The Time-Triggered Model of Computation

H. Kopetz

Technical University of Vienna, Austria

hk@vmars.tuwien.ac.at

Abstract: The Time-Triggered (TT) model of computation is a model for the representation and analysis of the design of large hard real-time systems. Central to this model is the concept of temporal accuracy of real-time information. The TT-model consists of four building blocks, interfaces that contain temporally accurate data, a communication subsystem that connects interfaces, a host computer that reads input data from interfaces and writes output data to interfaces, and a transducer that transforms the information representation in the environment into the digital form of the interface and vice versa. These four building blocks can be used recursively to describe arbitrary large hard real-time systems. The TT-model separates cleanly the design of the interaction pattern among components from the design of the components themselves. It thus supports a compositional design. In the final section the TT-model is compared to the client-server model.

Keywords: Real-time system, design representation, time-triggered, client-server model, temporal accuracy of data.

1. Introduction

With the advent of cost-effective microcomputers on a single silicon die, distributed architectures are becoming the norm in real-time applications. The integration of a sensor, transducer and signal-processing within a single unit has led to the concept of a smart sensor that acts as a node in a real-time network. Many such intelligent sensor, actuator, and computational nodes are interconnected to form a cluster that provides the services of the real-time computer system. Such a distributed real-time computer system must produce the intended results within a specified window of real-time. In this paper we focus on real-time systems where a single failure to produce results on time constitutes a system failure, i.e., on *hard real-time systems*. Hard real-time systems are fundamentally different from soft real-time systems, where the occasional violation of a deadline is tolerated. Hard real-time systems are more difficult to understand and to design than soft real-time systems because the designer must consider two dimensions of the design problem simultaneously: the value dimension and the time dimension of each result. One will only succeed in designing large hard real-time systems

systematically if there is a reduced representation, a system model, of such a large system that captures the essential properties of the design problem while abstracting from irrelevant detail.

A distributed computer system consists of nodes interconnected by a communication systems. The prevalent computational model for the description of distributed applications is the client-server model. In the client-server model, one node, acting as a client node, requests a service from another remote node, acting as a service node, using the services of the interconnection network. The client-server model has evolved over many years from the field of non-time critical distributed applications. The Open System Foundation [OSF 1992] and the CORBA [OMG 1993] initiative try to standardize client-server interfaces to achieve application interoperability in all types of networks. Recently, an initiative has been started to extend the client-server model to the domain of real-time systems[OMG 1998]. Extending the client-server model to cater for real-time applications is difficult, because the fundamental notions of time and timeliness are missing from the original client-server model. It has yet to be demonstrated whether these fundamental notions can be integrated into the client-server model as an addendum. It is our belief that a computational model of a distributed real-time computer system must be based around the notion of time as a first order concept. The TT-model discussed in this paper is such a computational model in that it is based on time and timeliness of real-time information. The objective of the TT-Model is to provide the means for the high-level description of large real-time systems. The model focuses on the temporal accuracy of the data elements in the interfaces between the subsystems.

It is the objective of this paper to present a time-triggered (TT) model for the representation of large distributed real-time systems. The rest of this paper is organized as follows. In the next section we introduce the concepts of real-time (RT) entities and real-time (RT) images and define the temporal accuracy relationship between an RT entity and the corresponding RT image. After a short discussion about layering versus partitioning, the building blocks and the structure of the TT-model are introduced in the core Section 4. The relation between the TT-model of computation and the time-triggered architecture as an implementation architecture is discussed in Section 5. Section 6 gives an example of a TT system model. Section 7 compares the TT-model with the client server model of a distributed computation. Finally, Section 8 concludes the paper.

2. Temporal Accuracy of Information

At the core of the TT-model is the notion of the temporal accuracy of real-time data. Real-time data loses its validity as time progresses. In order to refine this notion, the concepts of a real-time (RT) entity and of a real-time (RT) image as a picture of a RT entity are introduced [Kopetz 1997].

Real-time Entity: A controlled object, e.g., a car or an industrial plant, changes its state as a function of time. If we freeze time, we can describe the current state of the controlled object by recording the values of its state variables at that moment. We are normally not interested in *all* state variables, but only in the *subset* of state variables that is *significant* for our purpose. A significant state variable is called a *real-time (RT) entity*. Every RT entity is in the *sphere of control (SOC)* of a subsystem, i.e., it belongs to a subsystem that has the authority to change the value of this RT entity. Outside its sphere of control, the value of an RT entity can be observed, but cannot be modified.

A *real-time (RT) entity* is thus a state variable of relevance for the given purpose, and is located either in the SOC within the environment or in the SOC within the computer system. Examples of RT entities are the flow of a liquid in a pipe (in the SOC of the controlled object), the setpoint of a control loop (in the SOC of the operator), and the intended position of a control valve (in the SOC of the real-time computer). An RT entity has static attributes that do not change during the lifetime of the RT entity, and has dynamic attributes that change with time. Examples of static attributes are the name, the type, the value domain, and the maximum rate of change. The value set at a particular point in time is the most important dynamic attribute. Another example of a dynamic attribute is the rate of change at a chosen point in time.

Observation: The information about the state of an RT entity at a particular point in time is captured by the notion of an *observation*. An observation is an *atomic data structure*

$$Observation = <Name, t_{obs}, Value>$$

consisting of the name of the RT entity, the point in real time when the observation was made (t_{obs}), and the observed value of the RT entity. A continuous RT entity can be observed at any point in time while a discrete RT entity can only be observed when the state of this RT is not changing.

Real-time Image: A *real-time (RT) image* is a *current* picture of an RT entity. An RT image is valid at a given point in time if it is an accurate representation of the corresponding RT entity, both in the value and the time domains. While an observation records a fact that remains valid forever (a statement about an RT entity that has been observed at a particular point in time), the validity of an RT image is *time-dependent* and thus likely to be invalidated by the progression of real-time.

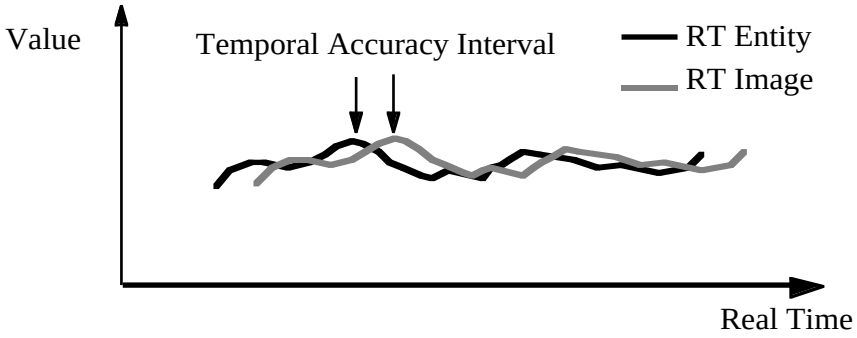

Figure 1**:** Time lag between RTentity and RTimage.

Temporal Accuracy: *Temporal accuracy* is the relationship between an RT entity and its associated RT image [Kopetz and Kim 1990]. The temporal accuracy of an RT image is defined by referring to the *recent history* of observations of the related RT entity. A recent history RH_i at time t_i is an ordered set of time points $\{t_i, t_{i-1}, t_{i-2}, \ldots . t_{i-k}\}$, where the length of the recent history, $d_{acc} = z(t_i) - z(t_{i-k})$, is called the *temporal accuracy interval* or the *temporal accuracy*. ($z(e)$ is the timestamp of event e generated by a reference clock z). Assume that the RT entity has been observed at every time point of the recent history. An RT image is temporally accurate at the present time t_i if

$$\exists\, t_j \in RH_i : Value\,(RT\,image\,at\,t_i) = Value\,(RT\,entity\,\,at\,t_j)$$

The present value of a temporally accurate RT image is a member of the set of values that the RT entity had in its recent history. Because the transmission of an observation message from the observing node to the receiving node takes some amount of time, the RT image lags behind the RT entity (See Figure 1).

The size of the admissible temporal accuracy interval is determined by the dynamics of the RT entity in the controlled object. The delay between the observation of the RT entity and the use of the RT image causes an error, *error(t)* of the RT image that can be approximated by the product of the gradient of the value *v* of the RT entity multiplied by the length of the interval between the observation and its use:

$$error(t) = \frac{dv(t)}{dt}\,(z(t_{use}) - z(t_{obs}))$$

If a temporally valid RT image is used, the worst-case error,

$$error = \left(\max_{\forall t} \frac{dv(t)}{dt}\, d_{acc}\right),$$

is given by the product of the maximum gradient and the temporal accuracy d_{acc}. In a balanced design, this worst-case error caused by the temporal delay is in the same order of magnitude as the worst-case measurement error in the value domain, and is typically a fraction of a percentage point of the full range of the measured variable.

State Estimation: The most important future point in time where the RT image must be in close agreement with the corresponding RT entity is t_{use}, the point in time where the value of the RT image is used to cause an action in the environment. If the time it takes to transport an observation from a node that observes a RT entity to another node that performs a computation and the output to the environment is longer than the temporal accuracy interval d_{acc}, then the state of the RT image at the time of use t_{use} must be estimated by a process called state estimation. State estimation involves the building of a model of an RT entity inside a computer to compute the probable state of an RT image at a selected future point in time. State estimation is a powerful technique to extend the temporal accuracy interval of an RT image, i.e., to bring the RT image into better agreement with the RT entity.

If the behavior of an RT entity can be described by a continuous and differentiable function *v(t)*, the first derivative *dv/dt* is sometimes sufficient in order to obtain a reasonable estimate of the state of the RT entity at the point t_{use} in the neighborhood of the point of observation:

$$v(t_{use}) \approx v(t_{obs}) + (t_{use} - t_{obs})\, dv / dt$$

If the precision of such a simple approximation is not adequate, a more elaborate series expansion around t_{obs} can be carried out. In other cases a more detailed mathematical model of the process in the controlled object may be required. The execution of such a mathematical model can demand considerable processing resources.

The most important dynamic input to the state estimation model is the precise length of the time interval $[t_{obs}, t_{use}]$. Because t_{obs} and t_{use} are normally recorded at different nodes of a distributed system, a communication protocol with small jitter (the jitter of a communication protocol is the difference between the maximum and the minimum protocol execution time) or a global time-

base with a good precision is a prerequisite for state estimation. This prerequisite is an important requirement for the design of a field bus.

3. Layering versus Partitioning

Model building implies the design of a framework that can be used to describe the reality at an abstract, i.e. less detailed, level. Two kinds of structuring of a computer system can be distinguished to reduce the system complexity: *horizontal* versus *vertical* structuring.

(i) Horizontal structuring (or *layering*) is related to the process of stepwise abstraction, of defining successive hierarchically-ordered new layers that are reduced representations of the system. Many software-engineering techniques (e.g., structured programming, virtual machines) propose one or another form of horizontal structuring.

(ii) Vertical structuring is related to the process of *partitioning* a large system into a number of *nearly* independent subsystems with their own resources (CPUs, memory, software) and with *well-specified interfaces* (in the temporal and value domain) among these subsystems so that these subsystems can be validated in isolation of each other. In distributed real-time systems *clusters* and *nodes* are the tangible units of partitioning.

While in a central computer system, layering is the common technique to combat complexity, the designer of a distributed computer system can take advantage of both techniques. In the following section we focus on system partitioning.

A large distributed computer application is normally partitioned into clusters. Some clusters are in the environment, e.g., the process that is to be controlled, and some clusters form the controlling computer system, the computational clusters (Figure 2). Each computational cluster consists of a set of nodes N that are interconnected by a communication system. A node that is a member of two clusters--we call such a node a gateway node G-- provides the interconnection between these two clusters. A node that transduces information from/to an environmental cluster is called a *transducer* T. Figure 2 shows a system consisting of three computational and two environment clusters.

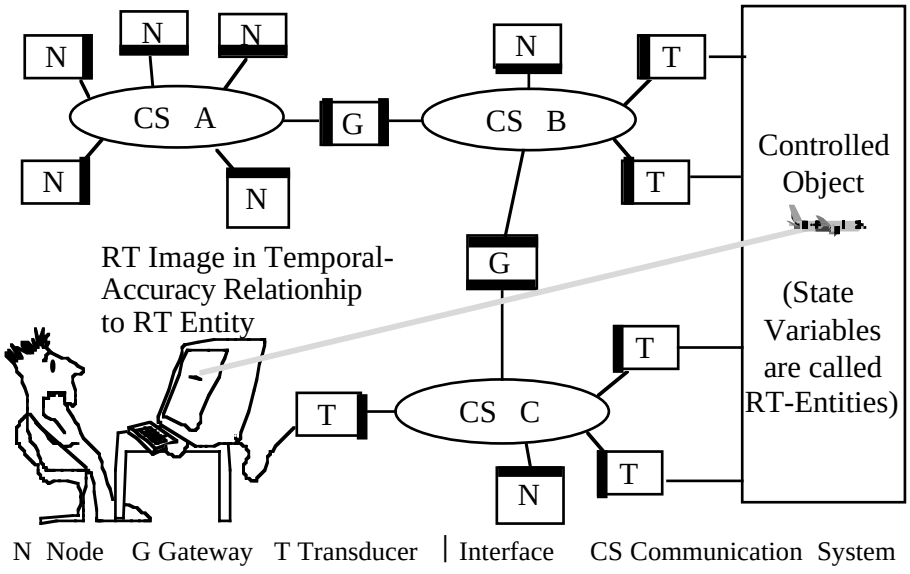

Figure 2: A distributed computer system consisting of three computational and two environment clusters.

From the point of view of design, a real-time system is first partitioned into nearly decomposable subsystems of high inner connectivity and low external connectivity. Some of these subsystems are then mapped into clusters and nodes of the distributed computer system. In a second step, each node can be structured internally according to the layering technique.

In many embedded applications this partitioning of a distributed computer system into nodes and clusters is dictated by the structure and the constraints of the controlled object. For example, in a distributed vehicle system it is expedient to assign a distinct node--or even a complete cluster--to the control of the engine, to the control of the brakes and to the control of the body electronics. Such a functional partitioning of the hardware can lead to a number of advantages concerning composability, error containment, and reusability:

(i) In a partitioned system, where few computational tasks share a processor, there is a reduced need for resource multiplexing. Resource multiplexing introduces complexity and temporal unpredictability into the system behavior. It is thus easier to arrive at tight execution time bounds of computations if there is only limited resource sharing among the tasks of a node.

(ii) The abstractions of partitioned systems also hold in case of failures. While in a layered system, it is difficult to define clean error-containment regions for the case of a fault in a shared resource (e.g., the CPU), the partitions (nodes and clusters) of a distributed system coincide with the units of failures where small and observable interfaces (the message interfaces) around these error-containment regions facilitate the error detection and error containment.

(iii) The complexity of an implementation of a partition can be hidden behind the precise specification of the communication network interface between the partition and the rest of the world. The attributes of data objects (value and temporal) that are contained in this interface can form the core of such a precise interface specification.

(iv) The future availability of highly integrated system chips makes partitioning and clustering economically and conceptionally attractive. The economic attractiveness is a consequence of the immense economies of scale of the semiconductor industry. The conceptual attractiveness results from reusing the same design pattern over and over again.

4. The TT Model of Computation

The Time–Triggered (TT) model of computation is based on partitioning a large distributed computer system into nearly autonomous subsystems with small and stable interfaces between these subsystems. It describes a large real-time system by the repetitive use of the following four basic building blocks:

(i) An *interface*, i.e., a boundary between two subsystems,

(ii) A *communication system* that connects interfaces,

(iii) A *host computer* that reads data from one or more interfaces, processes the data and writes data into one or more interfaces, and

(iv) A *transducer* that connects an RT entity in the environment to an interface or vice versa.

It is assumed that each one of these building blocks has access to a globally synchronized time base of sufficient precision. In the following sections we describe these building blocks in more detail. Section 5 relates these basic building blocks to the physical units, i.e., the nodes and the clusters of a distributed computer system.

4.1 Interface

The most important concept of the TT-model is the interface. An interface consists of a memory element that is shared between two interfacing subsystems and that contains valid RT images of the relevant RT entities. An interface can be viewed as a dual-ported memory, where the information-producing subsystem updates the RT image periodically in order to ensure that the RT image in the interface is valid. The information-consuming subsystem reads this temporally accurate information whenever needed.

Temporal Firewall: The concept of a *temporal firewall* [Kopetz and Nossal 1997] has been developed to describe the properties of interfaces in the TT-model. We distinguish between two types of temporal firewalls, *phase-insensitive (PI)* and *phase-sensitive (PS)* temporal firewall.

*A **PI temporal firewall** is a unidirectional **data-sharing interface** with **state-data semantics** where **at least one** of the interfacing subsystems accesses the temporal firewall according to an **a priori known time-triggered schedule** and where at all points in time the information contained in the temporal firewall can be assumed to be **temporally accurate** for at least d_{acc} time units into the future.*

There are no control signals crossing a temporal firewall. The information provider has to update the RT image in the temporal firewall according to the dynamics of the corresponding RT entity. If the information-providing subsystem ceases to operate, the information in the temporal firewall is likely to become invalidated by the passage of time.

If a user (e.g., a process in a host computer) uses a data-element of a PI temporal firewall within d_{acc} interval time units after reading then the result will be temporally valid at the time of use. Such users can access the data items of the temporal firewall at arbitrary instants without running the risk that the RT image will become invalid before it is used.

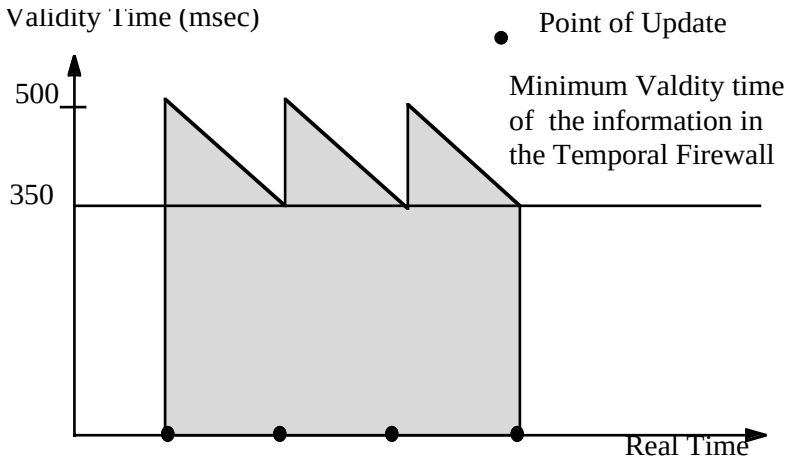

Figure 3: Information validity in the PI firewall of the previous example.

Example: Consider an RT-entity that denotes the temperature of the oil in an engine. The temperature changes at most 1 % of the range per second. An accuracy in the value domain of

.5 % is required. Assuming that the measurement is precise at the point of observation, d_{acc} is 500 msec at the instant of observation. If it takes 50 msec to transport the RT image to the temporal firewall of the receiver and the update period is 100 msec, then the remaining d_{acc} in a PI temporal firewall is 350 msec. If the update period is 50 msec, then d_{acc} is 400 msec.

If a system designer intends to make full use of the temporal validity of the firewall information at the point of update, the user tasks must be synchronized with the update points.

*A **PS temporal firewall** is a unidirectional **data-sharing interface** with **state-data semantics** where the **information producer** accesses the temporal firewall according to an **a priori known time-triggered schedule** and where the information contained in the temporal firewall is **temporally accurate** for at least d_{acc} time units into the future at the instant when the information producer delivers the information to the temporal firewall.*

When a user reads information from a PS temporal firewall it must ensure that the time interval between its use of the information and the point in time of information delivery by the producer is less than d_{acc} time unit or it must perform state estimation. The name "phase-sensitive" temporal firewall expresses the need for a user task to synchronize its start of execution with the *a priori* known instant of information delivery.

Example: Consider an RT-entity that denotes the position of a crankshaft of an engine. The maximum speed of the engine is 6000 revolutions per minute, i.e., one revolution per 10 msec. The required accuracy of the position measurement for fuel injection is about .1 degree, corresponding in the time domain to a temporal accuracy of about 3 μsec. With these parameters it is not feasible to built an PI firewall, since it is impossible to use the RT-image within 3 μsec after it has been observed. In this application a PS firewall with state estimation is required. Since an engine operation is regular, it is possible to estimate the future state of the position of the crankshaft provided the position, the speed, and the acceleration of the crankshaft is known at the point of observation. The producer of this information knows *a priori* when precisely the information will be delivered at the consumer's firewall. It can perform the state estimation of the position, the speed, and the acceleration for the instant when the RT image is made available to the consumer's firewall. The consumer can then perform a second state estimation to estimate the position of the crankshaft at the time of use t_{use} when the output signal to open the valve is relayed to the I/O circuitry.

Task scheduling is simpler if the temporal firewall is phase insensitive than if it is phase sensitive. It depends on the temporal properties of the application and the available communication resources whether the simpler PI temporal firewall can be designed.

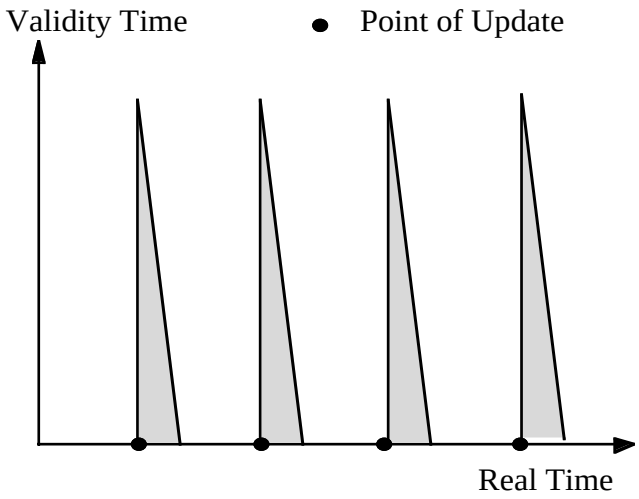

Figure 4: Information validity in the PS firewall of the previous example.

Stable Properties: The following stable properties characterize a temporal firewall. In the TT-model it is assumed that knowledge about these properties is available *a priori* to all interfacing subsystems:

(i) The addresses (names) and the syntactic structure of the data items in the temporal firewall. The meaning of the data items is associated with these names.

(ii) The points on the global time base when the data items in the temporal firewall are accessed by the TT subsystem. This information enables the avoidance of race conditions between the producer and the consumer. A race condition could lead to a loss of replica determinism in replicated temporal firewalls.

(iii) The temporal accuracy d_{acc} of the data items in the temporal firewall. This knowledge is important to guide the information consumer about the minimum rate of sampling the temporal firewall. The absolute time-points when the TT subsystem accesses the temporal firewall are reference points for the temporal accuracy of the information in the PS temporal firewall and can be used as the basis for state estimation.

Producer Obligation: The producer of the RT-images stored in the temporal firewall is responsible that the *a priori* guaranteed temporal accuracy of the RT-images is *always* given. It must update the state information with such a frequency that the guaranteed temporal accuracy is maintained, in the case of a PI firewall even immediately before the point of update. In case the producer of the information is the TT subsystem, the producer is allowed to access the temporal firewall only at the *a priori* established time points *t* to avoid race conditions for access to the temporal firewall. In case the producer of the information is not the TT subsystem, the producer is allowed to access the temporal firewall at any point in time outside a *critical interval* around *t*.. The duration of this critical interval is $[t\text{-}2g,\ t+d_{acd}+2g]$, where *t* is the *a priori* known access instant of the consumer, d_{acd} is the access duration of the consumer, and *g* is the granularity of the global time [Kopetz 1997, p.55].

Consumer Obligation: Based on the *a priori* knowledge about the temporal accuracy of the RT images in the temporal firewall, the consumer must sample the information in the temporal firewall with a sampling rate that ensures that the accessed information is temporally accurate at its *time of use* of this information. The consumer is only allowed to access the information in the temporal firewall when it knows (based on the *a priori* knowledge) that the producer is not

accessing it. If the consumer violates these access constraints, replica determinism may be lost, or, in the worst case, the consumed information may be corrupted.

The implementation of protected-shared objects (PSO) can avoid information corruption, but cannot guarantee replica determinism for the following reason: If two replicas perform an exclusive operation on a PSO at about the same time, it cannot be guaranteed that the temporal order of access to the PSO is the same in both replicas. In case it is different, a loss of replica determinism between these two replicas may occur.

4.2 Communication System

The time-triggered communication system connects interfaces and transports data elements from one interface to one or more other interfaces within *a priori* known deterministic time bounds. The semantics of the transported data is state-message semantics, i.e., a new version of a message replaces the current version in the receiving interfaces. State messages are not consumed on reading. There is thus no queuing of messages in the interfaces. The points in time when a message is taken from the sending interface and is delivered to the receiving interfaces are stored in dispatching tables. The contents of these dispatching tables are designed before run time and are *common knowledge* to all communicating partners. The temporal properties of the interfaces are thus precisely specified and do not change during the operation of the communication system.

The computational interaction pattern between the nodes of a cluster is reduced to the deterministic interaction between interfaces, established by the time-triggered communication system. The precise specification of the temporal and value properties of the interfaces can be developed without any knowledge about the (local) host implementation. The TT-model thus supports composability and host heterogeneity.

The operation of the time-triggered communication system between interfaces can be compared to the operation of a train system between stations. There is an *a priori* known time table that informs the clients about when a train is expected to arrive at a station and when a train will leave a station. The train system operates deterministically and independently from the activity at the stations and is synchronized with a known time standard. The client has to adapt to the time schedule of the train system.

4.3 Host Computer

A host computer is an encapsulated computational machine including one or more processing units, a memory, system software and application software. The host computer reads the input data from one or more interfaces and writes the output data into one or more interfaces. The instants when input data to the host computer are delivered at the interfaces and when output data from the host computer are fetched from the interfaces are known *a priori*. If a host computer has to react to a specific state change (event) it has to periodically sample the respective interface data items and trigger some action if the anticipated state change has been observed. The sphere of temporal control is thus always within the host computer and not delegated to the outside of the host computer.

The notion of an interrupt, i.e., an external event that interrupts a computation in a host computer, is foreign to the TT-model. The processing of such an external interrupt would

consume an unpredictable amount of computational resources of the host computer and would thus conflict with the predictability requirement.

4.4 Transducer

A transducer translates the environment's representation of an RT entity to the digital format of the corresponding RT image that is stored in the associated interface and vice versa. The time delay between sampling the RT entity in the environment and writing the RT image into the interface is constant and known *a priori*. Transducers are introduced to model the input/output system of a real-time system. From the modeling point of view it does not matter whether the I/O device is an analog sensor or a human operator behind a terminal. What is relevant is (i) that the I/O data is available in the corresponding interface within a known time-interval after it has been created and (ii) how long this RT image is temporally valid. The same reasoning, but in the opposite direction, applies to the output data generated by the computer system.

5. Implementation Issues

The TT-model described in this paper is a conceptual model for the design representation and the analysis of large real-time systems. Such a design representation of a system can only be implemented effectively if the semantic gap between this design representation and an object architecture can be bridged without intricated transformations that destroy the understandable structure of the design representation. The Time-Triggered Architecture (TTA) [Kopetz 1998a] is an exemplary architecture for the implementation of the TT-model.

5.1 Node

Figure 5 depicts the structure of a standard TTA node. A standard node consists of two subsystems, a host computer and a time-triggered communication protocol (TTP) controller with a controller internal data structure (the message descriptor list MEDL) that determines when a message must be sent or received.

The host computer of the node corresponds with the host computer in the TT model. It is a self-contained computer with its own operating system and the application software. It interfaces to the communication controller via the communication network interface CNI. The CNI is the concrete implementation of the interface building block of the TT model.

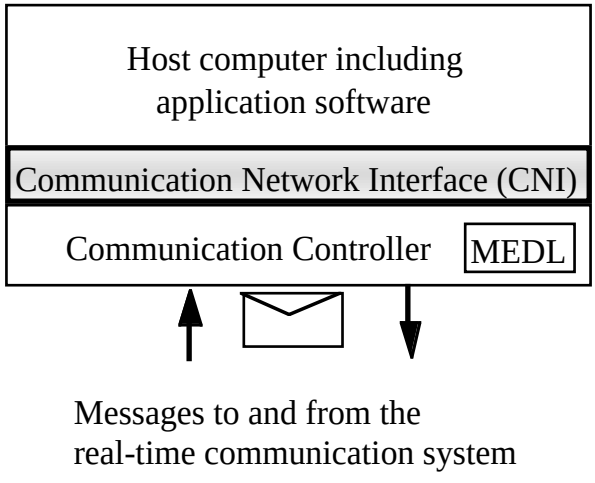

Figure 5: Structure of a standard TTA node.

The set of communication controllers of all nodes within a cluster, including the physical media, forms the communication system of the cluster. Every node contains a part of the communication system. If a node is a gateway node (see Figure 2) it contains two communication controllers and thus is a member of two clusters. A transducer node can be considered as a special gateway node that contains an I/O interface to an environment cluster.

5.2 Services of the Communication System

The time triggered communication protocol TTP provides the clock synchronization service that is needed for the implementation of the TT-model. The TTP communication controller contains its own data structure, the message descriptor list MEDL, that specifies the global interaction pattern among the nodes of a cluster and the temporal and value parameters of the communication network interface (CNI). There are a number of mechanisms in the TTP controller to increase the dependability. To avoid that a malicious host can interfere with the global communication system, the MEDL is not made accessible from the host computer. The TTP controller contains two independent physical channels such that the loss or disturbance of one channel can be tolerated. A special device in the TTP controller, the bus guardian, ensures that the controller will not access the bus outside its planned time slot, even if the controller itself becomes faulty[Temple 1998].

6. Example

Consider the example of an air traffic control system consisting of three computational and two environment clusters as shown in Figure 2. Cluster B and Cluster C interface to complex radar equipment that monitors the airspace. Cluster A, a computation cluster, dynamically computes some safety parameters, e.g., the expected future distance of the observed planes in order to avoid mid-air collisions. Cluster C interfaces to the operator.

The main interest of the operator at its interface is the temporally accurate display of the RT images of the RT entities that are in the monitored airspace. Figure 6 tries to capture this abstraction. Similarly, the computational cluster A needs the up-to-date attributes of the RT images to perform the collision avoidance calculations.

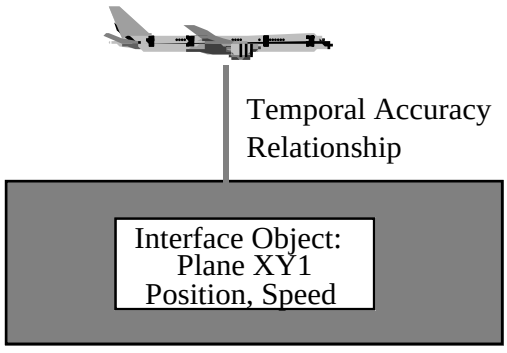

Figure 6: Temporal firewall data at operator interface.

A high-level requirements analysis of such a system must first try to capture all relevant RT entities. It then proceeds by identifying the temporal firewalls and the RT images that must be contained in each of these firewalls. In the next step the required temporal validity of the RT

images in the temporal firewalls must be established. The points in time when the interface objects are accessed (read or write) by the TT communication system will be determined by the schedule of the time-triggered communication system that transports the RT images between the interfaces.

At this point the architecture design phase is completed. The global interaction patterns among all subsystems have been established. The temporal and value attributes of all interfaces are precisely specified despite the fact that no knowledge about the hardware architecture, the operating system and the application software structure of the hosts is yet available (nor needed). The precise interface specification is sufficient to specify, implement and validate the interaction patterns among the components. The TT-model thus supports a compositional design.

In the next phase, the component design phase, the given interface specifications are taken as constraints for the development of the host and the transducer subsystems. The development of these subsystems can proceed in parallel, because the interactions among the subsystems are known and fully specified in the temporal domain and in the value domain at this phase of the design.

The host systems interfacing to the temporal firewalls can be heterogeneous computer systems or even legacy systems, provided they meet the temporal firewall specifications. Each subsystem can be based on a different hardware architecture and can use a different real-time operating system. As long as the value and temporal attributes of the interface objects in the temporal firewalls are satisfied, the interoperability of the subsystems is guaranteed. Each subsystem can be tested in isolation with respect to the interface specification.

During system integration the subsystems are integrated with the communication system to perform the system function. The interface properties are not modified by this integration. System integration is therefore a straightforward process, as has been demonstrated in an industrial application [Hedenetz and Belschner 1998].

7. The Client-Server Model versus the TT Model

In this section the TT-model is compared to the most widely used model for the description of distributed real-time systems, the client-server model. The client-server model assumes that one application, the *client*, requests the services from another application, the *server*, that can be at a different node of the distributed system. The client-server model provides a process-based horizontal service interface to the client at the API (application program interface) within the client node (Figure 7). Its focus is on the request-response transactions between a client process and a server process, possibly using the services of some intermediate brokers.

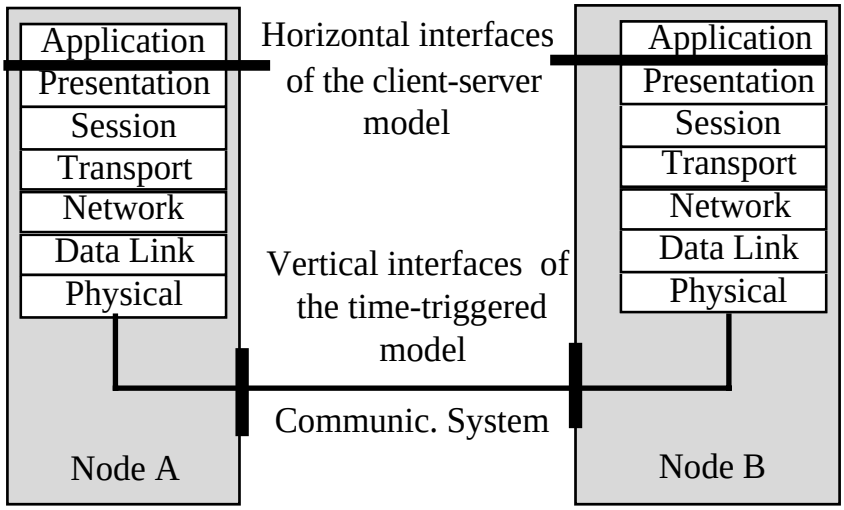

Figure 7: Interfaces in the client-server and the TT model.

In contrast, the TT model provides a state based vertical interface (Figure 7) at the boundary of a host, describing the temporal and value properties of the data exchanged between the interfacing partners. The internal structure of the interfacing subsystems is of subordinate concern to the TT model. Table 1 compares the characteristics of the two models.

The key problem in the application of the client-server model to distributed hard real-time systems is the missing phase control among the initiations of the remote client-server requests. Since the start-of-transaction commands are not globally coordinated there is always the possibility of a *critical instant* [Liu and Layland 1973] when all clients request services from the global communication system and the servers at the same point in time. No matter what scheduling strategy is employed, some transactions will have to wait for resources until other transactions have completed. This unpredictable jitter caused by these global interactions destroys the temporal composability.

Characteristic	Client-Server	TT-Model
Focus	Process based	State based
Structuring	horizontal-layer	vertical-partition
Driven by	Event messages	Sampling of states
Locus of Control	Global system wide events	Local within subsystem
Temporal concern	Timely request response transactions	Temporal accuracy of interface data
Temporal composability	Not supported	Supported

Table 1: Comparison of Client-Server model and TT-model.

One solution that has been proposed to solve this problem is the provision of as many resources as are required at the critical instant. This is a considerably more expensive solution than the *a priori* coordination of the interactions as performed in the TTA. Furthermore, it does not lead to composable design. From the economic viewpoint it is important to make the right choice about the implementation architecture (Table 2) for a soft and hard real-time application.

Architecture	Hard real-time application	Soft real-time application
Time-Triggered Architecture	adequate	Too expensive, because there is no need to provide all the resources required to handle all specified load and fault scenarios.
Client-Server Architecture	Too expensive, because the fear that the computer system might miss its deadline at the critical instant leads to the installation of hardware overcapacity with a low resource utilization.	adequate

Table 2: Choice of implementation architecture for hard and soft real-time systems.

Since many applications contain soft real-time functions and hard real-time functions in parallel, it can be economical to partition the system into a hard real-time partition and a soft real-time partition with a gateway between these partitions. The hard real-time parts of the system can be designed according to the TT paradigm and the soft real-time parts according to the client-server paradigm.

8. Conclusion

At the core of the TT-model are precisely specified interfaces and a predictable time-triggered communication system that decouples the interactions among the subsystems of a large distributed hard real-time system from the data processing functions at the nodes. The communication system defines the precise interaction patterns among these subsystems in the temporal domain and in the value domain. It transports RT images from one interface to another interface and does not have any knowledge about the hardware environment, the software environment, or the application task structure in the subsystems behind the interfaces.

The TT-model supports a two-phase design methodology, an architecture design phase and a component design phase. The interaction patterns among the subsystems and the precise temporal and value attributes of the interface objects are defined during an architecture design phase. For the ensuing component design phase, these interface specifications are considered to be implementation constraints. Since the temporal properties of the interface specifications do not change during system integration, the TT-model is composable and supports the constructive implementation of large real-time systems.

In the final section, the TT model has been compared with the client server. It is proposed to partition large real-time systems that contain soft and hard real-time functions into a hard real-time partition and a soft real-time partition.

A prerequisite for the implementation of the TT-model is the availability of an implementation of the time-triggered communication system model. Such a time-triggered communication system based on the TTP protocol has been developed and implemented at the Technische Universität

Wien during the last ten years. The future availability of VLSI controllers for TTP will open many new opportunities for the constructive design of large real-time systems.

Acknowledgments

Constructive comments on an earlier version of this paper by Brian Randell and Peter Puschner are warmly acknowledged. This paper has been supported by the ESPRIT LTR project DEVA, by the ESPRIT OMI project TTA, by the Brite Euram project X-by-Wire.

References

[Hedenetz and Belschner 1998]. Hedenetz, B. and R. Belschner (1998). "Brake by Wire" without Mechanical Backup by Using a TTP Communication Network. *SAE World Congress*, Detroit Michigan. SAE Press, Warrendale, PA, USA.

[Kopetz and Kim 1990] Kopetz, H. and K. Kim (1990). Temporal Uncertainties in Interactions among Real-Time Objects. *Proc. 9th Symposium on Reliable Distributed Systems*, Huntsville, AL, USA. IEEE Computer Society Press. pp. 165-174.

[Kopetz 1997] Kopetz, H. (1997). *Real-Time Systems, Design Principles for Distributed Embedded Applications; ISBN: 0-7923-9894-7*. Boston. Kluwer Academic Publishers.

[Kopetz 1998] Kopetz, H. (1998). A Comparison of CAN and TTP. Technische Universität Wien, Institut für Technische Informatik.

[Kopetz 1998a] Kopetz, H. (1998). The Time-Triggered Architecture. *ISORC 1998*, Kyoto, Japan. IEEE Press.

[Kopetz and Nossal 1997] Kopetz, H. and R. Nosssal (1997). Temporal Firewalls in Large Distributed Real-Time Systems. *Proceedings of IEEE Workshop on Future Trends in Distributed Computing*, Tunis, Tunesia. IEEE Press.

[Liu and Layland 1973] Liu, C. L. and J. W. Layland (1973). Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *J. of the ACM*. Vol. **20**. pp. 46-61.

[OMG 1993] OMG (1993). Object Managment Group: The common object request broker, architecture and specification (CORBA). Object Management Group, Framingham, Mass.

[OMG 1998]. OMG(1998). Real-Time CORBA, Request for Proposals. Object Management Group, Framingham, Mass.

[OSF 1992] OSF, 9. (1992). *Introduction to OSF DCE, Open System Fondation*. Englewood Cliffs, N.J. Prentice Hall.

[Temple 1998] Temple, C. (1998). Avoiding the Babbling Idiot Failure in a Time-Triggered Communication System. *Fault Tolerant Comp. Symp. 28*, Munich, Germany. IEEE Press.