Integration of a flexible network in a resource contracting framework*

R. Marau, L. Almeida, P. Pedreiras
DETI/IEETA
Universidade de Aveiro, 3810-193 Aveiro, Portugal
{marau,lda,pedreiras}@det.ua.pt

M. González Harbour, D. Sangorrín, J. L. Medina†
Grupo de Computadores y Tiempo Real
Universidad de Cantabria, 39005 - Santander, Spain
{mgh,daniel.sangorrin,medinajl}@unican.es

***Abstract*— In this paper we overview the integration of a framework that generically manages the system resources in the form of contracts, namely the FRESCOR framework, with a flexible network resource. We describe how a network resource, namely FTT-SE, supports the FRESCOR framework services and, likewise, how the network services are made available to the application through the contracting framework.**

I. Introduction

Networked Embedded Systems (NES) were originally associated with industrial supervision and control applications, which employed simple sensors, actuators and controllers. However, a steep evolution in this application domain is being experienced, pushed by the growing number of sensors and overall complexity present at the plant level. As an example, the use of imaging sensors, both for supervision and control purposes, is spreading widely in classes of applications such as mobile robotics, traffic control and assembly lines inspection. Consequently, the sensors become inherently more complex, so as the flows of information exchanged at the cell and plant levels, integrating periodic and aperiodic flows of short and large data, some of multimedia nature, with considerable variability during run-time.

The new demands and increased complexity posed by these applications pushed the development on new techniques and design methodologies. Two key aspects in this regard are the complexity management and the resource management. Complexity management is being addressed by the adoption of adequate middleware layers in NES (e.g. CORBA and Java RMI, DCOM, etc) [1], which facilitate distribution, aiming at transparent interaction mechanisms between objects, components or applications. Regarding the resource management (e.g. CPU, memory, network, energy, etc) several approaches have been proposed recently, aiming at fulfilling the needs of those emerging applications in aspects like dynamic configuration and QoS management, support for new and more efficient scheduling techniques, etc.

* This work has been funded in part by the Plan Nacional de I+D+I of the Spanish Government under grant TIC2005-08665-C03 (THREAD project), and by the European Union's Sixth Framework Programme under contracts FP6/2005/IST/5-034026 (FRESCOR project) and IST-004527 (ARTIST2 NoE). This work reflects only the author's views; the EU is not liable for any use that may be made of the information contained herein.

† In a Post-doctoral internship in CEA LIST, Laboratoire des Logiciels pour la Sûreté des Procédés, Boîte 94, F-91191, Gif-sur-Yvette, France.

The FIRST Scheduling Framework (FSF) [2] provides a high-level abstraction for real time resource schedulers while maintaining predictability and performance efficiency. It provides a homogeneous interface so it can be used in different platform architectures. This framework was initially designed to cope with the application needs for processor and network management, although with some limitations in the latter. The Framework for Real-time Embedded Systems based on COntRACTS (FRESCOR) aims at extending the FSF framework for multi-resource reservation, comprising various classes of resources commonly found in NES applications.

Regarding the communication subsystem, the recently proposed Flexible Time triggered communication over Switched Ethernet (FTT-SE) [3] provides flexible and deterministic real-time communication services combined with dynamic Quality-of-Service (QoS) management. This protocol has been developed specifically to address the requirements presented by the emerging applications referred above, combining realtime requirements with a high degree of adaptability. It looks, thus, a natural network candidate for inclusion in a contracting framework, to efficiently exploit and enrich the high level of flexibility that it already offers.

This paper analyzes the integration of the FTT-SE protocol in the scope of the contract model framework and describes how this integration can be performed. Using the former FSF framework, a network resource implementation exists for the Real-Time Ethernet Protocol (RT-EP) [4]. However, for FTT-SE, a different architecture must be used due to the different data link layer features of the two network protocols.

The remainder of the paper presents an overview of FRESCOR and FTT-SE background in sections II and III, discusses the integration of FTT-SE under FRESCOR in section IV, details the contracting procedure in section V and shows conclusions and on-going work in section VI.

II. FRESCOR background

The FRESCOR framework is based on the notion of contracts between the application and the system resources manager. These contracts are created, managed and enforced by a *Contract Layer*, which assures that sufficient resources capacity is available. The framework is divided in modules that allow abstracting away the specificities of the resources typically found in NES. Of particular interest to this work are the

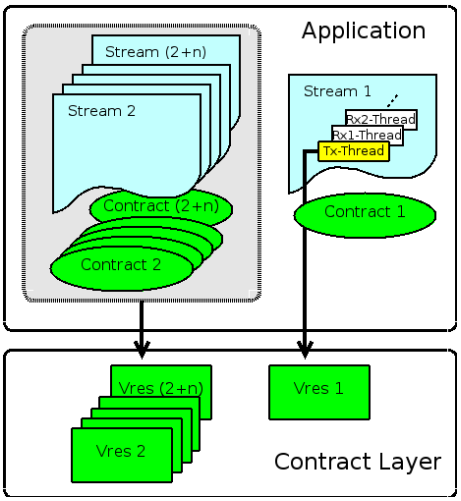

Fig. 1. Contract layer

Core module, which contains the basic contract information that must exist in all contracted resources, the *Spare capacity* module, which defines how the application may take advantage of currently unused resource capacity, and the *Distribution* module that deals with issues of distributed applications.

The contract parameters associated to these modules are referred in table I. The *Contract id* is a unique identifier inside one resource (here a network resource) that distinguishes the contracts globally, the *Resource type* and the *Resource id* inform about the kind of, and which resource the contract refers to; the *Minimum budget* and *Maximum period* define the minimum resource capacity required by the application, *Importance* and *Weight* allow prioritizing the contracts associated to one resource when distributing spare bandwidth.

TABLE I
CONTRACT PARAMETERS

Core	Contract id Resource type Resource id Minimum budget Maximum period Deadline
Spare capacity	Granularity Maximum budget Minimum period Utilization set Importance & Weight Stability
Distribution	Protocol dependent information

A. FRESCOR application model

Within FRESCOR, the application is a global entity enclosing several *Threads* that access the system resources by means of *Virtual resources (Vres)* residing in the *Contract Layer*. Each *Vres* holds one associated contract. In distributed applications, this layer is also distributed and comprises the management of both, the processors and the network resources. Several network contracts may be atomically negotiated as a group, ensuring that they are either all accepted, or all rejected as a whole. The contract negotiation for a given stream is initiated at the sending node. If successful, the associated *Vres* may be created in that node, or in some other node that may be in charge of scheduling the network traffic.

In a distributed application, the FRESCOR framework considers the network as just another resource that is managed by contracts. Each of these contracts refers to one Stream through which application threads exchange messages. Each stream has a unique identifier mapped on the *Contract id* and used as a stream descriptor by the application when accessing the *Contract Layer*. Figure 1 shows the FRESCOR distributed application model with a network resource highlighting just the network contracts under two possible situations: when a contract *Virtual resource* is created in a transmitter node (Stream 1), or in a contract group situation (Streams 2 .. (2+n)).

III. FTT-SE BASICS

The FTT-SE protocol was designed to support hard real time applications using Switched Ethernet networks in a flexible but predictable manner. It supports both time-triggered and event-triggered communication semantics in two well defined and temporally isolated communication subsystems, namely the Synchronous and the Asynchronous Messaging Systems, SMS and AMS, respectively [3].

The protocol is based on a master-slave paradigm, in which a master controls the access to the network by the remaining nodes in the system (slaves). This allows managing the communication load submitted to the switch at each instant, thus preventing overloads and maintaining a predictable behavior.

However, the master-slave control is carried out on a cyclic basis, i.e., the master sends out one control message per cycle, only, indicating which messages must be transmitted therein. The cycle is called Elementary Cycle (EC) and it is triggered by the master control message called Trigger Message (TM), which is broadcast to all nodes.

The master holds the System Requirements DataBase (SRDB) that contains, among other information, the current communication requirements, and builds the traffic schedules for each EC on-line. Changes to these requirements can be carried out at run-time and are subject to an admission control that guarantees continued timely communication.

The requirements tables for synchronous and asynchronous message streams hold the following parameters, respectively:

$$SM_i\left(C_i, D_i, T_i, O_i, S_i, \{R_i^1..R_i^{k_i}\}\right) \quad , i = 1..N$$
$$AM_i\left(C_i, D_i, mit_i, S_i, \{R_i^1..R_i^{k_i}\}\right) \quad , i = 1..N$$

C_i is message i transmission time, D_i is its deadline, T_i the period, mit_i is the minimmum inter-transmission time and O_i the offset. Both D_i, T_i/mit_i and O_i are expressed as integer numbers of ECs. S_i is the sender node and $\{R_i^1..R_i^{k_i}\}$ is the set of k_i receivers for this message stream.

Finally, FTT-SE also provides mechanisms to synchronize the application threads with their periodic communications. This synchronization plus the use of offsets is the basis for the so-called network-centric approach to the design of distributed systems, which facilitates the synchronization of threads in different nodes and the reduction of end-to-end delays.

IV. Integration of FTT-SE under FRESCOR

One important goal of the integration was to keep the performance level of the FTT-SE real-time communication services throughout the abstraction process associated with the creation of a middleware. The FRESCOR framework was selected as middleware because it facilitates achieving this goal given its simple and generic application interface and real time concerns. Moreover, its modular flexibility extends the resource management to an holistic application perspective which reduces the project design complexity.

This section describes how the FTT-SE protocol can be integrated as a FRESCOR pluggable resource. This integration allows abstracting away the network access from the application perspective and it defines two sets of services, the negotiation procedure and the communication access primitives. The former handles the contract (re-)negotiations requested by the application to change the Stream properties provided by the network resource. Once a contract is accepted the application may start using the respective communication Stream through the services provided by the latter.

As referred before, the FRESCOR modules used when integrating FTT-SE are the *Core*, *Spare capacity* and *Distribution* modules. Each of these takes its role in the contract negotiation with the parameters described in table I. The *Distribution* module needs specific attention since it contains network protocol dependent information. For the RT-EP distributed resource no special parameters were required, thus the *Core* parameters *Minimum period* and *Maximum Budget* were enough to carry out the network management. However, FTT-SE includes several features that require appropriate configuration and management, which must thus be included in this module:

- Two communication triggering models are provided by the network, namely time-driven and event-driven, which must be defined in the contract specification;
- To take advantage of the multiple forwarding paths in the network switch and still provide real-time guarantees, the contract must include the specific switching path used by each channel, i.e., the identification of the producer and consumers involved and the switch ports they are connected to;
- To exploit the explicit synchronization between time-driven channels supported by the network, the contracts, or contract groups, must include two additional parameters, one describing the *Contract id* of the channel to synchronize with, and another specifying the desired synchronization offset. If no synchronization is specified the channel is considered as float and the network will arbitrarily allocate relative offsets to the contract;

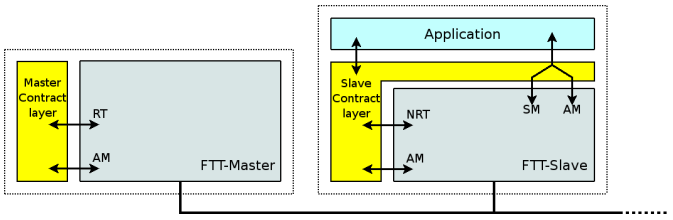

Fig. 2. Architecture overview

The integrated FRESCOR / FTT-SE architecture is sketched in Fig. 2. The network contract negotiation procedure is centralized in the FTT master node and it is handled by the *Master Contract Layer*. This is a natural choice since the FTT master centralizes all the real time requirements of current communication channels and controls the network access. The contracts are then reflected on the involved slave nodes. The *Slave Contract Layer* handles network contract requests, holds local contract copies and makes them available to the application threads. This centralized approach is substantially different from the one taken in the RT-EP implementation, where the negotiation procedure is fully distributed requiring every node to keep a consistent replica of all running contracts.

A. The FRESCOR / FTT-SE interface

The communication between the *Master Contract Layer* and the *Slaves Contract Layer*, both for conveying negotiation requests and publishing the contract copies, uses permanent bi-directional channels between the master and each slave node in the network, implemented with FTT-SE asynchronous messages (AM).

The network contracts in the *Master Contract Layer* are reflected in the FTT Master, in its Requirements Table (RT). On the other side, the *Slave Contract Layer* keeps the copies of its contracts, reflecting them in the respective FTT Slave, in the Node Requirements Table (NRT). This layer also provides interfaces to the application, to negotiate contracts and to access the communication services, both synchronous (SM) and asynchronous (AM).

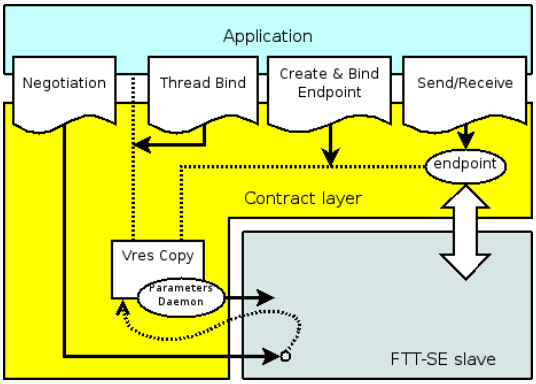

Fig. 3. FRESCOR interface for network contracts

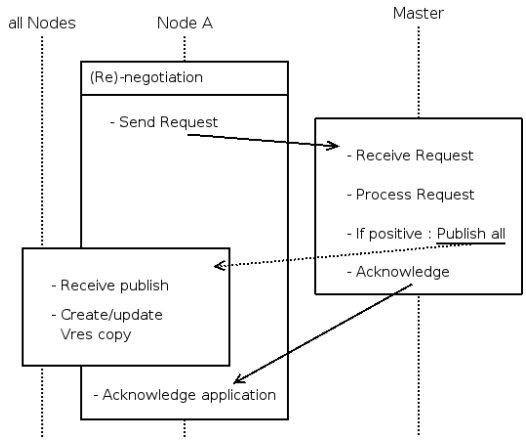

Fig. 4. Negotiation steps

B. Supporting the application interface

Figure 3 shows the FRESCOR common resource interface and the objects involved in network contracts. The *Negotiation* service allows the system to establish the required resource reservations and, in this case, involves communication with the *Master Contract Layer*. Upon a negotiation success, the respective contract *Virtual resource(s)* is created/updated in the *Master Contract Layer* and the *Virtual resource copies* are created/updated in the *Slaves Contract Layer*. The *Thread Bind* allows associating an application thread with a contracted resource and provides access to the respective *Vres copy*. The access to the contracted resource, a Stream in this case, is made through an *endpoint*, which is created and bound to the *Vres* by the *Create & Bind Endpoint* services. Finally, the *Send/Receive* services allow the communication through the respective endpoint.

The network *Virtual resources* in the contract layer must be consistent with the communication parameters within FTT-SE so that the protocol actually enforces the contracted communication parameters with its control mechanisms. Therefore, a *parameters daemon* is used to keep such consistency.

V. Internals of the contracting procedure

The establishment of network contracts with FTT-SE, as referred before, requires an interaction between the *Slave Contract Layer* of the involved nodes and the *Master Contract Layer*. The process is triggered by the thread that manages the contract or the contract group and its sequence diagram is depicted in Fig. 4.

The request is enqueued in the *Master Contract Layer* until it can be processed (Fig. 5). At that point, it is removed from the requests queue and submitted for admission process, which involves the admission control in the FTT-SE master. If the contract is accepted, which may result in changes to other contracts, the master updates the FTT-SE internal structures and publishes all *Vres* that were updated. The respective slaves receive this information and update/create the respective *Vres copies*. Then, the master acknowledges the negotiation result.

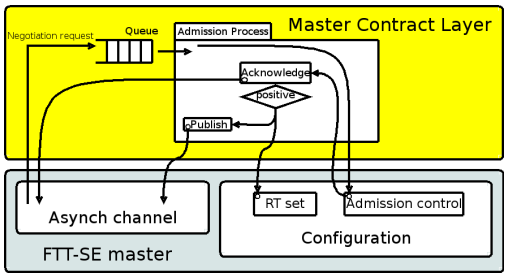

Fig. 5. Master contract negotiation procedure

VI. Concluding remarks and On-going work

The FRESCOR framework has been proposed recently to cope with the growing application complexity and interoperability requirements in embedded systems. The approach followed by FRESCOR allows abstracting the management of the application resources, which are accessed through a common interface based on contracts.

In this paper we discussed the integration of the FTT-SE real-time communication protocol within the FRESCOR contract layer framework. This framework efficiently exploits the FTT-SE natural ability for dynamically adapting the network resource usage while maintaining predictability. Another positive aspect in this symbiosis is that the interface given by the framework to the application can be commonly applied together with other shared resources of the system.

Previously, only the RT-EP network protocol had been integrated within the FSF/FRESCOR framework. Such protocol works over a shared medium with a priority based event-triggered messaging paradigm. The integration of FTT-SE brings in the features of a different network paradigm and topology, namely time-triggered and event-triggered communication over Switched Ethernet. We believe that the dynamism of FTT-SE will impact positively on the efficiency of network management in the FRESCOR framework. On the other hand, the abstraction provided by FRESCOR will benefit the FTT-SE protocol in terms of its usability and applications development.

Currently we are carrying out the temporal analysis of the negotiation process. In the near future we plan to apply the implementation herein described to an application that allows illustrating its flexibility and negotiation capabilities.

References

[1] N. Wang, D. Schmidt, K. Parameswaran, and M. Kircher, "Towards a Reflective Middleware Framework for QoS-enabled CORBA Component Model Applications," IEEE Distributed Systems Online special issue on Reflective Middleware (Vol. 2, No. 5), May 2001.

[2] M. Aldea *et al.*, "FSF: A Real-Time Scheduling Architecture Framework," in *12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*. San Jose (CA, USA): IEEE, Apr. 2006, pp. 113–124.

[3] R. Marau, L. Almeida, and P. Pedreiras, "Enhancing real-time communication over COTS Ethernet switches," in *WFCS'06: IEEE International Workshop on Factory Communication Systems*, 27 June 2006, pp. 295–302.

[4] J. M. Martínez and M. G. Harbour, "RT-EP: A Fixed-Priority Real Time Communication Protocol over Standard Ethernet," in *10th International Conference on Reliable Software Technologies, Ada-Europe*. Springer, June 2005, pp. 180–195.