Henderson, and A. Robson. *CANDLE*: A high level language and development environment for CAN control systems. In *Proceedings of 4th International Workshop on Discrete Event Systems (WODES'98), Cagliari, Italy*. IEE, 1998.
14. T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.
15. A. Tanenbaum. *Modern Operating Systems*. Prentice Hall International, 1992.
16. S. Tripakis. *The Formal Analysis of Timed Systems in Practice*. PhD thesis, Université Joseph Fourier, Grenoble, December 1998.
17. S. Yovine. *Méthodes et Outils pour la Vérification Symbolique de Systèmes Temporisés*. PhD thesis, Institut National Polytechnique de Grenoble, May 1993.
18. S. Yovine. Model checking timed automata. In G. Rozenberg and F. Vaandrager, editors, *Embedded Systems, Papers from the European Educational Forum School on Embedded Systems, Veldhoven, The Netherlands*, volume 1494 of *Lecture Notes in Computer Science*, pages 114–152. Springer Verlag, 1997.