Formal Specification of Gateways in Integrated Architectures

R. Obermaisser

Vienna University of Technology, Austria

Abstract. Complex embedded computer systems can encompass multiple application subsystems, such as a multimedia, a powertrain, a comfort and a safety subsystem in the in-vehicle electronic system of a typical premium car. Information exchanges between these application subsystems are essential to realize composite services that involve more than one application subsystem and to reduce redundant computations and sensors. A major challenge is to resolve the property mismatches at the interfaces between application subsystems, such as incoherent naming, divergent syntax, or different communication protocols. Also, fault isolation capabilities are required to prevent common mode failures induced by the propagation of faults between application subsystems. The contribution of this paper is a formal specification of gateways that contain structured collections of time-sensitive variables associated with timing information (called real-time databases) in order to separate the application subsystems. The formal specification can serve as a basis for automatic code generation or formal verification.

1 Introduction

Large distributed embedded systems (e. g., complete on-board electronic system of a car) consist of numerous application subsystems, each providing a part of the overall application functionality. Designers follow a divide-and-conquer strategy in order to manage the system's complexity by structuring the overall functionality into nearly-independent subsystems [1, chap. 8]. For example, in-vehicle electronics are usually grouped into several domains, including the safety-related powertrain and chassis domains, as well as the non-safety critical comfort and multimedia domains [2]. Each domain comprises a set of Electronic Control Units (ECUs) interconnected by a network (e. g., Controller Area Network (CAN) [3], FlexRay [4]).

However, the subdivision of the overall system usually does not lead to fully independent application subsystems. Interactions between application subsystems are required for improved quality-of-service, for implementing application services that span more than one application subsystem, and for exploiting redundancy [5].

The requirement of sharing information between application subsystems becomes a challenge, if the overall system encompasses heterogeneous application subsystems, which exchange messages using different communication protocols, incoherent naming, and divergent syntax or semantics. In this case, property mismatches at the interfaces between application subsystems need to be resolved.

In previous work [5], we have introduced a framework for the realization of gateways between application subsystems as part of the Dependable Embedded Components and Systems (DECOS) architecture [6]. The gateways proposed in this framework support selective redirection of information between networks in conjunction with the necessary property transformations. Central to this framework is a *real-time database* [7, p. 289], which is contained in the gateway and stores real-time images for the information exchange between the interconnected networks.

This paper describes the formal specification of these gateways. The formal gateway specification is expressed using state machines with timing constraints and gateway-specific operations (e. g., operations for accessing the real-time database). Existing solutions for the specification of real-time systems, such as timed automata [8], calendar automata [9] and time-triggered automata [10], were considered for developing this formal gateway specification. The formal gateway specification serves as the input for an automatic code generation tool, which yields data structures and code that serve as a parameterization of a generic architectural gateway service. Furthermore, the formal gateway specification is a baseline for the formal verification of systems using the proposed gateways.

The paper is structured as follows. Section 2 explains the gateway framework. The formal specification of the gateways is the focus of Section 3. The gateway specification formally captures the information to control the behavior of the gateway (i. e., selective redirection and property transformations). Section 4 gives an overview of the model-based generation of the gateways using the gateway specification as a starting point. The paper concludes with a discussion in Section 5.

2 Gateways based on a Real-Time Database

A real-time system can be modeled using a set of *real-time entities* [11], which are significant state variables that are located in the computer system or the environment. The current value of such a real-time entity is called a *real-time image* and can be sent within a message on network. Redirection of information through a gateway occurs when a real-time image contained in a message is required by another Distributed Application Subsystem (DAS) connected to the gateway. We denote such a real-time image that is relevant at the gateway as a *convertible element*.

The presented gateways recombine convertible elements acquired from one network into messages for another network, while converting between different temporal and syntactic specifications and resolving naming incoherences. For this purpose, the gateway maintains a real-time database with convertible elements called the *gateway repository*. The gateway repository decouples the different networks accessed by the gateway and allows the convertible elements that are necessary for constructing a particular message to arrive at different points in time.

In addition, the gateway contains for each accessed DAS a so-called *network-adapter*, which implements the communication protocol of the network of the DAS and performs information exchanges between the network and the gateway repository (see Figure 1).

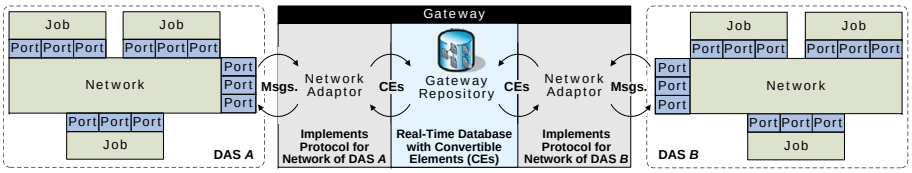

Fig. 1. Gateway

2.1 Network Adaptors

A network adaptor can acquire convertible elements from a network and write them into the gateway repository. Depending on the protocol, the acquisition of a message with convertible elements can involve the exchange of several messages at input and output ports, e. g., the transmission of a request message before a response message carrying the convertible elements arrives. Secondly, a network adaptor can read convertible elements from the gateway repository, construct messages and disseminate them on a network. Thereby, information can be redirected between networks, if the read convertible elements have been placed in the gateway repository by another network adaptor.

The specification of the network adaptors occurs using state machines with timing constraints and will be explained in Section 3.

2.2 Gateway Repository

For the storage of convertible elements, the gateway repository takes into account the information semantics of convertible elements. Due to the respective characteristics of state and event semantics, the gateway repository distinguishes two types of storage elements in analogy to state and event ports. For convertible elements with state semantics, the repository contains state variables that are overwritten whenever a new version of the convertible element arrives (update-in-place). Convertible elements with event semantics, on the other hand, are stored in queues.

In addition to the data of the convertible elements, the gateway repository also stores meta-information about convertible elements. The meta-information maintained in the gateway repository includes three dynamic attributes (most recent update instant, update request indication, number of queued instances) and a static attribute (temporal accuracy offset).

- **Most recent update instant.** The point in time of the most recent update t_{update} is a dynamic attribute associated with each convertible element with state semantics. t_{update} is set to the current time t_{now}, whenever a network adaptor overwrites the convertible element in the gateway repository.
- **Temporal accuracy interval and offset.** Due to the dynamics of real-time entities, the validity of convertible elements is time-dependent. For this reason, the gateway repository maintains for each convertible element with state semantics a dynamic attribute called the *temporal accuracy interval* d_{acc}. At any given instant, d_{acc} denotes how long the convertible element will still remain a valid image of the respective real-time entity in case no update of the convertible elements occurs in the meantime.

The *temporal accuracy offset* d_{offset} is a static attribute that determines the temporal accuracy interval immediately after an update of the convertible element. In conjunction with the instant of the most recent update, the temporal accuracy offset allows to compute the temporal accuracy interval of a convertible element: $d_{\text{acc}} = d_{\text{offset}} - (t_{\text{now}} - t_{\text{update}})$. Hence, only the temporal accuracy offset needs to be stored in the gateway repository, because the temporal accuracy interval can be computed on-the-fly.

- **Update request indication.** In order to support on-demand communication activities, the gateway repository contains boolean *update request indications*. For a convertible element with state or event semantics, the respective update request indication b_{req} denotes whether a new convertible element needs to be transferred into the gateway repository. By setting the update request indication, a network adaptor can demand convertible elements from the other network adaptors. A network adaptor receiving messages from a network can initiate receptions conditionally, based on the value of the update request indication.
- **Number of queued instances.** Every convertible element with event semantics possesses this dynamic attribute. It denotes the number of instances of the convertible element that are currently queued in the gateway repository.

Using the introduced attributes, we can control the behavior of a network adaptor. For example, the meta-information provides the network adaptors with information for the decision whether to actively engage in the acquisition of convertible elements for the update of the gateway repository. A network adaptor can react to the imminent invalidation of temporal accuracy, e. g., by starting a protocol to perform an update of the convertible element in the gateway repository.

3 Formal Specification of Gateways

This section formally defines a gateway with multiple network adaptors and a real-time database. Based on the notion of the gateway state, we also describe the execution of a gateway over time.

3.1 Definition of Network Adaptor

A network adaptor is a state machine with local variables, clock variables, locations, and edges. An edge interconnects two locations of the network adaptor and can be associated with a guard, assignments to variables and communication actions. The guard expresses a boolean condition, which defines whether the edge can be taken. Communication actions are used to express interactions with the gateway repository and the ports. Variables are used to capture the internal state of the network adaptor. In particular, variables can store messages and convertible elements. In a communication action, variables can serve as the source for the transfer of a convertible element into the gateway repository or the transfer of a message to a port for being transmitted. In analogy, variables can serve as the destination when executing a communication action to read a convertible element from the gateway repository or to receive a message from a port.

Formally, a network adaptor is a tuple $\langle L, l, X, R, E \rangle$, where

L is a finite set of symbols denoting locations,
$l \in L$ is the initial location,
X is a finite set of clocks $X = \{X_1, X_2, ...\}$,
R is a set of local variables: $R \subset \{(a,b)|a \in \mathbb{N} \wedge \mathrm{b} \in \mathfrak{P}(\mathbb{N})\}$ Each local variable is associated with a name $a \in \mathbb{N}$. We call the set of all variable names $\eta(R) := \{z|\exists(z,b) \in R\}$. In addition, each variable possesses a corresponding domain described by a subset of $\mathbb{N}$. We use the function $domain(x)$ to determine the domain of a variable with name x: $domain(x) := \mathrm{b}$ where $(a,b) \in R \wedge x = a$
E is a set of edges: $E \subseteq L \times L \times \Phi(\bar{Z}) \times \alpha(\bar{Z}) \times \chi(\bar{Z})$, where
$\bar{Z}$ is the state of the gateway (cf. Section 3.3)
Φ is the guard defined by the function $\bar{Z} \mapsto \{\mathbb{T}, \mathbb{F}\}$
α is the assignment action defined by the function $\bar{Z} \mapsto \bar{R}$, where $\bar{R}$ denotes the local variables state (cf. Section 3.3), i. e., the values of the local variables at a certain point in time
χ is the communication action defined by a function $\chi : \bar{Z} \mapsto \mathfrak{P}(\mathbb{N}) \times \mathfrak{P}(\mathbb{N}) \times \mathfrak{P}(\mathbb{N}) \times \mathfrak{P}(\mathbb{N})$, controlling in each state $\bar{Z}$ the effect of the network adaptor on the port and repository state. $\chi(\bar{Z}) = (C_{out}, C_{in}, M_{in}, M_{out}\}$, where C_{out} is the set of convertible elements, which are pushed into the gateway repository. The set of messages pulled out of the gateway repository is captured by C_{in}. M_{in} is the set of messages, which are received by the gateway, i. e., transferred from a port to variables. M_{out} is the set of messages, which are sent by the gateway, i. e., transferred from variables to a port.

3.2 Definition of a Gateway

A gateway consists of one or more network adaptors, ports with messages, and the gateway repository with convertible elements. Formally, a gateway is a tuple $Z = \langle A, V, M, C, Y \rangle$, where

A is a finite set of network adaptors: $A =< A_0, A_1, ..., A_z >$,
$A_0 =< L_0, l_0, X_0, R_0, E_0 >, \ldots, A_z =< L_z, l_z, X_z, R_z, E_z >$
V is the global set of variables (i.e., local variables of all network adaptors extended with network adaptor names): $V = \{(c,b)|c = (i,a) \wedge (a,b) \in R_i \wedge 0 \leqslant i \leqslant z\}$
M is a finite set of messages: $M \subset \{(u,v,p,n,d,\tau)|u \in \mathbb{N}, v \subset \eta(V), p \in \{\mathrm{ET,TT}\}, n \in \mathbb{N}, d \in \mathbb{N}, \tau : domain(v) \mapsto domain(v)\}$, where each message possesses a message name u, a set of associated variables v, a control paradigm p, a queue length n (only relevant for event information), and a temporal accuracy offset d (only relevant for state information). A transfer syntax can be specified using the function $\tau : domain(v) \mapsto domain(v)$ The associated variables are defined with respect to the set of all variable names $\eta(V) := \{z|\exists(z,b) \in V\}$.
C is a finite set of convertible elements: $C \subset \{ (u,v,p,n,d)|u \in \mathbb{N}, v \subset \eta(V), p \in \{\mathrm{ET,TT}\}, n \in \mathbb{N}, d \in \mathbb{N}\}$, where each convertible elements possesses a name u, a set of associated variables v, a control paradigm p, a queue length n (only relevant for event information), and a temporal accuracy offset d (only relevant for state information).
Y is the global set of clocks (i.e., local clock variables of all network adaptors extended with network adaptor names): $Y = \{((z,a),b)|b \in \mathbb{N}, a \in X_i \wedge 0 \leqslant i \leqslant z\}$

3.3 Gateway State

The capturing of the gateway state serves the definition of the execution semantics of a gateway. The gateway state embodies all past history of the gateway. Thus, at any specific time the future outputs of the gateway depend only on the current state of the gateway and the future inputs (i. e., messages at ports).

Formally, the state of a gateway $\bar{Z} = \langle t, \bar{V}, \bar{M}, \bar{C}, \bar{Y} \rangle$ at time t consists of the global time t ($t \in \mathbb{N}$), the variables state $\bar{V}$, the message state $\bar{M}$, the repository state $\bar{C}$, and the clocks state $\bar{Y}$. These constituting elements of the gateway state will be explained in the following.

State of Variables At a certain time t, the variables state $\bar{V}$ of the gateway encompasses the values of the local variables of all network adaptors:

$$\bar{V} \subset \{(x,y) | \exists (a,b) \in V \text{ with } a = x \wedge y \in b\} \quad \text{where} \underbrace{\underset{x \in \eta(V)}{\forall} \left| (x,y) \in \bar{V} \right| = 1}_{\text{exactly one value for each variable}}$$

The name of each variable is a 2-tuple ($x \in \mathbb{N} \times \mathbb{N}$) identifying the network adaptor and the local variable. The value y of the variable is an element of the variable domain.

In addition, we can also define the local variables state $\bar{R}$ of a single network adaptor, where each variable name is a natural number ($x \in \mathbb{N}$).

$$\bar{R} \subset \{(x,y) | \exists (a,b) \in R \text{ with } a = x \wedge y \in b\} \quad \text{where} \underbrace{\underset{x \in \eta(R)}{\forall} \left| (x,y) \in \bar{R} \right| = 1}_{\text{exactly one value for each variable}}$$

Port State The port state consists of the state of the state ports ($\bar{M}^{(tt)}$) and the state of the event ports ($\bar{M}^{(et)}$). Each element of the port state ($\bar{M}^{(tt)}$ and $\bar{M}^{(et)}$) is a 5-tuple $< x, y, z, l, t >$, where x is a message name, y is a variable name, z is a value, l is the number of queued elements, and t is the global point in time of the last update.

Formally, we can define the state of event and state ports as follows:

$$\bar{M}^{(tt)} \subset \{(x,y,z,1,t) | \exists (u,v,p,n,d,\tau) \in M, \exists (a,b) \in V \text{ with } x = u \wedge y \in v \wedge y = a \wedge z \in b \wedge p = \text{TT}, t \in \mathbb{N}\}$$

$$\bar{M}^{(et)} \subset \{(x,y,z,l,0) | \exists (u,v,p,n,d,\tau) \in M, \exists (a,b) \in V \text{ with } x = u \wedge y \in v \wedge y = a \wedge z \in b^n \wedge p = \text{ET} \wedge l \in \{0,1,...,n\}\}$$

For state ports, the queue length in the 5-tuple is always 1, because state ports provide no message queues. The value z of a constituting variable v of a message u is an element of the variable's domain b. For event ports, l is the actual number of messages and must be smaller or equal to the maximum queue length. The value z is an element of the Cartesian power b^n of the domain b over the maximum queue length n. We model the queue at an event port as a vector of dimension n, where the first vector element represents the most recently enqueued message.

The port state needs to assign exactly one value to each constituting variable of a message. In order to express this property, we demand that the following constraint holds:

$$\underset{(x,y) \in S_{\text{msg}}}{\forall} \left| (x_1, y_1, z_1, l_1, t_1) \in \bar{M}^{(tt)} \text{ with } x_1 = x \wedge y_1 = y \right| + \left| (x_2, y_2, z_2, l_2, t_2) \in \bar{M}^{(et)} \text{ with } x_2 = x \wedge y_2 = y \right| = 1$$

$$S_{\text{msg}} = \{(x,y) | (u,v,p,n,d,\tau) \in M \wedge u = x \wedge v = y\}$$

Repository State The repository state encompasses the state of convertible elements with state information ($\bar{C}^{(tt)}$) and the state of convertible elements with event information ($\bar{C}^{(et)}$). For each convertible element the repository state ($\bar{C}^{(tt)}$ and $\bar{C}^{(et)}$) contains a 6-tuple $< x, y, z, l, t, r >$, where x is a convertible element name, y is a variable name, z is a value, l is the number of queued elements, t is the global point in time of the last update, and r is the number of update request indications.

Formally, we can define the repository state as follows:

$$\bar{C}^{(tt)} \subset \{(x,y,z,1,t,0) | \exists (u,v,p,n,d) \in C, \exists (a,b) \in V \text{ with } x = u \wedge y \in v \wedge y = a \wedge z \in b \wedge p = \text{TT}, t \in \mathbb{N}\}$$

$$\bar{C}^{(et)} \subset \{(x,y,z,l,0,r) | \exists (u,v,p,n,d) \in C, \exists (a,b) \in V \text{ with } x = u \wedge y \in v \wedge y = a \wedge z \in b^n \wedge p = \text{ET} \wedge l \in \{0,1,...,n\}, r \in \mathbb{N}\}$$

For the convertible element with event information, l is the actual number of convertible elements in the queue, b^n is the Cartesian power of the domain b over the maximum queue length n, and r is the number of requested convertible elements.

The repository state needs to assign exactly one value to each combination of convertible element name and variable name. In order to express this property, we demand that the following constraint holds:

$$\underset{(x,y) \in S_{\text{msg}}}{\forall} \left| (x_1, y_1, z_1, l_1, t_1) \in \bar{M}^{(tt)} \text{ with } x_1 = x \wedge y_1 = y \right| + \left| (x_2, y_2, z_2, l_2, t_2) \in \bar{M}^{(et)} \text{ with } x_2 = x \wedge y_2 = y \right| = 1$$

$$S_{\text{msg}} = \{(x,y) | (u,v,p,n,d,\tau) \in M \wedge u = x \wedge v = y\}$$

Clock State At any specific time, the clock state consists of the values of all clock variables.

$$\bar{Y} \subset \{(x,y) | x \in Y, y \in \mathbb{N}\} \quad \text{where} \underbrace{\underset{x \in Y}{\forall} \left| (x,y) \in \bar{Y} \right| = 1}_{\text{exactly one value for each clock}}$$

3.4 Formal Definition of Gateway Execution

In the execution of a gateway, we can distinguish two types of transitions, namely *timed transitions* and *untimed transitions*. In the following these two types of transitions and the sequence of their execution will be explained.

Timed Transitions During the execution of a timed transition, a tick of the global time base elapses. The time progress of one tick is reflected by incrementing the clock variables by a value of one, while the variables state, port state, and repository state remain unchanged. Formally, a *timed transition* $\bar{Z}_i \xrightarrow{T} \bar{Z}_{i+1}$ is defined as follows:

$$\bar{Z}_i = \left\langle t_i, \bar{V}_i, \bar{M}_i^{(tt)}, \bar{M}_i^{(et)}, \bar{C}_i^{(tt)}, \bar{C}_i^{(et)}, \bar{Y}_i \right\rangle,$$

$$\bar{Z}_{i+1} = \left\langle t_{i+1}, \bar{V}_{i+1}, \bar{M}_{i+1}^{(tt)}, \bar{M}_{i+1}^{(et)}, \bar{C}_{i+1}^{(tt)}, \bar{C}_{i+1}^{(et)}, \bar{Y}_{i+1} \right\rangle$$

$$t_{i+1} = t_i + 1, \bar{V}_{i+1} = \bar{V}_i, \bar{M}_{i+1}^{(tt)} = \bar{M}_i^{(tt)}, \bar{M}_{i+1}^{(et)} = \bar{M}_i^{(tt)}, \bar{C}_{i+1}^{(tt)} = \bar{C}_i^{(tt)}, \bar{C}_{i+1}^{(et)} = \bar{C}_i^{(tt)}$$

$$\bar{Y}_i = \{\{x_1, y_1\}, \{x_2, y_2\}, ...\}, \bar{Y}_{i+1} = \{\{x_1, y_1 + 1\}, \{x_2, y_2 + 1\}, ...\}$$

Untimed Transitions The execution of untimed transitions is instantaneous. During an untimed transition $\bar{Z}_i \xrightarrow[A]{I,O} \bar{Z}_{i+1}$ a network adaptor A processes input (i. e., incoming

messages I and outgoing messages O at ports) and executes assignment and communication actions. Like a timed transition, an *untimed transition* $\bar{Z}_i \xrightarrow[A]{I,O} \bar{Z}_{i+1}$ links a source state $\bar{Z}_i$ with a target state $\bar{Z}_{i+1}$:

$$\bar{Z}_i = \left\langle t_i, \bar{V}_i, \bar{M}_i^{(tt)}, \bar{M}_i^{(et)}, \bar{C}_i^{(tt)}, \bar{C}_i^{(et)}, \bar{Y}_i \right\rangle,$$
$$\bar{Z}_{i+1} = \left\langle t_{i+1}, \bar{V}_{i+1}, \bar{M}_{i+1}^{(tt)}, \bar{M}_{i+1}^{(et)}, \bar{C}_{i+1}^{(tt)}, \bar{C}_{i+1}^{(et)}, \bar{Y}_{i+1} \right\rangle$$

As described in Section 2, ports are the interface between the gateway and the networks. Consequently, both the gateway and the networks can cause changes to the port state. The effects on the port state due to networks are captured by the sets I and O:

$$I = \{(u_1, v_1, z_1), (u_2, v_2, z_2), \dots \mid \underset{j=1,2,\dots}{\forall} : \; u_j \in \eta(M) \wedge \exists (a,b) \in V \text{ with } a = v_j \; \wedge z_j \in b\}$$
$$O = \{(u_1, v_1), (u_2, v_2), \dots \mid \underset{j=1,2,\dots}{\forall} : \; u_j \in \eta(M) \wedge v_j \in \eta(V)\}$$
$$\eta(M) = \{u | \exists (u, v, p, n, d, \tau) \in M\}$$

The set I contains tuples each with a message name, a variable name and a value of the respective value domain. The value is used to perform an update-in-place of the message at a state port or to enqueue a message at an event port. The set O contains only message and variable names, which are used to dequeue messages at event ports. $\eta(M)$ is the set of all message names.

The update of the port state is defined below. The new state of a state port is the union of the unmodified ports (i. e., no update by the gateway or a network) and the ports with updated messages (i. e., either through the network or through the communication action of a network adaptor). In case of an update-in-place of the port, the most recent update instant is equal to the current global time t.

$$\bar{M}_{i+1}^{(tt)} = \underbrace{\{(x,y,z,l,t') \in \bar{M}_i^{(tt)} \mid x \notin M_{out} \wedge \nexists (u,v,b) \in I \text{ with } u = x\}}_{\text{neither send operation nor update of port through communication system}} \cup$$
$$\underbrace{\{(x,y,z,l,t'') \mid t'' = t \wedge (x,y,z',l,t') \in \bar{M}_i^{(tt)} \wedge \exists (u,v,b) \in I \text{ with } u = x \wedge y = v \wedge z = b\}}_{\text{communication system delivers msg. to port}} \cup$$
$$\underbrace{\{(x,y,z,l,t'') \mid t'' = t \wedge (x,y,z',l,t') \in \bar{M}_i^{(tt)} \wedge x \in M_{out} \wedge \exists (i,j) \in \bar{V}_{i+1} \wedge i = y \wedge z = j\}}_{\text{value of a variable is copied as part of a send operation of a state message}}$$

In analogy, the new state of an event port is the union of the unmodified ports and the ports with enqueued or dequeued messages.

$$\bar{M}_{i+1}^{(et)} = \underbrace{\{(x,y,z,l,0) \mid \left((x,y,z',l',0) \in \bar{M}_i^{(et)} \wedge \exists (u,v,b) \in I \text{ with } u = x \wedge v = y \wedge (x,y,z,l,0) = enqueue(x,y,z',l',0,b)\right)\}}_{\text{communication system delivers event message to port}} \cup$$
$$\underbrace{\{(x,y,z,l,0) \mid (x,y,z',l',0) \in \bar{M}_i^{(et)} \wedge (u,v) \in O \wedge u = x \wedge v = y \wedge l = l' - 1\}}_{\text{communication system retrieves event message from port}} \cup$$
$$\underbrace{\{(x,y,z,l,0) \mid (x,y,z',l',0) \in \bar{M}_i^{(et)} \wedge x \in M_{out} \wedge \exists (i,j) \in \bar{V}_{i+1} \wedge i = y \wedge (x,y,z,l,0) = enqueue(x,y,z',l',0,j)\}}_{\text{send operation of an event message}} \cup$$
$$\underbrace{\{(x,y,z,l,0) \mid (x,y,z',l',0) \in \bar{M}_i^{(et)} \wedge x \in M_{in} \wedge l = l' - 1\}}_{\text{receive operation of an event message}} \cup$$
$$\underbrace{\{(x,y,z,l,0) \in \bar{M}_i^{(et)} \mid x \notin M_{out} \wedge x \notin M_{in} \wedge (x,y) \notin O \wedge \nexists (u,v,b) \in I \text{ with } u = x \wedge v = y\}}_{\text{neither send operation, receive operation, nor update of port through communication system}}$$

Networks can deliver messages to a port, thereby adding another queue element (first line in the definition of $\bar{M}_{i+1}^{(et)}$). Also, networks can retrieve a message from a port, thus leading to the removal of a message from the queue (second line in the defini-

tion). Furthermore, the addition or removal of messages can occur through the send and receive operations within the communication action (lines 3 and 4 in the definition).

In the definition of the port state update, we use a supporting function $enqueue$, which inserts an additional message at the queue of an event port. The message queue of an event port with a maximum length of n is represented as a vector of size n. $enqueue$ rotates all messages in this queue using a matrix multiplication of the vector. Subsequently, the new message is inserted at position 1 of the vector.

$$enqueue(x,y,z,l,n,d) = (x,y,z',l',d) \text{ where } l' = l+1 \wedge z' = z \cdot \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & 0 & \cdots & 0 & 0 \\ & & & & \\ 0 & 1 & & \vdots & \vdots \\ & & & & \\ 0 & 0 & \ddots & 0 & 0 \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix} + \begin{pmatrix} n \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

The update of the gateway repository state is similar to the update of the port state. However, the union contains only the unmodified convertible elements and the convertible elements altered by the communication actions. Unlike the ports, the gateway repository is only accessed by the gateway (and not by the networks). In case of an update-in-place, the most recent update instant t'' of the convertible element is set to the current global time t.

$$\overline{C}_{i+1}^{(tt)} = \underbrace{\{(x,y,z,l,t') \in \overline{C}_i^{(tt)} \mid x \notin C_{out}\}}_{\text{no push operation (state information)}} \cup \underbrace{\{(x,y,z,l,t'') \mid t'' = t \wedge (x,y,z',l,t') \in \overline{C}_i^{(tt)} \wedge x \in C_{out} \wedge \exists (i,j) \in \overline{V}_{i+1} \wedge i = y \wedge j = z\}}_{\text{variable is copied as part of a push operation of a convertible element with state information}}$$

$$\overline{C}_{i+1}^{(et)} = \underbrace{\{(x,y,z,l,0) \mid (x,y,z',l',0) \in \overline{C}_i^{(et)} \wedge x \in C_{out} \wedge \exists (i,j) \in \overline{V}_{i+1} \wedge i = y \wedge (x,y,z,l,0) = enqueue(x,y,z',l',0,j)\}}_{\text{push operation of a convertible element with event information}} \cup$$

$$\underbrace{\{(x,y,z,l,0) \in \overline{C}_i^{(et)} \mid x \notin C_{in}\} \cup \{(x,y,z,l,0) \mid (x,y,z',l',0) \in \overline{C}_i^{(et)} \wedge x \in C_{in} \wedge l = l'-1\}}_{\text{pull operation of a convertible element with event information}} \cup$$

$$\underbrace{\{(x,y,z,l,0) \in \overline{C}_i^{(et)} \mid x \notin C_{out} \wedge x \notin C_{in}\}}_{\text{neither push nor pull (event information)}}$$

The new variables state is the union of the variables which remain unchanged by communication actions, the variables which are assigned a new value through a pull operation (i. e., new value from the gateway repository), and the variables which are assigned a new value through a receive operation (i. e., new value from a port). In the definition of the variables state, we use a supporting function $front$, which yields the first element in the queue of a port or a convertible element in the gateway repository.

$$front(z,l,n) = z \cdot \mathfrak{e}_l \text{ where } \mathfrak{e}_l \text{ is the canonical unit vector of dimension } n$$
$$\mathfrak{e}_1 = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \end{pmatrix}^T,\ \mathfrak{e}_2 = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \end{pmatrix}^T, \mathfrak{e}_n = \begin{pmatrix} 0 & 0 & 0 & \cdots & 1 \end{pmatrix}^T$$

Finally, the global time and the clock state of an untimed transition remain unchanged (i. e., $\bar{Y}_{i+1} = \bar{Y}_i, t_{i+1} = t_i$).

Sequence of Timed and Untimed Transitions A gateway contains an ordered set of network adaptors. The execution of the network adaptors occurs in cycles. Starting with the first network adaptor, untimed transitions are taken as long as guards Φ of the first network adaptor are fulfilled. When no guard is satisfied any more, the execution proceeds with the second network adaptor. A cycle terminates with the last network

$$\bar{V}_{i+1} = \{(i,j) \mid (i,j) \in \underbrace{\alpha(\bar{Z}_i)}_{\substack{\text{variables after}\\ \text{assgnment}}} \wedge \underbrace{\nexists (u,v,p,n,d) \in C \text{ with } (i \in v \wedge u \in C_{in})}_{\text{no pull operation operation affecting this variable}} \wedge \underbrace{\nexists (u,v,p,n,d,\tau) \in M \text{ with } (i \in v \wedge u \in M_{in})}_{\text{no receive operation operation affecting this variable}}\} \cup$$

$$(i,j) \mid (i,j') \in \underbrace{\alpha(\bar{Z}_i)}_{\substack{\text{variables after}\\ \text{assgnment}}} \wedge \underbrace{\exists (u,v,p,n,d) \in C \text{ with } (i \in v \wedge u \in C_{in})}_{\text{pull operation affecting this variable}} \wedge \underbrace{(x,y,z,l,t,r) \in \bar{C}_i^{(st)} \wedge x = u \wedge y = i \wedge j = z}_{\substack{\text{value } z \text{ of state-conv.elem. read from the gateway repository}\\ \text{and used as new value } j \text{ of the variable } i}}\} \cup$$

$$(i,j) \mid (i,j') \in \underbrace{\alpha(\bar{Z}_i)}_{\substack{\text{variables after}\\ \text{assgnment}}} \wedge \underbrace{\exists (u,v,p,n,d) \in C \text{ with } (i \in v \wedge u \in C_{in})}_{\text{pull operation affecting this variable}} \wedge \underbrace{(x,y,z,l,t,r) \in \bar{C}_i^{(et)} \wedge x = u \wedge y = i \wedge j = \mathit{front}(z,l,n)}_{\substack{\text{value } z \text{ of event-conv.elem. read from the gateway repository}\\ \text{and used as new value } j \text{ of the variable } i}}\} \cup$$

$$(i,j) \mid (i,j') \in \underbrace{\alpha(\bar{Z}_i)}_{\substack{\text{variables after}\\ \text{assgnment}}} \wedge \underbrace{\exists (u,v,p,n,d,\tau) \in M \text{ with } (i \in v \wedge u \in M_{in})}_{\text{receive operation affecting this variable}} \wedge \underbrace{(x,y,z,l,t) \in \bar{M}_i^{(st)} \wedge x = u \wedge y = i \wedge j = \tau(z)}_{\substack{\text{value } z \text{ of state msg. read from the port}\\ \text{and used as new value } j \text{ of the variable } i}}\} \cup$$

$$(i,j) \mid (i,j') \in \underbrace{\alpha(\bar{Z}_i)}_{\substack{\text{variables after}\\ \text{assgnment}}} \wedge \underbrace{\exists (u,v,p,n,d,\tau) \in M \text{ with } (i \in v \wedge u \in M_{in})}_{\text{receive operation affecting this variable}} \wedge \underbrace{(x,y,z,l,t) \in \bar{M}_i^{(et)} \wedge x = u \wedge y = i \wedge j = \tau(\mathit{front}(z,l,n))}_{\substack{\text{value } z \text{ of event msg. read from the port}\\ \text{and used as new value } j \text{ of the variable } i}}\}$$

adaptor when no more untimed transitions of the last network adaptor can be executed, because no guard is satisfied. At this point, a timed transition is taken advancing the value of all clock variables by 1. Subsequently, the next cycle starts with the execution of untimed transitions of the first network adaptor.

$$\underbrace{\bar{Z}_k \underbrace{\xrightarrow[A_0]{I_k,O_k} \bar{Z}_{k+1} \xrightarrow[A_0]{I_{k+1},O_{k+1}} \cdots \xrightarrow[A_0]{I_{l-1},O_{l-1}} \bar{Z}_l}_{\substack{\text{guard of network adaptor } A_0\\ \text{is enabled}}} \underbrace{\xrightarrow[A_1]{I_l,O_l} \bar{Z}_{l+1} \xrightarrow[A_1]{I_{l+1},O_{l+1}} \cdots \xrightarrow[A_1]{I_{m-1},O_{m-1}} \bar{Z}_m}_{\substack{\text{guard of network adaptor } A_1\\ \text{is enabled}}} \xrightarrow[A_2]{I_m,O_m} \cdots \bar{Z}_n \underbrace{\xrightarrow[A_x]{I_n,O_n} \bar{Z}_{n+1} \xrightarrow[A_x]{I_{n+1},O_{n+1}} \cdots \xrightarrow[A_x]{I_{o-1},O_{o-1}} \bar{Z}_o}_{\substack{\text{guard of network adaptor } A_x\\ \text{is enabled}}}}_{\text{transitions of all network adaptors } A_0 \ldots A_x \text{ of the gateway}} \underbrace{\xrightarrow{T}}_{\substack{\text{timed}\\ \text{transition}}} \bar{Z}_{o+1} \underbrace{\xrightarrow[A_0]{I_o,O_o} \bar{Z}_{o+1} \xrightarrow[A_0]{I_{o+1},O_{o+1}} \cdots \xrightarrow[A_0]{I_p,O_p} \cdots}_{\substack{\text{guard of network adaptor } A_0\\ \text{is enabled}}}$$

4 Model-Based Generation of Gateways

Based on the introduced formal specification of the gateway, we have realized a tool for automatic code generation of gateways in a prototype implementation of the DECOS architecture [12, 13]. The protoype implementation consists of five nodes interconnected by the Time-Triggered Protocol (TTP) [14], a cluster with three nodes interconnected by the Controller Area Network (CAN), and a cluster with three nodes interconnected by the Local Interconnect Network (LIN). The gateways execute within the TTP nodes, each of which is a multiprocessor node consisting of a connector unit and two application computers.

The purpose of the connector unit is the implementation of the time-triggered communication protocol for the physical network. The connector unit provides the application computers with a global time base and supports the periodic exchange of state messages at a priori specified global points in time. The connector unit contains a TTP communication controller and is realized using a single board computer equipped with a MPC855 PowerPC from Freescale.

The application computers host the application software (i. e., jobs belonging to one or more DASs) in conjunction with the gateways. Each application computer is implemented on a Soekris net4521 embedded computer from Soekris Engineering[1], which is based on a 133 MHz 486 class ElanSC520 processor from AMD. We deploy on all application computers the real-time Linux variant LXRT/RTAI [15] extended by a time-triggered scheduler [12] as the operating system. Time-triggered LXRT/RTAI tasks are used both for executing the jobs containing the application code, as well as for the middleware implementing the gateways.

[1] www.soekris.com

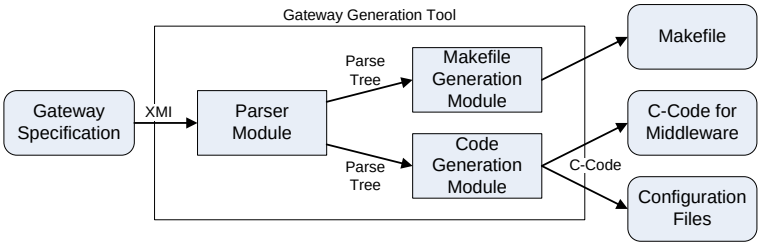

Fig. 2. Overview of Gateway Generation Tool

As an input, the code generation tool uses an instance of a UML meta-model that has been derived from the formal definition of a gateway in Section 3. UML was selected for the code generator due to the availability of code libraries (e. g., for parsing and checking compliance to the meta-model) that have eased the implementation of the code generation tool. In addition, a wide range of supporting tools (e. g., editors for creating UML models) can be used for creating gateway specification models.

Both the code for the network adaptors and configuration data structures are automatically generated from the gateway specification model using a *gateway generation tool*. The tool is based on the XML C parser toolkit developed for the Gnome project. It takes as input an XML Metadata Interchange (XMI) representation of the gateway specification UML model. The output of the code generation tool are C source files with code for the network adaptors and configuration data structures.

Figure 2 depicts the structure of the gateway generation tool. The parser module processes the XMI input and builds a parse tree in memory. The parse tree is used for producing code for the gateway middleware, as well as for constructing a makefile.

5 Discussion

The use of gateways for the interconnection of networks with different communication protocols is an important problem that has received much attention in previous work. Many authors have focused on formal specifications based on communicating finite state machines. This paper describes a novel solution for the specification of gateways based on a real-time database in-between the interconnected networks. The real-time database stores temporally accurate real-time images in conjunction with meta information (e. g., instant of most recent update, information w.r.t. to update requests). The major benefit of the real-time database is the ability for a constructive realization of gateways in distributed real-time systems. Large, complex gateways can be divided into smaller modules, which are not only simpler but facilitate reuse and localize changes. For each network, developers can independently specify which messages update the real-time database and which messages are sent with the information from the real-time database. The introduced state machines with timing constraints provide a powerful and intuitive formalism for this task. They enable developers to specify the protocols for accessing specific networks along with the corresponding syntax and naming transformations.

Acknowledgments

This work has been supported in part by the European IST project ARTIST2 under project No. IST-004527.

References

1. H.A. Simon. *The Sciences of the Artificial*. MIT Press, 1996.
2. G. Leen and D. Heffernan. Expanding automotive electronic systems. *Computer*, 35(1):88–93, January 2002.
3. Robert Bosch Gmbh, Stuttgart, Germany. *CAN Specification, Version 2.0*, 1991.
4. FlexRay Consortium. BMW AG, DaimlerChrysler AG, General Motors Corporation, Freescale GmbH, Philips GmbH, Robert Bosch GmbH, and Volkswagen AG. *FlexRay Communications System Protocol Specification Version 2.0*, July 2004.
5. R. Obermaisser. A model-driven framework for the generation of gateways in distributed real-time systems. In *Proc. of the 28th IEEE Real-Time Systems Symposium*, Tucson, Arizona, USA, September 2007.
6. R. Obermaisser, P. Peti, B. Huber, and C. El Salloum. DECOS: An integrated time-triggered architecture. *e&i journal (journal of the Austrian professional institution for electrical and information engineering)*, 3:83–95, March 2006. Available at http://www.springerlink.com.
7. H. Kopetz. *Real-Time Systems, Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, Boston, Dordrecht, London, 1997.
8. R. Alur. Timed automata. In N. Halbwachs and D. Peled, editors, *Lecture Notes in Computer Science*, volume 1633, pages 8–22. Spinger-Verlag, 1999.
9. B. Dutertre and M. Sorea. Modeling and verification of a fault-tolerant real-time startup protocol using calendar automata. In *Proc. of the FORMATS/FTRTFT'04, LNCS 3253*, pages 119–214, September 2004.
10. P. Krcal, L. Mokrushin, P.S. Thiagarajan, and W. Yi. Timed vs time-triggered automata. In *Proc. of the 15th International Conference on Concurrency Theory*, September 2004.
11. H. Kopetz and K. H. Kim. Temporal uncertainties in interactions among real-time objects. In *Proc. of Ninth Symposium on Reliable Distributed Systems*, pages 165–174, Huntsville, AL ,USA, October 1990.
12. B. Huber, P. Peti, R. Obermaisser, and C. El Salloum. Using RTAI/LXRT for partitioning in a prototype implementation of the DECOS architecture. In *Proc. of the Third Int. Workshop on Intelligent Solutions in Embedded Systems*, May 2005.
13. R. Obermaisser and P. Peti. Realization of virtual networks in the decos integrated architecture. In *Proc. of the 14th Int. Workshop on Parallel and Distributed Real-Time Systems*, April 2006.
14. *Time-Triggered Protocol TTP/C – High Level Specification Document*, July 2002.
15. D. Beal et al. RTAI: Real-Time Application Interface. *Linux Journal*, April 2000.