Towards a DDS-based Platform Specific Model for Robotics

Juan Pedro Bandera Rubio, Adrián Garcés and Jesús Martínez

SDIR VI, ICRA 2011
May 9, 2011
Shangai (China)

University of Málaga,
University of Extremadura,
Spain
Motivation and Context
Frameworks and Middleware
*Nerve*: Architecture and Design
DDS in Nerve
Case Study: RLbi architecture
Conclusions
Motivation and Context

Frameworks and Middleware

Nerve: Architecture and Design

DDS in Nerve

Case Study: RLbl architecture

Conclusions
Motivation and Context

- **Robot Control Architectures**: lots of legacy and monolithic code

**RLblI architecture**

Towards a DDS-based Platform Specific Model for Robotics
Motivation and Context

Robotics software falls into the category of DRE systems
- stringent requirements in terms of performance, real-time and robustness

Challenges for robotics developers
- Platform and network independency whether possible
- Modularity and scalability (more processors, distributed environments)
- Select, manage and adjust the appropriate middleware and frameworks
- Robustness, safety, liveness, quality of service attributes...

Using existing paradigms for distributed computing? Which one?
- client/server-based TCP/IP protocols
- remote procedure calls / remote method invocation
- component-based development process

We propose a lightweight middleware called Nerve
- C++, platform-independent modular services
- Standard-based communications using the DDS approach
Motivation and Context

Frameworks and Middleware

Nerve: Architecture and Design

DDS in Nerve

Case Study: RLbl architecture

Conclusions

Towards a DDS-based Platform Specific Model for Robotics
Or the problem of communicating distributed tasks

One-to-one (request/response) and one-to-many (publish/subscribe) models

Many proposals in the robotics community:

- Ad-hoc TCP/IP solutions (Player, Carmen, ROS):
  - (Usually) good to use existing drivers and modules
  - Interoperability with existing legacy code

- CORBA-based solutions (Orocos, Miro, Orca, Robocomp):
  - Ice or CORBA ways of invoking remote object methods (standard)
  - Defining interfaces in a platform-independent way through an IDL language
  - Steep learning curve
  - The ORB *must* be real-time compliant (Orocos, Miro)

- Existing frameworks can not cover all the features needed by a critical networked system for robotics:
  - simultaneous high-performance and quality-of-service communication and concurrency models -> **Nerve**
Motivation and Context

Frameworks and Middleware

Nerve: Architecture and Design

DDS in Nerve

Case Study: RLbl architecture

Conclusions

Towards a DDS-based Platform Specific Model for Robotics
Nerve Features

- Encapsulation of critical tasks as platform-independent services
  - executed as threads or processes at deployment time
- Asynchronous execution model: services react to events available from event queues
- Internal selection of the fastest communication mechanism
  - zero-copy buffering for threads, shared memory for local processes, reliable multicast for remote processes

Nerve wrappers and execution logic

OpenSplice DDS

(OpenDDS support in progress)
Towards a DDS-based Platform Specific Model for Robotics

- **Nerve::Execution_Manager**
- **Nerve::Topic_Manager**
- **ReaderFactory**
- **ACE_Task**
- **ACE_Message_Queue**
- **ACE_Message_Block**
- **Nerve::Service**
- **svc.conf (deployment)**
- **IReader**
- **IWriter**
- **Nerve::Topic**
- **WriterFactory**

Virtual data buses (“topic flows”) are managed by the Nerve Service with DDS. The main executable is loaded by the svc.conf (deployment) file. The Nerve Execution Manager coordinates service logic and publish/subscribe operations.
Index

- Motivation and Context
- Frameworks and Middleware
- *Nerve*: Architecture and Design
- DDS in Nerve
- Case Study: RLbl architecture
- Conclusions
Data Distribution Service (OMG 2001)

A publish/subscribe specification for critical and real-time systems

A fully distributed global data space. No intermediate brokers.

**Dynamic discovery** of topic data

Commercial and open source implementations

**IDL**

```c
module topic_sample{
    struct data_packet{
        string sensor_name;
        unsigned long value;
    };
    #pragma keylist data_packet sensor_name
}
```

Towards a DDS-based Platform Specific Model for Robotics
Data Distribution Service (II)

- Very low latency & predictable data dissemination (deadlines)
- Fairness
- Stability under overload condition (queues on readers and writers)
- Scalability
- Persistence (using backends)
- Reliability (best effort or reliable delivery)
Quality of Service:
- for publishers, writers, topics, readers and subscribers
- QoS policies matched based on a request vs. offered model
- Powerful but difficult to understand and apply

```
data_packet
sensor_name  value
  “s1”     34
  “s2”     28
```
### Data Distribution Service (IV)

#### Data Timeliness

<table>
<thead>
<tr>
<th>QoS Policy</th>
<th>Applicability</th>
<th>RxO</th>
<th>Modifiable</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEADLINE</td>
<td>T, DR, DW</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>LATENCY BUDGET</td>
<td>T, DR, DW</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>TRANSPORT PRIORITY</td>
<td>T, DW</td>
<td>-</td>
<td>Y</td>
</tr>
</tbody>
</table>

#### Data Delivery

<table>
<thead>
<tr>
<th>QoS Policy</th>
<th>Applicability</th>
<th>RxO</th>
<th>Modifiable</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRESENTATION</td>
<td>P, S</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>RELIABILITY</td>
<td>T, DR, DW</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>PARTITION</td>
<td>P, S</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>DESTINATION ORDER</td>
<td>T, DR, DW</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>OWNERSHIP</td>
<td>T, DR, DW</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>OWNERSHIP STRENGTH</td>
<td>DW</td>
<td>-</td>
<td>Y</td>
</tr>
</tbody>
</table>

#### Data Availability

<table>
<thead>
<tr>
<th>QoS Policy</th>
<th>Applicability</th>
<th>RxO</th>
<th>Modifiable</th>
</tr>
</thead>
<tbody>
<tr>
<td>DURABILITY</td>
<td>T, DR, DW</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>DURABILITY SERVICE</td>
<td>T, DW</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>LIFESPAN</td>
<td>T, DW</td>
<td>-</td>
<td>Y</td>
</tr>
<tr>
<td>HISTORY</td>
<td>T, DR, DW</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>
- Motivation and Context
- Frameworks and Middleware
- *Nerve*: Architecture and Design
- DDS in Nerve
- Case Study: RLbl architecture
- Conclusions
Towards a DDS-based Platform Specific Model for Robotics
Towards a DDS-based Platform Specific Model for Robotics
Input: Stereo cameras.

Perception: Filters the huge amount of input data to extract human pose.

These components (specially feature extraction and tracking) represent a very important percentage of the total computational load of the system.
Gesture are **represented** using global and local trajectory features.

**Gesture recognition** may become a time-consuming process.

**Learning** requires human supervision.

Recognition and learning can easily be conducted in parallel to other processes (i.e. imitation).
Gestures are retargeted to robot motion space **only if** physical imitation is required.
The original multithreaded architecture

<table>
<thead>
<tr>
<th>Measured Service (in fps)</th>
<th>Orig. RLblI</th>
</tr>
</thead>
<tbody>
<tr>
<td>MovementCapture</td>
<td>8.41</td>
</tr>
<tr>
<td>PersonModel</td>
<td>26.80</td>
</tr>
<tr>
<td>RobotModel</td>
<td>26.80</td>
</tr>
</tbody>
</table>
The previous architecture can be improved using some of the characteristics we have exposed and the capabilities of *Nerve*:

- **Capturing images, filtering features and tracking** can be executed in its own service. This service is a strong candidate to be executed in a different PC.
- The **recognition and learning** stages should be executed in parallel with respect to the other steps of the process.
- The **retargeting** stage, **motion generation** component and **output** component are only used in certain situations, and they are independent from the previous processes.
RLbi architecture improved with Nerve

Our final deployed architecture

<table>
<thead>
<tr>
<th>Measured Service (in fps)</th>
<th>Orig. RLbi</th>
<th>Nerve-RLbi (local)</th>
<th>Nerve-RLbi (distributed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MovementCapture</td>
<td>8.41</td>
<td>10.16</td>
<td>14.98</td>
</tr>
<tr>
<td>PersonModel</td>
<td>26.80</td>
<td>27.50</td>
<td>40.21</td>
</tr>
<tr>
<td>RobotModel</td>
<td>26.80</td>
<td>27.50</td>
<td>40.23</td>
</tr>
</tbody>
</table>
Motivation and Context

Frameworks and Middleware

Nerve: Architecture and Design

DDS in Nerve

Case Study: RLbl architecture

Conclusions
Conclusions and future work

- Nerve is now being used in the context of DRE systems for robotics. The results when applied to the case study (RLbl) are promising.

- Using standards (DDS) for communications to cover publish/subscribe and request/response needs.

- Improving modularity without sacrificing high-performance and robustness.

Next steps:
- releasing the first public beta release using a BSD open source license (~May)
- integration with modern visual approaches for designing software systems as models: MDA, UML, MofScript... (now in progress using SmartSoft)
Towards a DDS-based Platform

Thanks for your attention!

Questions, advice?

University of Malaga,
University of Extremadura,
Spain