An approach to formalize object interactions in distributed real-time systems

Michael Mock

Fraunhofer Institute for Autonomous intelligent Systems
D-53754 St. Augustin, Germany
e-mail: mock@ais.fhg.de

1. Introduction

Distributed control systems are becoming more complex and more heterogeneous. On the one hand, technological trends and application needs lead to an increased autonomy of the individual components: hardware (controllers, PLCs, industrial PCs) is equipped with more and more computing capabilities, networking layers and infrastructure (field-busses, industrial Ethernet, even wireless communication) are enabling interactions over several system hierarchies, and applications are more demanding with respect to the integration of higher level control and management systems, and require integration with distributed programming systems. On the other hand, even in complex, decentralized scenarios, interactions between the individual, autonomous nodes must be coordinated for meeting global real-time requirements. Hence, methodologies are needed that allow for describing and analyzing the structure of heterogeneous, distributed systems, the interactions of objects in such systems, and that provides a means to describe and evaluate the real-time facilities of these systems.

In [1], different distributed programming models (shared data space, object oriented frameworks, event based models) and their relation to communication infrastructure have been investigated for establishing distributed control structures. The approach described in this paper extends this work by developing a coherent, formal framework that allows to formally describing the effects of distributed programming structures on object interactions and communication with respect to their real-time characteristics. The classical approach to express and implement real-time requirements is that of "deadlines". However, deadlines are imposed at the level of a rather specific system and programming models, either processes or messages, or distributed threads in real-time COBRA. As we want to evaluate different distributed programming models and system structures for their feasibility real-time requirements, the notion of "deadline" must be independently from this model. For this purpose, we propose the use of the abstract notion of "precision distance" to express the application's timing requirements between different objects. Since the precision distance is defined independently of a specific programming or communication model, it can be used to describe timing requirements at the abstract object level, leaving the choice open of how the objects communicate and interact with each other.

Only few work addresses the problem formalizing real-time requirements for networks of interacting objects. In [2], the effects of communication jitter on object interactions are analyzed, but only specific object-relations are addressed. The general notion o f the precision distance is introduced for the special case of replicated objects. In [3], time-triggers in real-time database systems are used to describe time coherence between dependent objects in a special database application context. The model of [4] extends this approach to (data-) objects, but is not distributed and is restricted to consistency considerations (rather than real-time considerations). In particular, control loops cannot be modeled. In contrast to that, [5] presents a model completely based on (multi-rate control) loops, but does not consider no distribution of the effects of different programming models. Finally, real-time UML [6] provides sequence charts (alternatively concurrency charts) as basic means for describing interactions between objects, where different communication styles are expressed by different styles of arrows and timing requirements can be expressed by timing annotations. However, the effects of distribution and of different communication QoS characteristics are not reflected at the modeling level. Furthermore, there is no translation of the graphical notation in a formal notation that could be used in formal analysis.

The roadmap of the paper is as follows: section 2 introduces the model for describing interactions between the objects, section 3 applies the model to an example from industrial automation, and section 4 gives some conclusions.

2. The formal model

Similar to the approach in [2], we the basic structuring principle is based on objects that encapsulate data and methods. Objects can interact by the means of communication, whereby the structuring principle of communication at the programming level is explicitly left open such that different programming models can be modeled.

Communication Model

We adopt the model of structuring distributed applications by the means of objects that provide access to encapsulated data by the object interface. In the following, we denote objects with small letters $o, p, q, \ldots$ ... and use the notion of an objectspace (OS) to denote the set of objects. Objects are assumed to reside on computing nodes, this is, objects are not distributed by themselves and do not move between nodes (however, nodes can be mobile). More formally, if we denote by $N = \{n, p, q, \ldots\}$ the set computing nodes, we assume that there is a mapping $LO : OS \rightarrow N$ that uniquely defines the location of an object by assigning a node to each object. Now, we introduce the **communication distance** as the basic means for the describing the communication structure between objects. The goal is to develop a single and handy abstraction that allows capturing various system structuring aspects that determine the communication between the objects (topology, communication protocol used, characteristics of the medium, ...). We focus on the temporal aspects of communication, which constitutes the main idea behind to the following definition:

DEFINITION 1: DIRECT COMMUNICATION DISTANCE

Let $CON \subseteq OS \times OS$ a symmetric relation over the objectspace OS describing the connectivity of the network infrastructure with the semantic that if a pair of objects o and p is part of CON, i.e. $(o, p) \in CON$, then o can communicate directly with p at the abstraction layer provided by network infrastructure (and vice versa). In this case, we say o and p are *connected.* Let furthermore $\Re_0^+$ denote the domain of the positive real numbers (including zero). Then, the direct communication distance between connected objects of an objectspace OS is defined as a function

$$dcd : CON \rightarrow \{\Re_0^+\} \times \{\Re_0^+ \cup \{\infty\}\}$$

with the meaning that if

$$dcd(o, p) = (l, u),$$

then a message sent from o to p at real-time *t*, then it will not arrive a p before real time $t + l$, thus giving a lower bound on the communication delay, and will be handled in p no later than at real time $t + u$, if $u \neq \infty$. In this case, we say that the direct communication distance is *finite*. If $u = \infty$, then no such upper bound exists. For the sake of simplicity, we assume that dcd is commutative and associative. ■

The definition of the direct communication distance generalizes the notion the "temporal uncertainty" as introduced in [2] which describes the general communication characteristics of a distributed real-time system. Since the direct communication distance applies to individual objects, it can be used to describe the individual positioning of the objects in the complete system structure. Note that, unlike the communication cost count that is usually indicated as weight of an edge in connectivity graphs describing multi-hop networks (see for instance [7]), the communication distance can abstract from the actual network topology and describe the communication relation between objects as seen at the programming and application layer.

The communication distance does not only refer to network topology, but reflects also properties and guarantees as provided by the underlying communication protocols. Other important characteristics of a real-time communication protocol, such as the jitter, giving the maximum difference between the minimum and the maximum delay of messages, can be easily derived. Also, similar definitions for multicast communication can be given.

Object Model

Since our final goal is to describe structuring of distributed computations and their temporal behavior, the model must comprise a means to describe the interaction and communication of several objects involved in a distributed computation. To this end, we introduce the notion of a computational path, describing the sequential composition of objects in a distributed computation.

DEFINITION 2: COMPUTATIONAL PATH

Let OS be an objectspace with the connectivity relation CON. A *computational path* CP is

$$CP = (o_{i1}, o_{i2}, \ldots, o_{in}), n \in \mathrm{N}, n \geq 2, o_{i1}, o_{i2}, \ldots, o_{in} \in OS$$

a finite sequence of that are connected in the sequence of their appearance in CP, i.e.

$$\underset{l=1,n-1}{\forall} (o_{il}, o_{il+1}) \in CON$$

We say that the computational path CP *starts* at the object o_{i_1} and *ends* at o_{i_n}. The objects o_{i_1} and o_{i_n} are said to be *connected* by CP.■

The notion of the computational path is the basic means for describing interactions between objects. Note that its definition is independent of a specific programming model and can be used for describing the actual object interactions that implement a specific programming model. In contrast to the path expressions used in [8] to describe global real-time computations, the same object can appear several times within a computational path, thus allowing, for instance, modeling control loops or client/server interactions between two objects.

The notion of the computational path now allows for extending the model to express timing considerations where multiple objects are involved. Since we are most interested in reasoning about the effects of the structure of object interactions (rather than in issues related to local object executions), we will assume that the overall timing behavior of a distributed computation is predominated by the communication costs and that, hence, effects of method executions are only of minor impact and can be subsumed in the communication costs. This can be expressed using the direct communication distance by assuming that choosing appropriate lower and upper bounds for the direct communication distance express timing delays incurred by method executions. These considerations finally lead to the following definition:

DEFINITION 3: COMPUTATIONAL DISTANCE

Let OS be an objectspace with the connectivity relation CON and let $CP-SET$ be the set of computational paths in OS. Then, the computational distance is defined as a function

$$cd : CP-SET \to \{\Re_0^+\} \times \{\Re_0^+ \cup \{\infty\}\}$$

with the meaning that if

$cd(CP) = (l,u)$, with $CP = (o_{i1}, o_{i2}, ..., o_{in})$

then a computation initiated by a message sent from o_{i_1} over CP at real-time *t*, will result in a message arriving at o_{i_n} that does not arrive before real time $t+l$, and that will be handled in o_{i_n} no later than at real time $t+u$, if $u \neq \infty$. In this case, we say that the computational distance is *finite*. If $u = \infty$, then no such upper bound exists. ■

Hence the computational distance between to objects depends on the computational path that is chosen to connect these objects. As the computational path depends on the choice of a distributed programming model, the computational distance can be used in the evaluation of such models. In this sense, it differs significantly from routing cost considerations for wide are networks [7], where the communication costs are determined by the topology of the network infrastructure, only.

Precision Distance

The purpose of this subsection is to introduce the formal means for expressing real-time requirements between without referring to a specific programming model. Informally, the system designer should be able to express consistency relationships between objects that have to be fulfilled under temporal constraints. For instance, in industrial automation, if temperature increases about a critical level at three different places, then a cooling process should start within a certain amount of time. Hence, the model must provide for a formalization of the notions of "relevant events", "consistent reaction", and "staying in a consistent state", even if multiple objects are involved. To this end, we extend the notion of states and observations as defined in [2, 9] to object state variables and object histories that reflect the changes of the variables over the time. More precisely, we assume that each object of the object space contains a number of state variables that reflect the relevant changes of the object state. An object can change the state of a variable to reflect changes in the physical environment, in which case it acts as "rt-entitiy" as explained in [9], or in course of a method execution, which can be triggered either by incoming messages or by the progress of time. We assume that non-relevant or inconsistent intermediate states of a variable do not appear in the model (in practice, further internal variables can used for this purpose when programming an object), such that each state assignment to the object variable represents a relevant event and is considered as an observation in terms of the model of [2, 9]. These considerations lead to the following definition:

DEFINITION 4: OBJECT HISTORIES

Let OS be an objectspace and let $V = \{v_1, v_2, \ldots\}$ be a set of variable identifiers. Without loss of generality, we assume that all variables have values in the same domain VD. The mapping $OV : OS \to POT(V)$ associates set of object variables to each object. All sets of object variables are pair wise disjoint. Then, an *observation*

$$os = (o, v, t, value), o \in OS, v \in OV(o), t \in \Re^+, value \in VD$$

describes that the object o has assigned the value $value$ to its state variable v at real time t, denoted as point of observation. The *history* of an object variable is then defined as:

$$\begin{aligned} HS_v(o) = &(o, v, t_i, value_i), \\ &(o, v, t_{i+1}, value_{i+1}), \\ &(o, v, t_{i+2}, value_{i+2}), \\ &\ldots \end{aligned}$$

with $t_i < t_{i+1} < t_{i+2} < \ldots$

The history of a variable v at the object o is the sequence of observations of that variable where the points of observations are ordered in time. Let $[t_l, t_u], t_l, t_u \in \Re^+, t_l < t_u$ denote a time interval. The *history projection* on an time interval is defined as:

$$HS_v(o)[t_l, t_u] = (o, v, t_i, value_i), \\ (o, v, t_{i+1}, value_{i+1}), \\ \ldots, \\ (o, v, t_{i+n}, value_{i+2})$$

with those observations of v whose points of observations are in $[t_l, t_u]$.■

Based on the notion of the history, we can now introduce the precision distance as means to describe the relationships between objects.

DEFINITION 5: PRECISION DISTANCE

Let OS be an objectspace and let $PG \subset OS, PG = \{o_1, o_2, \ldots, o_n\}, n \in \mathrm{N}$ be a finite subset of objects, denoted as precision group, and let $v_1.v_2, \ldots, v_n$ be variables of $o_1, o_2, \ldots, o_n$, respectively. Let $PCR \subset VD^n$, the **precision consistency relation**, denote a relation that describes the application semantics, i.e., a tuple of variable values $(val_1, val_2, \ldots, val_n)$ is consistent if and only if $(val_1, val_2, \ldots, val_n) \in PCR$. Let $pd \in \Re^+$ denote the length of a time interval. We say, the object histories fulfill PCR with the **precision distance** pd, if and only if:

$$\underset{t_0 \in \Re^+}{\forall} \underset{o_j \in PG}{\forall} \exists (o_l, v_l, t_{i_l}, val_{i_l}) \in HS_{v_l}(o_l)[t_0, t_0 + pd]:$$
$$(val_{i_1}, val_{i_2}, \ldots, val_{i_n}) \in PCR$$

this is, for each point in time t_0, there are observations in the time interval $[t_0, t_0 + pd]$ in the histories of the object variables such that the variable values fulfill the precision consistency relation PCR.■

3. Industrial application example

In this section, the formal notions introduced in section 2 are applied to a (simple) industrial automation example. In the scenario described in Figure 1, let object q on the PLC encapsulate some sensor data from a machine and let object p on the controller node reflect the status of q, which is finally displayed by the operator object o. Then, the direct communication distance between p and q, and between between q and o, are determined by and capture the communication characteristics of the field-bus, and the local are network, respectively.

Lets now consider the programming structure. Nested RPC calls from o to p to q are modeled as $CP - RPC = (o, p, q, p, o)$. A program based on publisher/subscriber communication would be modeled by the computational path $CP - PS = (q, p, ec, o)$, reflecting that the controller object p receives messages containing events from the object q, and publishes these events by sending messages to (an additional) event channel object ec, which finally forwards them to the controller object o.

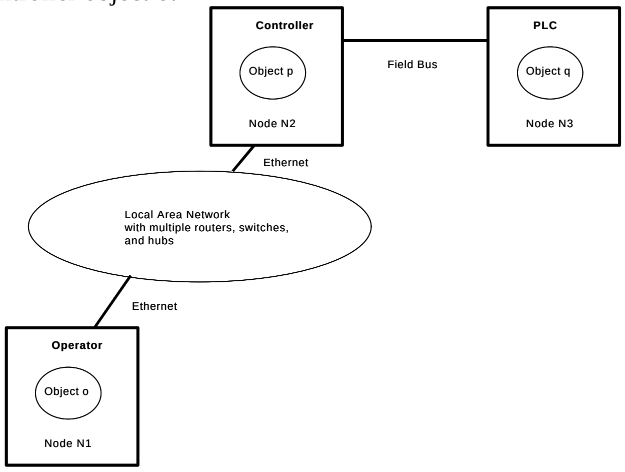

Figure 1: Example for the direct communication distance function.

Although the existence of the event channel object is hidden from the programmer by the implementation of the of the distributed publisher/subscriber programming model, its existence and the structural effects of that model are properly reflected in the formal model using the notion of the computational path. Finally, the temporal gap between the sensor object and the operator object is modeled with the precision distance using the "identity" relation as precision consistency relation on the respective variables.

References

[1] M. Mock, "An Architecture Supporting Loose and Close Cooperation of Distributed Autonomous Systems," presented at 4th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2001), Magdeburg, Germany, 2001.

[2] H. Kopetz and K. H. Kim, "Temporal Uncertainties in Interactions among Real-Time Objects," presented at 9th IEEE Symposium on Reliable Distributed Systems, Huntsville, AL, 1990.

[3] H. F. Korth, N. Soparkar, and A. Silberschatz, "Triggered Real-Time Databases with Consistency Constraints," presented at 16th Conference on Very Large Database Systems, Brisbane, Australia, 1990.

[4] H. R. Callison, "A Time-Sensitive Object Model for Real-Time Systems," *ACM Transactions on Software Engineering and Methodology*, vol. 4, pp. 287-317, 1995.

[5] M. Törngren, "Fundamentals of Implementing Real-Time Control Applications in Distributed Computer Systems," *Real Time Systems*, vol. 14, pp. 219-250, 1998.

[6] B. P. Douglass, *Doing Hard Time*. Reading, MA: Addison-Wesley, 1999.

[7] A. Tannenbaum, *Computer Networks*: Prentice Hall, 1981.

[8] L. R. Welch, B. Ravindram, B. A. Shirazi, and C. Bruggemann, "Specification and Modeling of Dynamic, Distributed Real-Time Systems," presented at 19th IEEE Real-Time Systems Symposium, Madrid, Spain, 1998.

[9] H. Kopetz, *Real-Time Systems*. Boston: Kluwer Academic Publishers, 1997.