Data Streams – an Analysis of the Interactions Between Real-Time Tasks

Mário J.B. Calha
EST - Instituto Politécnico de Castelo Branco
Av. do Empresário
6000-767 Castelo Branco, Portugal
mjc@est.ipcb.pt

José A.G. Fonseca
DET / IEETA - Universidade de Aveiro
Campus Universitário de Santiago
3810-193 Aveiro, Portugal
jaf@det.ua.pt

Abstract

In distributed systems the communication paradigm used for intertask interaction is the message exchange. Several approaches have been proposed that allow the specification of the data flow between tasks, but in real-time systems a more accurate definition of these data flows is mandatory. Namely, the specification of the required tasks' and messages' parameters and the derivation of the unspecified parameters have to be possible. Such an approach could allow an automatic scheduling and dispatching of tasks and messages. The data streams present a possible approach to the holistic scheduling and dispatching in real-time distributed systems where different types of analysis that correlate the various parameters are done. The results can be used to fulfil different goals like the level of buffering that is required at each node of the distributed system.

1. Introduction

The design of real-time software must incorporate all of the fundamental concepts associated with high-quality software like abstraction and modularity. In addition, real-time software poses a set of unique problems for the designer like:

- Representation of interrupts and context switching,
- Concurrency as manifested by multitasking and multiprocessing,
- Intertask communication and synchronization,
- Wide variations in data and communication rates,
- Representation of timing constraints,
- Asynchronous processing,
- Necessary and unavoidable coupling with operating systems, hardware, and other external system elements.

Several real-time software design methodologies have been proposed to grapple with some or all of the problems noted above. Some design methods extend one of the three classes of design, namely: data flow [6], data structure [2], or object-oriented [7] methodologies. Others introduce an entirely separate approach, using finite state machine models or message passing systems [5], Petri nets [1], or a specialized language [3] as a basis.

More recently the use of time Petri nets in real-time systems is proposed in [9], [10], [11] and [14].

Data flow-oriented design methods are the most widely used in the industry. Data-flow models are used to show how data flows through a sequence of processing steps. The data is transformed at each step before moving on to the next stage. These processing steps or transformations are program functions when data-flow diagrams are used to document a software design.

Extensions to data flow representations that provide the mechanics for real-time software design have been proposed. Hassan Gomaa [4] proposed one extension called Design Method for Real-Time Systems (DARTS). This extension allows real-time system designers to adapt data flow techniques to the special needs of real-time applications.

Another data flow-oriented method is based on simple data streams. A data stream shows the information flow between tasks. The tasks follow the producer-consumer model. Data flows from one producer task to one or more consumer tasks. Consumer tasks may also produce data to other tasks thus becoming consumer/producer tasks. This way, a data stream represents only the tasks and data that make use of critical resources like the processing units and the network. This method differs from other approaches, like the Petri nets and other behavioural diagrams, mainly in the way the transitions between tasks are specified. These transitions don't specify the triggering conditions but follow a simpler approach more suited to distributed control systems. In this approach only the possible sets of messages to be transmitted has to be specified. This way, the internal computational aspects of tasks remain encapsulated.

The result of the proposed approach is a set of data streams, where tasks and messages have some parameters derived. These parameters will make possible a holistic system scheduling of all entities. Also, the data stream analysis is shown to be beneficial in different phases of a distributed system planning.

0-7803-9402-X/05/$20.00 © 2005 IEEE

This paper is organized in five further sections. In section 2, various scenarios of task interaction are considered. Section 3 presents the data streams. The algorithms to build the data streams are explained in section 4. Applications of the data streams are suggested in section 5, together with a technique to derive the necessary level of message buffering for all nodes. Finally, some conclusions and further work are discussed in the last section.

2. Interaction between tasks

The most basic form of intertask interaction in distributed systems is message exchange. This enables a sending task, producer task, to transmit a single message to a receiving task, consumer task. This is the communication paradigm used for intertask interaction. Message passing between a pair of tasks can be supported by two message communication operations: *send* and *receive*, defined in terms of destinations and messages. In order for one task to communicate with another, one task sends a message (a sequence of bytes) to a destination and another task at the destination receives the message. This activity involves the communication of data from the sending task to the receiving task and may involve the synchronization of the two tasks.

Due to the interaction between tasks, several scenarios may occur in an ordinary system. A common restriction to all scenarios is that any message consumed has to be available in the beginning of the task execution and any produced message is only available after the end of the task execution. Several scenarios were identified. These were arranged in two groups: the basic scenarios and the expansion scenarios. The basic scenarios represent the simplest form of interactions where the messages to be transmitted are defined in the diagrams. These scenarios comprise both unicast and multicast communication. The expansion scenarios are built upon the basic scenarios, where the set of possible messages to be transmitted is defined. The actual messages to be transmitted are only defined during system functioning. A set of four basic scenarios and a set of eight expansion scenarios are now presented.

2.1. Basic scenarios

The first two scenarios refer to unicast and are depicted in Figure 1.



Figure 1 – Unicast of a single message or multiple messages

The basic interaction between tasks is shown in Figure 1-a). In this scenario, task T_0 produces message M to task T_1.

A producer task can unicast several messages, M_1 to M_n, to another task as shown in Figure 1-b). If all messages have the same properties, apart from the message size, than this scenario is just an extension of the previous one. The total transmission time is the cumulative transmission time of each message.

Figure 2 depicts two scenarios that refer to multicast.

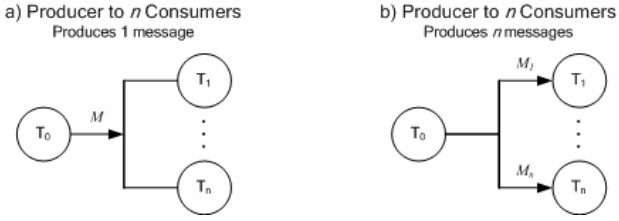


Figure 2 – Multicast of a single message or multiple messages

Multicasting of a single message to several tasks is shown in Figure 2-a). The combination of the multicast property with the possibility of sending several messages results in the situation shown in Figure 2-b). This scenario makes possible sending different messages to different tasks. Again, the transmission time is the cumulative transmission time of each message.

2.2. Expansion scenarios

The expansion scenarios comprise four unicast scenarios and four multicast scenarios.

The first two unicast scenarios, which show the transmission of a message from a set of messages, are depicted in Figure 3 and the other two unicast scenarios are depicted in Figure 4.



Figure 3 – Unicast of a single message from a set of messages

Figure 3-a) shows task T_0 producing one message to Task T_1. This message is one of a set of possible messages to be transmitted. In this scenario it is reasonable to demand that all messages have the same properties apart from the message size. This way the transmission time is calculated using the message with the largest size. The possibility of not transmitting any message is considered in Figure 3-b).

For the definition of these scenarios, it is only required the definition of the possible sets of messages and their properties. The computational aspects of the involved tasks, which lead to each message set, do not need to be considered. Therefore, at this level of abstraction, the transitions don't have associated

conditions that need to be met, because the question is not "*What conditions lead to the production of each set of messages?*", but simply "*What sets of messages can be produced?*". Therefore this approach leads to the encapsulation of tasks reducing the complexity of the analysis.

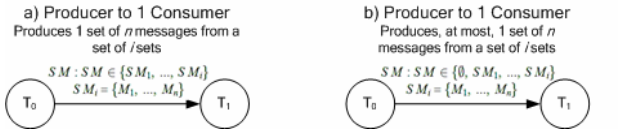


Figure 4 – Unicast of a single message set from a set of message sets

Figure 4-a) shows the unicast transmission of a set of messages, *SM*, where task T_0 produces one set of messages to Task T_1. This set of messages is selected during system functioning from the possible sets of messages.

The possibility of not transmitting any set of messages is considered in Figure 4-b).

The first two multicast scenarios are depicted in Figure 5 and the other two multicast scenarios are depicted in Figure 6.

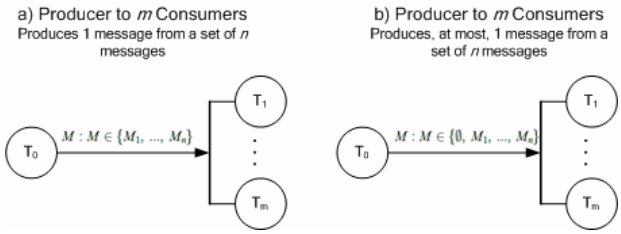


Figure 5 – Multicast of a single message from a set of messages

Figure 5-a) shows the situation of a multicast of a single message *M* from a set of messages, $M_1 \ldots M_m$. Only one of these messages is transmitted after each execution of task T_0. This scenario is basically the union of the scenarios depicted in Figure 2-a) and Figure 3-a).

On the other hand, Figure 5-b) considers the possibility of the task either transmit a message from the set of messages, $M_1 \ldots M_m$ or not transmit a message at all. Because this scenario has a non-periodic produced message it should be considered as asynchronous communication.

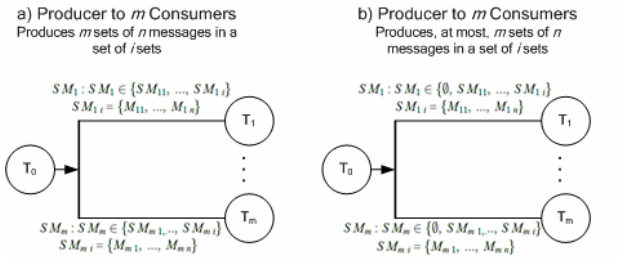


Figure 6 – Multicast of multiple message sets from a set of message sets

Figure 6-a) shows the multicast of multiple sets of messages from a set of message sets. On the other hand, Figure 6-b) considers the possibility of the task not transmit a set of messages.

With these scenarios many other can be constructed.

3. Data streams

A data stream may begin with a producer task and end with a consumer task. According to the scenario, various data streams might be identified. For example the scenario in Figure 7 depicts a task that produces four messages. Message M_1 is consumed by two different consumer tasks, T_1 and T_2, message M_2 is consumed by task T_3, message M_3 is consumed by task T_4 and message M_4 is consumed by task T_5. From this scenario, five data streams are identified as shown in Figure 7.

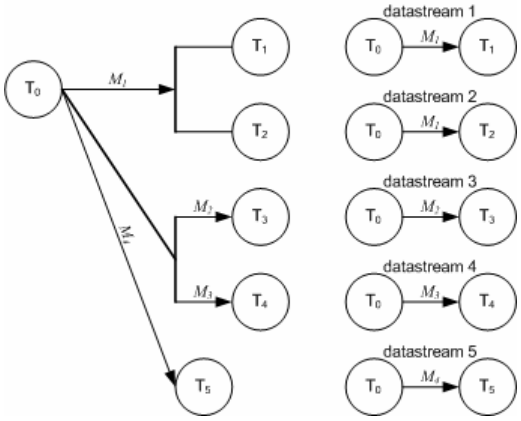


Figure 7 – Example of data stream identification

In a broader view of the data flows, a data stream can also begin with a message, or even sets of messages, and end with a message, or even sets of messages.

3.1. Data stream triggering

From the point of view of the entity, data streams can be triggered either with a task or a message. While from the point of view of time, these possibilities can be further divided into synchronous or asynchronous triggering.

A task triggering is accomplished by a producer task. On the other hand, a message triggering is accomplished by a message that is produced through unspecified means. This message can be produced by a task not controlled by the data stream analysis, or it can come from an external node, or it can result from an event generated at some node.

A synchronous task, or message, triggering is accomplished through a periodic task, or message. While an asynchronous task, or message, triggering is accomplished by an aperiodic task, or message. If aperiodic triggering can be transformed into a sporadic triggering using a sporadic server [8] then a minimum

inter-arrival time (MIT) can be defined. If the MIT is less then the period of the data stream then some kind of buffering technique might be needed.

3.2. Task role changing

The client-server model has two protocols that are the request-reply and the request-reply-confirmation. These protocols involve two tasks, T_0 and T_1. The request-reply involves the exchange of two messages while the latest involves the exchange of three messages.

Task T_0 begins as a producer task and then changes to a consumer task, or a consumer/producer task, according to the protocol. This means that T_0 has two functional parts. In the first part it behaves like a producer task and in the last part it behaves as either a consumer task or a consumer/producer task. The correspondent data streams are shown in Figure 8.

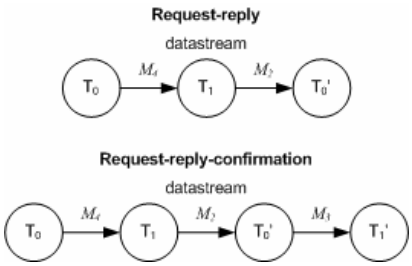


Figure 8 – Data streams of the protocols used in the client-server model

From the data streams it is clear that T_0 and T_0' are two different tasks. The same happens with T_1 and T_1'.

3.3. Sporadic data streams

The previous rationales are all based on a data stream that is executed on a periodic basis. But in cases where the starting entity (task or message) is started asynchronously, as a sporadic entity, similar approaches can be used. The minimum interarrival time (MIT) must be at least equal to the period of the data stream.

Usually, if there is an asynchronous event, this event is treated by a well defined procedure that may involve various tasks. That is, apart from the first event, the remaining entities are executed, or transmitted, according to a known pattern. It is enough to have a minimum inter-arrival time of the asynchronous event that is never smaller than the period of the data stream. Considering this constraint, the remaining parameters evaluation and data stream analysis can be done as in the case of a periodic data stream.

4. Building the data streams

A possible algorithm for data stream construction is done in four steps. The first step is checking the parameters of the tasks and the messages. Then the data streams are created. The next step is the verification of the period and deadline of every entity in the various data streams. If there are no contradictory values, like having two entities in the same data stream with different periods, the remaining parameters are determined in the final step.

The main steps of this algorithm will be further discussed in the following sub-sections.

4.1. Creating the data streams

A data stream represents a single stream of information that is extracted from the graphs that represent the interaction between tasks. A conceptual structure to store the data streams is shown in Figure 9.

This structure is mostly made of linked lists. The top linked list stores a data stream per position. Then, each node of a data stream stores a task, *PTask*, and a linked list of produced messages, *LProd*. *LProd* is a linked list of produced message sets, *AND Produced Sets*. Each of these sets can have alternative sets of produced messages, *OR Produced Sets*. In the end, the message sets that are produced are equal to the number of positions of the AND Produced Sets, because only one of each OR Produced Sets is actually transmitted.

This way, this data streams structure supports the scenarios previously presented.

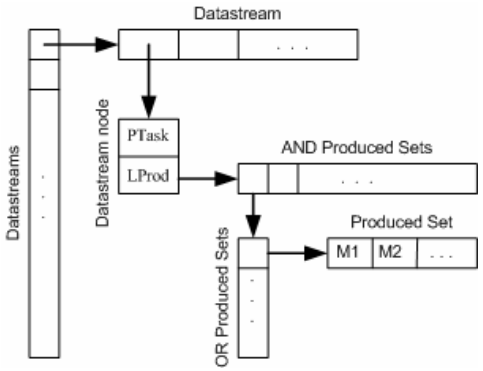


Figure 9 – Structure to store data streams

The extraction of the data streams is an iterative process. The main procedure of this process, *FindDataStreams*, searches the task list looking for producer tasks and also looking for non-produced messages. These two types of entities mark the beginning of a new data stream. Then, for each data stream the iterative procedure, *FindPrecedences*, follows the chain of interaction between tasks, creating new data streams where appropriate.

4.2. Filling the data streams

This procedure tries to define the period, the initial phases and the deadlines of the tasks and messages on each data stream.

It is easily understood that the period of the entities in a data stream has to be the same. On the other hand, the values of the initial phases and the deadlines depend on the level of coexistence, defined by the system designer, between the execution windows and the transmission

windows, being a window the interval that begins in the release instant of the entity and ends in its relative deadline. If the designer chooses to have a high level of coexistence between the windows then this will lead to the following set of equations, where:

Task refers to the consumer and/or producer task;

MsgP refers to the message produced;

MsgC refers to the message consumed;

Ph_{Task}, Ph_{MsgP} and Ph_{MsgC} represent the initial phases;

C_{Task}, C_{MsgP} and C_{MsgC} represent the execution/transmission times;

D_{Task}, D_{MsgP} and D_{MsgC} represent the relative deadlines;

It should be noticed that the available time (for message transmission and for processing at the nodes) is divided in fixed duration time intervals called elementary cycles (ECs). The EC constitutes the system resolution. Events happening in the same EC don't have precedence relationships, i.e., may be viewed as simultaneous. The duration of an EC is referred as t_{EC}.

- Producer tasks

$$Ph_{Task} = 0$$

$$\left\lceil \frac{C_{Task}}{t_{EC}} \right\rceil \times t_{EC} \le D_{Task} \le \left\lceil \frac{\sum_{i=0}^{n} C_{Task_i}}{t_{EC}} \right\rceil \times t_{EC}$$

$$Ph_{MsgP} = Ph_{Task} + \left\lceil \frac{C_{Task}}{t_{EC}} \right\rceil \times t_{EC}$$

$$D_{Task} + \left\lceil \frac{C_{MsgP}}{t_{EC}} \right\rceil \le D_{MsgP} \le D_{Task} + \left\lceil \frac{\sum_{i=0}^{n} C_{MsgP_i}}{t_{EC}} \right\rceil \times t_{EC}$$

Where *n* in the second equation refers to the number of tasks and in the fourth equation refers to the number of messages.

These equations show the values of the initial values and the bounds for the deadlines. The higher bound for a deadline is dependent upon the execution time of the tasks in a node, or dependent upon the transmission times of all messages.

- Consumer tasks

$$Ph_{Task} = Ph_{MsgC} + \left\lceil \frac{C_{MsgC}}{t_{EC}} \right\rceil \times t_{EC}$$

$$D_{MsgC} + \left\lceil \frac{C_{Task}}{t_{EC}} \right\rceil \le D_{Task} \le D_{MsgC} + \left\lceil \frac{\sum_{i=0}^{n} C_{Task_i}}{t_{EC}} \right\rceil \times t_{EC}$$

Other approaches were already proposed in [12] and [15]. These approaches consider that the execution windows and the transmission windows do not coexist.

Following this methodology, the various parameters can be defined for a scheduling attempt. After a successful scheduling, the parameters can be considered conservative and adjustments can be made. If, on the contrary, the scheduling fails then some parameters will have to be relaxed.

4.3. Interdependent data streams

Interdependency occurs when two, or more, data streams are related due to a common task in their nodes. This interdependence between data streams will create an extra constraint upon the initial phases of its tasks and messages. The period of the entities in interdependent data streams has to be the same.

In order to consider the interdependencies between data streams, the initial phases of each task that participates in different data streams has to be adjusted to the largest value defined by each data stream. Considering a task that participates in n data streams:

- $Ph = MAX(Ph_i)$, i = 0, … , n

This way, a task is only executed when all consumed messages are ready. The messages that arrived earlier must be kept in buffers until the consumer task is ready to consume them.

The final tasks' and messages' parameters can only be known after considering every interdependent data stream.

5. Applications of the data stream analysis

The data stream analysis is beneficial in different phases of a distributed system planning. This analysis is useful in situations like: computational and network load evaluation, early stage system parameters definition, real-time procedures, when the various entities don't have specific deadlines; systems with a high network load, where the messages have strict deadlines; systems with a high node load, where the tasks have strict deadlines.

At an early stage where very few parameters are defined, apart from the execution times and message size, this analysis helps in the definition of suitable values for the initial phases and the deadlines. After this, the system developer can fine tune any parameter according to some special needs. If, on the contrary, almost every parameter is clearly defined, then this analysis is useful to check the assumptions.

A real-time procedure is implemented by a set of tasks that communicate through message passing. For instance in a closed-loop control of the temperature of a room, a sensor generates regular samples of the room temperature and a control system decides to act when the temperature reaches a certain threshold level. A deadline might be evaluated so that a dangerous temperature level is never reached. Now, to acquire the temperature samples and to implement the closed-loop control system several tasks are needed. These tasks could be

spread across different nodes. The challenge here is what should be the tasks' and messages' parameters, namely initial phase and deadline, so that, at least, the primary objective concerning the deadline of the procedure is met.

Secondary objectives, like loosing the constraints of more constrained resources or extra gains from trying to finish the procedure earlier than necessary, should also be taken into account. For instance, these secondary objectives might be formulated towards a simpler admission of new tasks. The admission of new tasks might involve reviewing the data streams in order to accommodate the extra node and network load.

Also the data streams can be useful in the definition of the necessary message buffering. The level of buffering required for each producer task to store messages in the time between their production and their consumption can be determined.

6. Conclusions and future work

In this paper, a possible approach to the holistic scheduling and dispatching in real-time systems was presented. In this approach, data streams that give support to different types of analysis, which correlate the various parameters, have been done.

Considering previous works by the authors, with the basic and expanded scenarios, new types of data streams are now possible. This led to a significant enhancement in what concerns possible interactions between tasks. The notation was extended in order to accommodate the new possibilities. The impact of this diversity in task interaction was considered and several applications were indicated.

The data streams based approach is valuable in helping to achieve a schedulable holistic system where timeliness is a must. The new possible scenarios, together with the buffering technique, are now being integrated into the simulator SIMHOL. This simulator supports the joint scheduling and dispatching of messages and tasks and was presented in [13].

References

[1] Vidondo, F., "GALILEO: Design Language for Real-Time Systems", Proc. ITT Conf. on Programming Productivity and Quality, ITT Corporation, pp. 198-210, June 1983.

[2] Jackson, M., "System Development", Van Nostrand Reinhold, 1983.

[3] Steusloff, H. U., "Advanced Real-Time Languages for Distributed Industrial Process Control", Computer, vol. 17, no. 2, pp. 37-46, February 1984.

[4] Gomaa, H., "A Software Design Method for Real Time Systems", CACM, vol. 27, no. 9, pp. 938-949, September 1984.

[5] Witt, B. I., "Communication Modules: A Software Design Model for Concurrent Distributed Systems", IEEE Computer, vol. 18, no. 1, pp. 67-77, January 1985.

[6] Ward, P. T., and S. J. Mellor, "Structured Development for Real-Time Systems", 3 volumes, Yourdon Press, 1985, 1986.

[7] Booch, G., "Software Engineering with Ada", 2d ed., Benjamin-Cummings, 1987.

[8] B. Sprunt, L. Sha, and J. Lehoczky. "Aperiodic task scheduling for hard real-time systems", *Journal of Real-Time Systems*, 1(1):27-60, 1989

[9] Okawa Y. , Yoneda T., "Schedulability verification of real-time systems with extended time Petri nets", *International Journal of Mini and Microcomputers*, vol. 18, n° 3, 1996, p. 148-156.

[10] Toussaint, J., F. Simonot-Lion and Jean-Pierre Thomesse (1997). Time constraint verifications methods based time petri nets. In: 6th Workshop on Future Trends in Distributed Computing Systems (FT-DCS'97). Tunis, Tunisia. pp. 262–267.

[11] Roux, O. H. and A.-M. Déplanche (2002). A t-time petri net extension for real time-task scheduling modeling. European Journal of Automation (JESA).

[12] Calha, M.J., J.A. Fonseca, "Adapting FTT-CAN for the joint dispatching of tasks and messages", Proceedings of the 4th IEEE International Workshop on Factory Communication Systems (WFCS'02), Vesteras, Sweden, August 2002.

[13] Calha, M.J., J.A. Fonseca, "SIMHOL – A graphical simulator for the joint scheduling of messages and tasks in distributed embedded systems", Proceedings of the 5th IFAC International Conference on Fieldbus Systems and their Applications (FeT 2003), Aveiro, Portugal, July 2003.

[14] Barreto, R., S. Cavalcante, P. Maciel, "A Time Petri Net Approach for Finding Pre-Runtime Schedules in Embedded Hard Real-Time Systems", 24th International Conference on Distributed Computing Systems Workshop (ICDCSW'04), Tokyo, Japan, March 2004.

[15] Calha, M.J., J.A. Fonseca, "Approaches to the FTT-based scheduling of tasks and messages", Proceedings of the 5th IEEE International Workshop on Factory Communication Systems (WFCS'04), Vienna, Austria, September 2004.