Journal of Systems Architecture 60 (2014) 221–233

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

# Comparative analysis of two different middleware approaches for reconfiguration of distributed real-time systems

Marisol García Valls *, Pablo Basanta Val

*Departamento de Ingeniería Telemática, Universidad Carlos III de Madrid, Leganés, Madrid, Spain*

ARTICLE INFO

*Article history:*
Available online 24 August 2013

*Keywords:*
Reconfiguration
Distributed systems
Middleware
Real-time

ABSTRACT

Software-based reconfiguration of distributed real-time systems is a complex problem with many sides to it ranging from system-wide concerns down to the intrinsic non-robust nature of the specific middleware layer and the used programming techniques. In a completely open distributed system, mixing reconfiguration and real-time is not possible; the set of possible target states can be very large threatening the temporal predictability of the reconfiguration process. Over the last years, middle ware solutions have appeared mainly for general purpose systems where efficient state transitions are sought for, but real-time properties are not considered. One of the few contributions to run-time software reconfiguration in distributed real-time environments has been the iLAND middleware, where the germ of a solution with high potential has been conceived and delivered in practice.[1] The key idea has been the fact that a set of bounds and limitations to the structure of systems and to their open nature needs to be imposed in order to come up with practical solutions. In this paper, the authors present the different sides of the problem of software reconfiguration from two complementary middleware perspectives comparing two strategies built inside distribution middleware. We highlight the lessons learned in the iLAND project aimed at service-based reconfiguration and compare it to our experience in the development of distributed real-time Java reconfiguration based on distributed tasks rescheduling. Authors also provide a language view of both solutions. Lastly, empirical results are shown that validate these solutions and compare them on the basis of different programming language realizations.

Crown Copyright © 2013 Published by Elsevier B.V. All rights reserved.

## 1. Introduction

Information and communication technologies are rapidly driving the road to an enhanced distributed computing paradigm where public, private, individual, and group's resources can all be put in common to provide enhanced processing power in a highly connected environment. In this way, the resources needed and utilized by application users are, in part, hosted remotely and provided from the outside. This is one of the principles of cloud computing that allows to make use of huge amounts of computational and storage resources [1] by means of reduced computing power devices as personal smart phones [2], tablets, etc. Applications and data are run and stored on virtualized resources that are potentially shared with large numbers of other users. At the same time, this brings in interesting properties for companies and society in general, such as energy efficiency and environment-friendly.

The appearance of efficient communications middleware technologies and flexible software paradigms as *service oriented architectures* (SOA) has been of paramount importance for enabling open and versatile environments. It provides a fertile soil for developing new applications with a decoupled and flexible structure. Still, support for real-time in such domains has not progressed so fast. There are a number of reasons for this. Although real-time research has provided numerous results in scheduling theory, the practical implementation of real-time software systems is still the bottleneck to timing predictability. Specially, the communications middleware level presents numerous challenges that are conceptually simplified in the theory. Middleware is often taken as a black box with a number of needed properties and assumptions that fit well with the real-time abstractions and models. However, in practice, middleware technologies have not completely eliminated the uncertainty brought in by the underlying techniques such as serialization, address resolution, transport and Internet level details, among others. In fact, as they introduce extra logic, they also increase the number of sources of unpredictability.

Providing real-time guarantees in distributed computing environments is a hard problem, and it becomes non tractable if these are considered to be open systems with dynamic behavior. The

* Corresponding author. Tel.: +34 916248783.
*E-mail addresses:* mvalls@it.uc3m.es (M. García Valls), pbasanta@it.uc3m.es (P. Basanta Val).

[1] http://sourceforge.net/projects/iland-project/

1383-7621/$ - see front matter Crown Copyright © 2013 Published by Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.sysarc.2013.08.010

question is that openness and dynamicity are characteristics of the emerging applications; systems undergo changes that may require a modification of their structure to adequately process and react to events from the environment. In this work, we refer to such changes as *reconfigurations*, i.e., the process of transitioning from the *current system structure* (or *configuration*) to the *target one*.

For open distributed systems, merging reconfiguration with real-time is not solvable with the available techniques. A set of bounds and limitations to the structure of systems needs to be imposed in order to reduce complexity by means of limiting the size of the space of solutions, i.e., the number of possible target configurations. This is one of the first lessons learned in our previous work in the iLAND project [3,4] that resulted in the identification of a set of phases for the complete time-bounded reconfiguration process. We further elaborated on some of the reconfiguration problems in [5] providing a high-level view of these, and showing how to reduce the space of solutions by embedding extra logic in the reference implementation of iLAND [3,6].

### 1.1. The contribution

Different languages such as Ada [27], Java (Real-Time Specification for Java or RTSJ) [28], or C/C++ have been instrumental for implementing distributed real-time systems. The actual distribution middleware for real-time systems such as RT-CORBA [29], Distributed RTSJ [22], DSA [30], or DDS [31] are actually silent about run-time reconfiguration aspects. Even the modeling languages such as MARTE UML [32,53,54] do not integrate run-time real-time reconfiguration capacities. They rather support the specification and usage of the basic building blocks that may be reused in other high-level abstractions for specific applications. As a result, these specifications do not provide templates or entities that enable control on dynamic functionality replacement during system operation. Providing practical approaches to real-time reconfiguration is one of the next challenges in real-time systems; in fact, this is a fundamental property of CPS (Cyber Physical Systems).

Only some initial steps have been taken to achieve real-time reconfiguration guided by specific research projects in different domains such as [25] for real-time systems and [34] for mainstream service provisioning. Their efforts in defining simple and reusable patterns and techniques for either real-time reconfiguration or considering a restricted view of timing parameters contribute to a general corpus of techniques that may be used to provide real-time system practitioners with a valuable glossary.

Our contribution subsumes results from two different approaches (initially presented in [5,33]) that provide practical real-time reconfiguration solutions embedded in the core of distribution middleware. These are: (1) iLAND, that is an DDS (Data Distribution Service for real-time systems) based middleware enhanced with reconfiguration logic and real-time resource management; and (2) DREQUIEMI, that is a middleware based on DRTSJ – Distributed Real-Time Java). The first presented reconfiguration approach is based on the empirical experience gained in the iLAND project [5] that focused on providing real-time reconfiguration capabilities for service-based real-time applications. The DRTSJ reconfiguration approach presents a Java-based middleware [33] to reschedule distributed task sets by means of Java language templates.

In our previous work ([5,33]), we discussed separately the individual problems and solutions to software reconfiguration on each of the two above-mentioned middleware. In this paper, we present an integrated view of them that provides a common corpus of reusable techniques that may be employed in different distributed real-time applications. We present a new comparative view, elaborating a common benchmark that draws interesting conclusions on both reconfiguration approaches and provides useful hints on their behavior and suitability under different conditions.

The rest of the paper describes the two previously introduced strategies under a unified and revised perspective. Section 2 introduces the problems of software reconfiguration. Section 3 presents the different approaches to software reconfiguration from the point of view of the distribution middleware. Section 4 describes system and application models for each of the middleware approaches: based on services or tasks. Also, it presents an overview of the API that shows the middleware realization of the reconfiguration approaches. Section 5 presents empirical results about the reconfiguration models of both middleware approaches and contributes a common benchmark for reconfiguration. Section 6 describes the related work, and section 7 draws some conclusions about the work.

## 2. Exposing the problems of *software* reconfiguration

Run-time reconfiguration challenges temporal predictability at the different architectural layers. We have investigated different sides of software reconfiguration in distributed systems that have real-time requirements. A general description of the challenges and approaches to reconfiguration is provided in this section.

### 2.1. Some reconfiguration challenges

Real-time applications may have different degrees of tolerance to deadline misses. In soft real-time applications, deadline misses result only in a degraded but acceptable operation, e.g., real-time video processing in intelligent co-operative nodes. We target at emerging distributed applications such as CPS where some parts have soft real-time requirements; they must be designed in order to react to the occurrence of events that may require some adaptation (or reconfiguration) of the system in terms of: (i) *functional structure* where some software parts need to be removed, replaced, or newly added, or (ii) *internal processing configuration* where the same functionality may continue to be provided but adaptation of the processing parameters is performed. An example of a reconfiguration event is one generated by an unexpected movement in some high-security perimeter as a sign of a possible intrusion detected by a sensor; it may require that a different camera set be activated instantaneously and higher resolution images are captured and sent at the same time.

Reconfiguration events can be triggered either *internally*, i.e., due to the self monitoring process, or *externally*, i.e., requested by an external entity such as a user. Moreover, reconfiguration triggers may arrive:

- *Synchronously*; at a specified time when the system is able to process them.
- *Asynchronously*; with an unknown arrival pattern. They can be modeled as a periodic task as well.

To adequately process reconfiguration events, real-time systems require that the transition, either to a new functional structure or a new internal processing configuration, is time bounded. However, there are different important threats to preserving temporal predictability in the presence of dynamic behavior. A number of considerations to be made are:

- **Dynamic *versus* static entity membership**. Systems can have a *fixed size* or *dynamic size*. The former does not allow any modification of the software execution units at run-time, whereas the latter does allow the spontaneous dis/appearance of active software units at any time.