DES (Data Exchange System), a publish/subscribe architecture for robotics.

C. Riquier, N. Ricard, C. Rousset
ECA
Rue des Frères Lumière
83130 La Garde

Abstract

This paper presents ECA software architecture for robotics projects such as Miniroc or AUVs. This architecture is made of two parts:

- Software architecture is the tool to exchange data between processes ➔ the DES: Data Exchange System.
- Functional architecture is the organization of processes in order to fulfill robot functions.

This paper presents the DES layer and gives an example of utilization.

The DES is based upon a publish/subscribe design.
A Process is a publisher of the data it "creates", and a subscriber to the data it needs. It doesn't need to know which process will publish the data it needs, neither which will use the data it publishes. Communications are based upon TCP/IP channels directly between the publishers and the subscribers. A "Mediator" manages all communications between processes. All processes ask what they want and tell what they can give. The Mediator tells everyone who can give what they want. Then all communication links are established directly between processes. At anytime, a process can ask something more, or stop sending a data. The Mediator also deals with process disappearance or arrival.
The same data can be published by several publishers with different priorities. Data can have a period of validity.
Processes can be on different computers and they don't need to know where the other processes are.
This architecture allows modular hot plug of payloads on robots.

I - INTRODUCTION

Several years ago (last century in fact !), all ECA robots were based upon point to point client-server architectures. Most of them had only two processes: one for HMI and one embedded in the vehicle.
Around year 2000, according to the increasing complexity of robots (more sensors, more autonomous behaviors, more computers, …), the need of a distributed, reusable and flexible architecture arises.
Among available concepts of architecture, we chose the "Publish – Subscribe" one. The first chapter compares three kinds of possible architecture.

The first "Publish – Subscribe" architecture we developed was named "BDC" (Broadcast Data Center). Our first AUVs were built around it. After two years of utilization, and according to robots more and more demanding for "real time" performances, we specified some improvements of the BDC which has been renamed "DES".
This paper only describes the DES architecture which now equipped our AUVs and "Miniroc" ground robots (military robots for DGA).

The last chapter of the document illustrates the utilization of the DES, with an example extract from the Miniroc architecture.

II – COMPARISON OF SEVERAL COMMUNICATION ARCHITECTURES

Distributed real-time applications have unique communication requirements. Real-time applications must handle different kinds of data flow, such as repetitive updates, single-event transactions, and reliable transfers; many nodes intercommunicate, making data flow complex; and dynamic configuration changes occur as nodes leave and join the network.
Strict timing requirements further complicate the entire design.
Traditional client-server architectures route all communications through a central server. This makes them ill-suited to handle real-time data distribution. Publish-subscribe architectures, designed to distribute data to many nodes simultaneously and anonymously, have clear advantages for real-time application developers: they are more efficient, handle complex communication flow patterns, and map well to underlying connectionless protocols such as multicast.

Distributed application developers have several choices for easing their communications effort:
- Client-server, either in the traditional form of a central server node intermediating for a set of clients or its updated manifestation – distributes objects and object brokers
- Publish-subscribe, in the form of middleware that distributes data – publications – anonymously among applications in one-to-many patterns.

II.1 Client-Server Architectures

Client-server communications generalize the data flow by allowing one server node to connect simultaneously to many client nodes. Thus, client-server is a many-toone architecture. It works well when the server has all the information. Examples of client-server applications include database servers, transaction processing systems and central file servers.
When the data is produced by multiple nodes for consumption by multiple nodes, client-server architectures are inefficient because they require an unnecessary transmission step:

instead of direct peerto-peer, the data must go through the server. The transmission to the server also adds unknown delay to the system. Furthermore, the server can become a bottleneck and presents a single point of failure. Multiple-server nets are possible, but they are very cumbersome to set up, synchronize, manage, and reconnect when failures occur. This resolves bottleneck and point-of-failure exposures, however it only increases inefficiencies and bandwidth consumption.

II.2 Object Brokers

CORBA and DCOM are the best-known examples of distributed object architectures. Distributed objects architectures are middleware that abstract the complex network communication functions and promote object re-usability, two features that substantially reduce the programming effort. Object brokers do not address several distributed realtime application data flow characteristics, however: they offer little support to control the properties governing deterministic data delivery (especially important for signal data) and are cumbersome and unwieldy when programming dynamic, many-to-many flow patterns. This largely derives from the distributed objects inherent and fundamental reliance on a broker to route requests and its object management requirements.

II.3 Publish-Subscribe

The publish-subscribe architecture is designed to simplify one-to-many data-distribution requirements. In this model, an application "publishes" data and "subscribes" to data. Publishers and subscribers are decoupled from each other too. That is,

- Publishers simply send data anonymously, they do not need any knowledge of the number or network location of subscribers.
- Subscribers simply receive data anonymously, they do not need any knowledge of the number or network location of the publisher.

An application can be a publisher, subscriber, or both a publisher and a subscriber.

Publish-subscribe architectures are best-suited to distributed applications with complex data flows.

The primary advantages of publish-subscribe to applications developers are:

- Publish-subscribe applications are modular and scalable. The data flow is easy to manage regardless of the number of publishers and subscribers.
- The application subscribes to the data by name rather than to a specific publisher or publisher location. It can thus accommodate configuration changes without disrupting the data flow.
- Redundant publishers and subscribers can be supported, allowing programs to be replicated (e.g. multiple control stations) and moved transparently.
- Publish-subscribe is much more efficient, especially over client-server, with bandwidth utilization.

Publish-subscribe architectures are not good at sporadic request/response traffic, such as file transfers. However, this architecture offers practical advantages for applications with repetitive, time-critical data flows.

III –DES: Data Exchange System

III-1 – General Principles of Publish – subscribe architectures

Several main features characterize all publish-subscribe architectures:
Distinct declaration and delivery. Communications occur in three simple steps:
- Publisher declares intent to publish a publication.
- Subscriber declares interest in a publication.
- Publisher sends a publication issue.

Named publications: PS applications distribute data using named publications. Each publication is identified by a name by which a publisher declares and sends the data and a subscriber declares its interest.

Many-to-many communications support: PS distributes each publication issue simultaneously in a one-to-many pattern. However, the model's flexibility helps developers implement complex, many-to-many distribution schemes quite easily. For example, different publishers can declare the same publication so that multiple subscribers can get the same issues from multiple sources.

Event-driven transfer. PS communication is naturally event-driven. A publisher can send the datum when it is ready. A subscriber can block until the datum arrives. The publish-subscribe services are typically made available to applications through middleware that sits on top of the operating system's network interface and presents an application programming interface (see Figure 1). The middleware presents a publishsubscribe API so that applications make just a few simple calls to send and receive publications. The middleware performs the many and complex network functions that physically distribute the data..

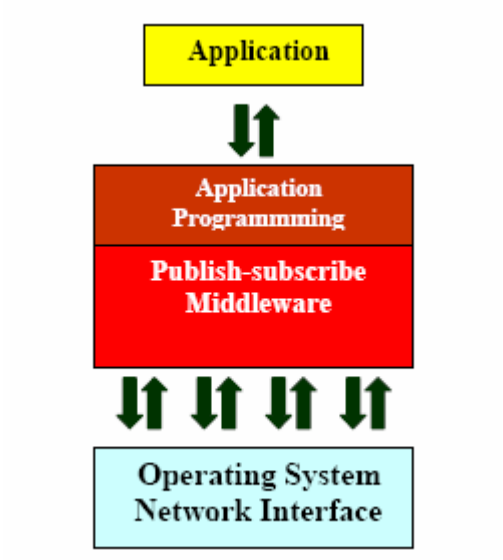

Figure 1. Generic Publish-Subscribe Architecture

III – 2 – DES overview

All processes of the architecture are called “Agents”.

There is one special agent which is essential: the MEDIATOR
It is the “heart” of the DES. This Deamon is connected with all the agents running in the system. Any time a new data flow is required or an existing data flow disappear, the Mediator send to the concerned agents the pieces of information they need to establish or destroy the data flow. So the Mediator neither sends nor receives any data flow. It just establishes them directly from publisher(s) to subscriber(s).
The figure 2, shows the sequence of life of a data flow: all the transitions between the steps of life are supervised by the Mediator.
The figure 3, shows the data flow itself once it is established (in the state “Publication” of the figure 2).

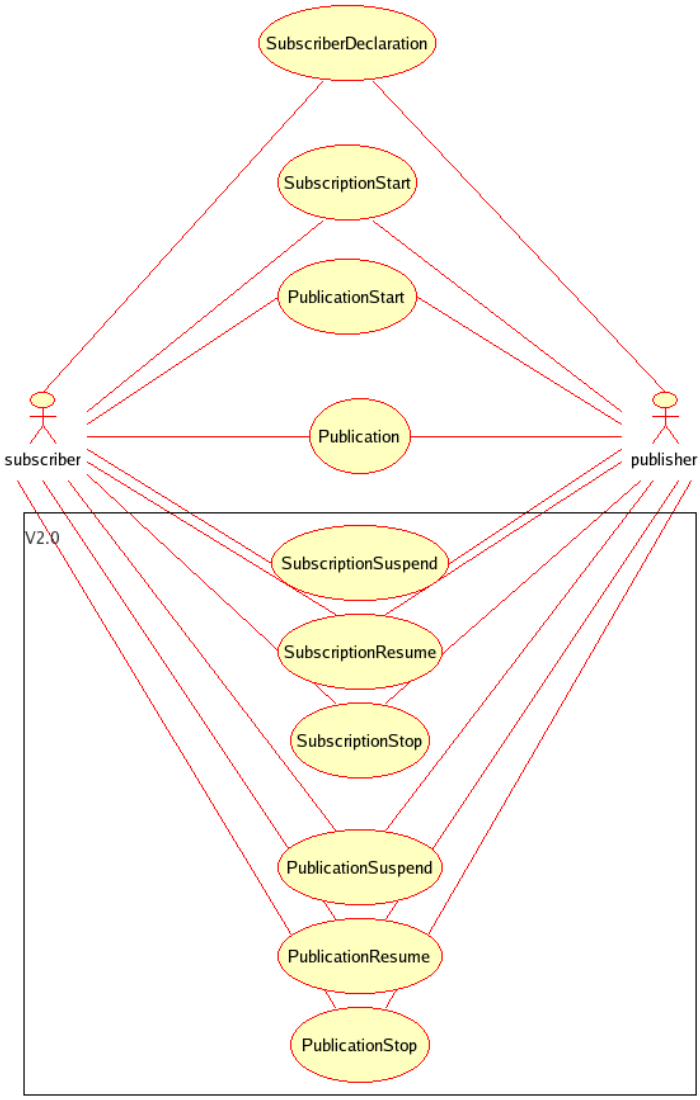

Figure 1 : sequence of life of a data flow

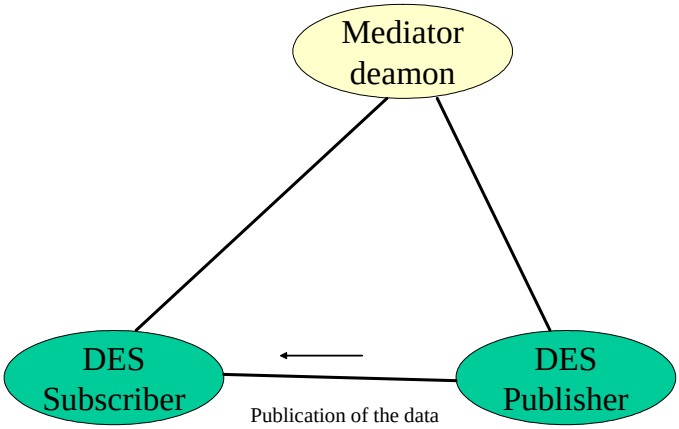

Figure 2 : data flow publication

Three over system processes can be used:

SWC: SoftWare Controller :
This service is a daemon which starts all the agents (including the Mediator), monitors them (from the OS point of view). Some of the agents are defined as “critic”. If one of the critical agents crashes, the SWC halts all the system properly.

DRC: Data Recording Center
This service is a special agent which subscribes to all the data you have configured, and records them with the date.

NTS: Network Time Synchronization
This service is not really part of DES architecture but it is required for dating of data, as soon as the architecture is distributed among several CPUs. We use the NTP (Network Time Protocol) implementation provided with the OS.

III – 3 - Different ways to exchange data through DES

The basic principle of publish subscribe is that the subscriber does not decide when to receive the data. It receives it when the publisher publishes it.
However the DES has a middle layer between data reception and the call from agent functions, which allows several ways to exchange data.

The event publication:
The publisher publishes its data. The subscriber DES layer receives it and run the associated callback of the subscriber agent. This allows you to synchronize the subscriber treatments on reception of data. You typically use this to synchronize a perception and guidance agent on the reception of the sensor acquisition agent.

The unsynchronized publication:
The publisher publishes its data. The subscriber DES layer receives it and store it. The subscriber agent can access the data when it needs it. Only the last data received is stored. You typically use this to get some parameters that you don't need to use when they are published, but only when you start your own treatments. It also allows you to get a data which has been published before your agent was started. For example, the kind of robot you are on, or the parameters of the current camera when you want to do some visual treatments on it.

The event publication with FIFO:
Same principle than the event publication but all received data are stored in a FIFO buffer and one event per data is generated, even if your precedent treatment is not completed. For example the subscription to a fire order need to receive all the order sequence (FIRE followed by a CONFIRM).

The unsynchronized publication with FIFO:
Same principle than the event publication, but all received data are stored in a FIFO buffer, and each time the agent request a data, the oldest received data is returned.

The event publication with a validity period:
The publisher defines a time of validity (T) on its data. An event is generated in subscriber agent when the data is received. Another event is generated T sec after the reception and the data is turned invalid. For example the mobility commands published to the agent dealing with the robot drive are using a validity period.

The unsynchronized publication with a validity period:
The publisher defines a time of validity (T) on its data. The data is available for the subscriber during T sec after the reception. After that delay, if the subscriber accesses the data, an invalid access is returned.

The multiple publications without priority:
Several publishers can publish the same data. If you don't define any priority, all published data from all publishers are received by the subscriber (all the precedent kinds of publication can be used).

The multiple publications with priorities:
Several publishers can publish the same data with different priorities. Only the data from the higher priority publisher, is received by the subscriber.

IV – DES Use Example

A typical illustration concerns the mobility commands of a mobile robot.

In our architecture the agents interfacing with hardware are named"EV" (for Virtual Equipment).

The example consists in:

- the "Laser EV" :
  - it acquires the laser rangefinder data
  - it publishes them periodically, event driven by the hardware acquisition.
- the "Vehicle EV" :
  - it subscribes to all the commands the vehicle is waiting for.
  - it publishes all the data coming from the vehicle
- a "Guidance Agent" :
  - it subscribes (event driven subscription) to EV laser data. Thus, the guidance treatments and the mobility command publication are synchronized on the laser data publication.
  - it subscribes to the odometry data (unsynchronized subscription) : when the agent receive a laser data, it accesses to last receive odometry data and utilizes dates of data to resynchronize odometry with laser data.
  - it publishes mobility commands
- a "Teleoperation Agent" :
  - it acquires the operator HMI commands
  - it publishes them periodically only if the operator want to take over manually the control of the vehicle.

Let's focus on mobility commands data flow:

- two publishers are able to publish these commands with different priorities :
  - higher priority for téléopération : the operator can supervise the autonomous guidance and take the hand over if a problem occurs.
  - lower priority for autonomous guidance
- the data published have a validity period, so :
  - the vehicle stop in case of communication loss with the HMI.
  - The lower priority publisher take the hand back, when the higher priority publisher stop to publish

On this very little example, let's imagine some evolutions:

- You want to decrease the number of CPU ➔ copy your payloads applications on the vehicle CPU and plug your laser on the vehicle CPU. Everything is still working without any recompilation neither configuration.
- You want to use your payload on another robot ➔ plug your all payload on the Ethernet switch of the other robot, and give it the address of the new vehicle Mediator. If the new vehicle does not already have a DES Virtual Equipment, you just need to develop the hardware interface.

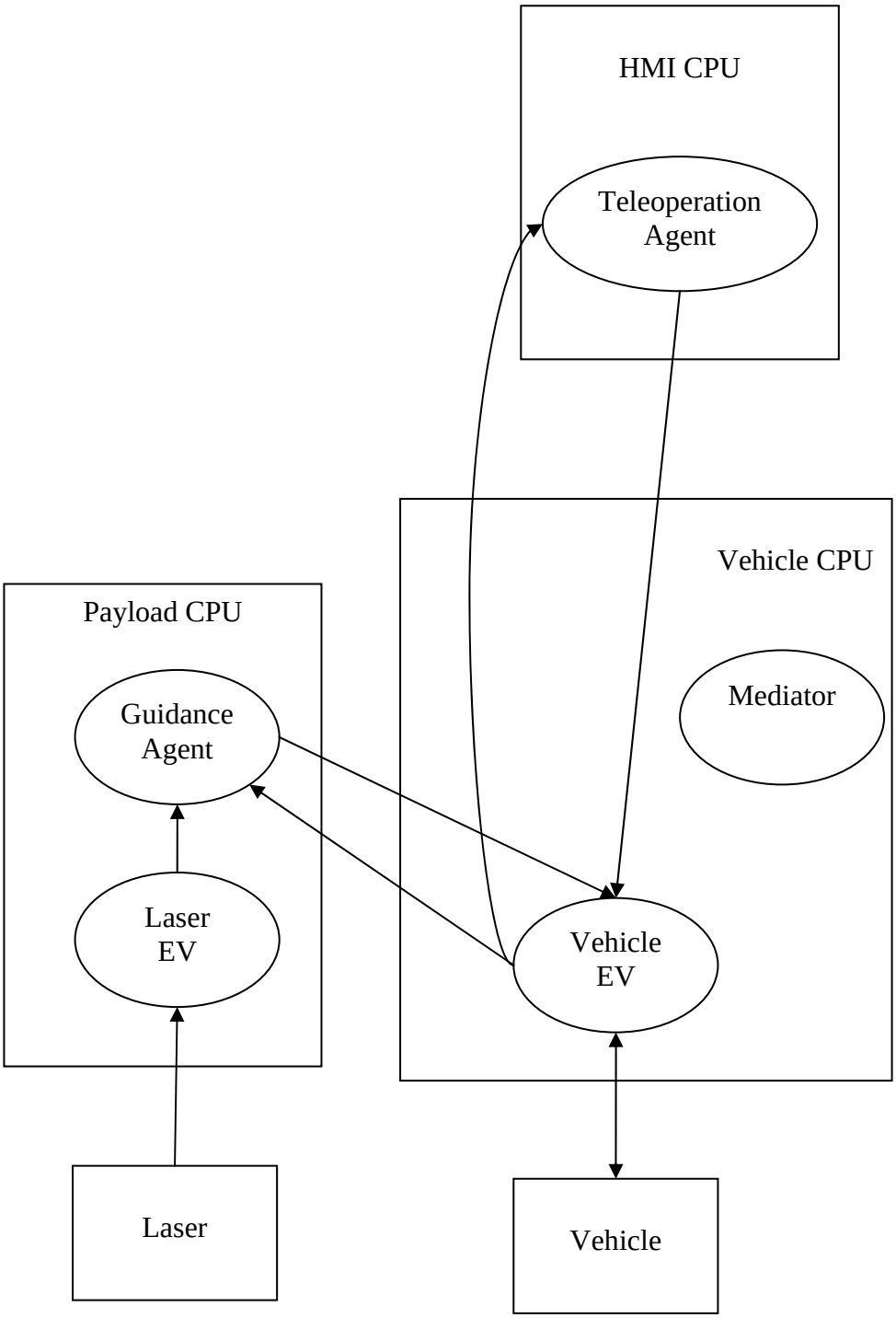

Figure 4: example of data flow