Demo: Ball and Plate Wireless Control

Maxime Louvel, François Pacull, Maria Isabel Vergara-Gallego
Univ. of Grenoble Alpes
CEA, LETI, MINATEC Campus
{FirstName.LastName}@cea.fr

Abstract

This demonstration uses the transactional guarantees of the LINC middleware to increase the reliability of a wireless control system. Three motors lift up a plate with a ball on top. In order to keep the ball on the plate, the motors must move at the same time.

Keywords

Reliability, Redundancy, Wireless Sensors and Actuators

1 Introduction

Wireless Sensor and Actuator Networks (WSANs) are now targeting control applications in many domains such as industries, medical, and mission-critical systems. Theses domains require more reliability and robustness than traditional monitoring systems.

Despite the efforts to improve communication error resilience [5], in most cases, end-to-end data delivery is not guaranteed in WSAN. Therefore, it is not possible to guarantee, at the application level, the state of the system.

This demonstrator consists of a ball and plate control. Three motors are used to raise up and down the plate. To keep the ball on the plate, the motors must be synchronised. While this might be quite simple from a control point of view, the challenge here is to control the motors with an unreliable wireless communication, and unreliable actuators. Indeed, if one motor stops working, the other two must stop to prevent the ball from falling down. This paper details how the demonstrator has been built thanks to the transactional guarantees offered by the LINC [4] middleware. The problem is solved with a simple rule, executed by several components to provide redundancy.

As illustrated in Figure 1, each motor is controlled by a wireless communicating device. The first motor is controlled by an *ArduinoUno*, which communicates through *ZigBee*, and the other two are controlled by an *Openpicus* communicating through *Wi-Fi*. The motors are controlled by a LINC rule that permits the movement of the three motors step-by-step. In this way, the total number of steps performed by each motor is kept synchronised. A user interface displays the demo state on a tablet. For redundancy purposes, the rule is executed by two coordinators: the first runs on a *Beaglebone black*, the second one on a *RaspberryPi* B.

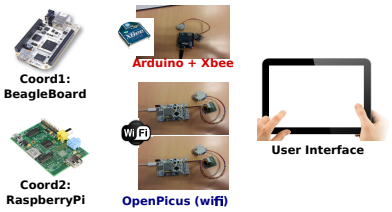

Figure 1. Platforms and hardware used in the Demo

2 LINC Transactions for Reliability

LINC [4] is a rule-based middleware able to coordinate the actions of a group of components through a high level protocol. It provides an abstraction layer that relies on the associative memory paradigm implemented as a distributed set of bags containing resources (tuples). Inspired by Linda [1], bags are accessed by three operations: `rd()`, `get()`, and `put()`, that permit to respectively read, consume or add resources in the bags.

These three operations are used within *production rules* [2]. A production rule is composed of a *precondition phase* and a *performance phase*. In the precondition phase a set of `rd()` operations are done to detect or wait for the presence of resources in the bags. In the performance phase a set of actions are performed in a *Distributed Transactional* way, so that operations on sensors, actuators, and software components are embedded in a transaction. This ensures that all the considered operations will be done in an atomic way. Consider the example of transaction embedding the sending of the command to the three motors. If one motor is broken, the rest of the transaction is not executed and all the motors stop, preventing the ball from falling.

LINC rules are executed by dedicated components called coordinators. A rule can be executed by several coordinators who may run in different machines or different networks to provide redundancy. Thanks to the transactions, the same rule may be executed several times. If the transaction con-

International Conference on Embedded Wireless Systems and Networks (EWSN) 2016
15–17 February, Graz, Austria

ISBN: 978-0-9949886-0-7

sumes a unique resource, LINC enforces that only one transaction will succeed, the other one will fail when consuming the resource.

Transactions are implemented with a two phase commit. The first phase of the transaction checks that all the actions involved can be executed. The resources are locked for this transaction. Any other transaction trying to access a resource reserved will be cancelled and tried again later by the coordinator. If one action fails, all the actions are cancelled. In the second phase, the resources are actually consumed and the command (added as resources in dedicated bags) are sent to the actuators. An action may fail because a resource has been consumed by another transaction, or due to a failure to access the remote bag.

In order to implement the transactions with sensors and actuators, a part of the LINC protocol has been embedded in the micro-controller of the sensors/actuators [3].

3 Demo Description

Listing 1 shows a simplified rule controlling the system. The precondition phase reads (line 1) the current step of one of the motors (they are all synchronised) and computes the new step (line 2). Then, the performance phase (after `::`) starts by consuming the current step of the motors (line 5 to 7) and sends the commands to lift up the three motors (line 8 to 10). Note that the actual rule also makes the motors lift-down by a more complex computation in line 2.

```
1  {*,!}["OpenPicus2", "Actuators"].rd(step)&
2  INLINE_COMPUTE: new_step=str(int(step)+1) &
3  ::
4  {
5  ["Arduino1", "Actuators"].get("NStep", step);
6  ["OpenPicus1", "Actuators"].get("NStep", step);
7  ["OpenPicus2", "Actuators"].get("NStep", step);
8  ["Arduino1", "Actuators"].put("NStep", new_step);
9  ["OpenPicus1", "Actuators"].put("NStep", new_step);
10 ["OpenPicus2", "Actuators"].put("NStep", new_step);
11 }.
```

Listing 1. Rule to control the movement of the three motors

In addition to executing the rule, each coordinator informs the other one, for recovery purposes, when:

- it starts a transaction;
- it finishes the first phase, informing as well on the success or failure of the transaction;
- it finishes the second phase of a transaction.

The information required to finish the transaction is also exchanged (instantiated variables). Hence, one coordinator may finish the transaction if the other fails, ensuring the consistency of the system.

In the normal mode all the equipment and the communication are working properly. Both coordinators try to change the step of all the motors. The transaction ensures that only one rule succeeds, the second one to execute will fail when getting the step value from the devices, because meanwhile it has changed.

This demo can tolerate the following failures:

- Connection failure may occur when sending a command to one of the controllers.
  - if this happens in the first phase of a transaction, the transaction is rolled back;
  - coordinators register the error and will periodically retry;
  - when the communication is back, the normal execution is resumed.
- One of the coordinators is stopped (e.g. for maintenance). Here, the second coordinator continues the normal execution.
- One of the coordinators unexpectedly fails:
  - If it was not executing a transaction, the other coordinator continues the normal execution;
  - If it was in the first phase of the transaction, the other coordinator enters a recovery mode to cancel the reservations that might have been done and execute the transaction;
  - If is was in the second phase of the transaction, the other coordinator enters a recovery mode to confirm or abort the transaction as the failed coordinator would have done.
- One or more equipment (OpenPicus or Arduino) is disconnected or stopped. The transactions will fail until all the motors are accessible, preventing the ball from falling of the plate.

Finally, the demo provides a lightweight interface used to monitor the state of the system. This interface is accessible from a web browser (e.g. a normal tablet). The interface is also synchronised with the rest of the demo. It is seen as another equipment and, if absent, it prevents the motors from moving. This demonstrates that software and hardware can be safely coordinated in a wireless and distributed system.

4 Conclusions

This demonstration shows how the LINC coordination environment can be used to build reliable control systems in WSAN. The LINC primitives have been embedded in the devices, so that transactions can be performed and the state of the system is kept consistent. This demonstration shows that, even under hardware or communication errors, the synchronisation between the motors is kept.

Acknowledge

This Work was supported by the European projects TOPAs (676760) and Arrowhead (332987).

5 References

[1] N. Carriero and D. Gelernter. Linda in context. *Commun. ACM*, 32:444–458, 1989.

[2] T. Cooper and N. Wogrin. *Rule-based Programming with OPS5*, volume 988. Morgan Kaufmann, San Fransisco, 1988.

[3] H. Iris and F. Pacull. Smart sensors and actuators: A question of discipline. *Sensors & Transducers Journal*, 18(special Issue jan 2013):14–23, 2013.

[4] M. Louvel and F. Pacull. Linc: A compact yet powerful coordination environment. In *Coordination Models and Languages*, pages 83–98. Springer, 2014.

[5] M. A. Mahmood, W. K. Seah, and I. Welch. Reliability in wireless sensor networks: A survey and challenges ahead. *Computer Networks*, 79:166 – 187, 2015.