Poster: Integrating Rich User Interfaces with Real Systems

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

Abstract

This paper offers to rely on a coordination environment, called LINC, to build rich user interfaces for the distributed systems. Thanks to the abstraction and loose coupling offered by LINC, it is possible to use interfaces designed by designers to monitor and control the whole system from a single computer. The paper gives three examples of interfaces.

1 Introduction

Today's systems, so called Internet of Things (IoT), Cyber Physical Systems or Systems of Systems are integrating many heterogeneous and distributed components. They require a coordination layer [11] to tackle the challenges arising in such systems [7].

Based on Linda [3], several coordination environments have emerged in the last decades to help the applications' developer [1, 6, 8–10, 12]. These solutions help to design applications to target the smart* systems. However, most of the time, the graphical user interfaces are ad-hoc and added on top of the application. This prevents from offering a real user experience beyond data plotting in the cloud. For instance, how to build a single interface to monitor a distributed system? How to control a real system from an interface accessible from anywhere? How to provide information to several users, with different views and remotely located?

This paper offers to tackle this issue by integrating the user interfaces as the rest of the application components (e.g. sensors, actuators, embedded systems, existing/independent/autonomous systems, . . .)

2 Rich user interface design

LINC [8] is a coordination environment that has been used in several domains such as building automation [4], smart cities [5], and lift energy management [2]. Following Linda, LINC uses a resource based approach implementing the shared memory as a distributed set of bags. Resources (i.e. tuples) are added, read and removed from those bags. This ensures a decoupling, both in space and time, between the resource producer and the resource consumer. Moreover, this resource approach offers a universal abstraction layer with a very simple API. The coordination of the application is then ensured by an inference engine with transactional guarantees. The application then evolves from one consistent state to another one. More details on LINC may be found in [8].

2.1 Building user interfaces

The loose coupling and the abstraction layer are crucial to provide a rich user experience in today's systems. Indeed, even if the application is distributed, it must be possible to access all the needed information from a single computer, smart phone, or tablet. This paper does not only target monitoring but also controlling applications. A typical example deals with maintenance of systems where a user will connect with a tablet to check the status of the parking, the road, or the factory. It should be possible to remotely introspect the systems and, in some cases, to fix problems as well. Moreover, the same information may be presented differently to several users (e.g. using realistic pictures/schematics or graphics).

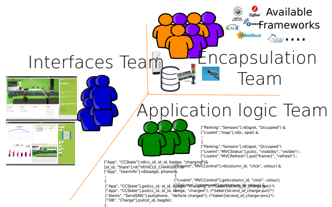
A Model-View-Controller (MVC) approach has been designed on top of LINC. The status of the application is stored in bags. Hence, any kind of information may be added due to the abstraction layer. The graphical aspects of the interfaces are designed, by designers, using standard web technologies (HTML5, SVG). These designs are then updated according to the bags' content when the interface is served to the web browser. User interaction is captured with javascript and added in bags. The controller is implemented by coordination rules enacted by the LINC engine. A simple interaction is to switch on/off a light when clicking on the corresponding graphical element. To do that, the onclick method adds the resource `("eid", "clicked")` in the control bag. The controller then removes this resource, reads the current state of the light, and produces accordingly the resource to trigger the command to switch the light on or off (in the light bag).

Figure 1 presents the three roles for applications development. A first team is in charge of encapsulating the sensors, actuators and other specific components. This team basically provides the low level functions to add resources produced by sensors and to send commands to actuators when

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

ISBN: 978-0-9949886-0-7

resources are added in a specific bag. The second team is in charge of coordination rules, i.e. application's behavior. Finally, the last team, mainly composed of designers, is responsible for the interface design. Interfaces are integrated between the last two teams: the designers set the graphical elements id and the coordination team implements the rules to connect the interface with the system.



```
["Parking","Sensors"].rd(spot, "Occupied") &
["LiveInt","map"].rd(c, spot) &
::
{
["Parking","Sensors"].rd(spot, "Occupied")
["LiveInt","MVCStatus"].put(c, "visibility", "visible") ;
["LiveInt","MVCRefresh"].put("frame1", "refresh") ;
}.
```

```
["LiveInt","MVCControl"].rd(column_id, "click", colour) &
::
{
["LiveInt","MVCControl"].get(column_id, "click", colour);
["Column","command"].put(column_id, colour) &
}.
```

```
["App", "CCState"].rd(cc_id, st_id, badge, "charging") &
[st_id, "State"].rd("VEHICLE_CHARGED") &
["App", "UserInfo"].rd(badge, phone) &
::
{
 ["App", "CCState"].get(cc_id, st_id, badge, "charging"); (*\label{lst:end_of_charge:get}*)
 ["App", "CCState"].put(cc_id, st_id, badge, "charged"); (*\label{lst:end_of_charge:put}*)
 ["Alerts", "SendSMS"].put(phone,  "Vehicle charged"); (*\label{lst:end_of_charge:sms}*)
 ["DB", "Charge"].put(st_id, bagde);
}.
```

Figure 1: Interface design integration

Implementing the MVC with LINC allows to duplicate an interface and to synchronize some or all the interfaces. Indeed coordination rules can keep the consistency between several bags and several interfaces can connect to the same LINC component.

2.2 Examples of user interfaces

Figure 2 shows the interface of a smart parking [5]. The left part is a picture of the parking. The right part is a live interface to monitor and control several equipment in the parking (car sensors, charging stations, lights, cameras, . . .). It is possible to click on a graphical element get information about the corresponding real equipment. It is also possible to force the state of some pieces of equipment by clicking on the associated graphical element.

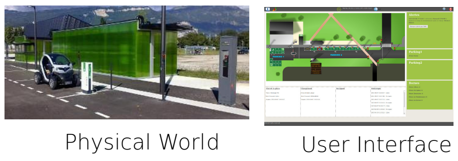


Figure 2: Smart parking interface

Figure 3 shows a light control interface. The left side shows two pictures of the real system with several lights and a box to control all the lights. The right side shows the interface built to control the box. When a light is on in the real world, the interface is automatically updated to make the same light shine as well. From the interface it is possible to click on a light to switch on or off in the reality. The new state will then be propagated to the interface.

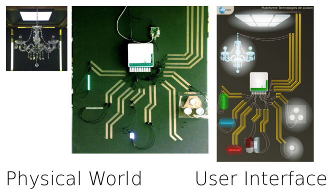


Figure 3: Light control interface

Figure 4 shows an interface to monitor the room of an elderly. The idea is to be able to raise alerts on unexpected behaviors (e.g. activity at night). For privacy concerns, as the person is represented in a graphical manner, this interface enables to follow her behavior without being as intrusive as with a video camera. The actions on lights, windows or doors opening are also monitored. Then it is possible to use the interface to check when an alert is raised.

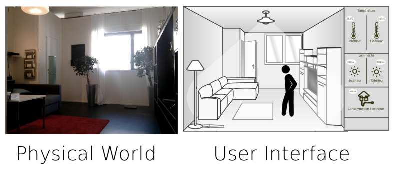


Figure 4: Elderly support

3 Conclusion

This paper has presented preliminary work to design rich user interfaces for distributed systems. Based on the LINC coordination environment, it is possible to monitor and control a distributed system from a single and rich user interface.

Acknowledge

This Work was supported by the French national program "Programme Investissements d'Avenir IRT Nanoelec" ANR-10-AIRT-05.

4 References

[1] J. Barbosa, F. Dillenburg, G. Lermen, A. Garzão, C. Costa, and J. Rosa. Towards a programming model for context-aware applications. *Computer Languages, Systems & Structures*, 38(3):199–213, 2012.

[2] V. Boutin et al. Energy optimisation using analytics and coordination, the example of lifts. In *Emerging Technology and Factory Automation (ETFA), 2014 IEEE*, pages 1–8, Sept 2014.

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

[4] L. Ducreux et al. Resource-based middleware in the context of heterogeneous building automation systems. In *IECON 2012-38th Annual Conference on IEEE Industrial Electronics Society*, pages 4847–4852, Montreal, Canada, 2012. IEEE.

[5] L. Ducreux et al. Rapid prototyping of complete systems, the case study of a smart parking. In *19th IEEE International Symposium on rapid prototyping, Amsterdam, October 2015*, 2015.

[6] C. Julien and G.-C. Roman. Egospaces: Facilitating rapid development of context-aware mobile applications. *Software Engineering, IEEE Transactions on*, 32(5):281–298, 2006.

[7] E. A. Lee. Cyber physical systems: Design challenges. In *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*, pages 363–369. IEEE, 2008.

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

[9] M. Mamei and F. Zambonelli. Programming pervasive and mobile computing applications: The tota approach. *ACM Transactions on Software Engineering and Methodology*, 18(4):15, 2009.

[10] A. L. Murphy, G. P. Picco, and G.-C. Roman. Lime: A coordination model and middleware supporting mobility of hosts and agents. *ACM Transactions on Software Engineering and Methodology*, 15(3):279–328, 2006.

[11] G. A. Papadopoulos and F. Arbab. Coordination models and languages. *Advances in computers*, 46:329–400, 1998.

[12] M. Viroli, D. Pianini, and J. Beal. Linda in space-time: an adaptive coordination model for mobile ad-hoc environments. In *Coordination Models and Languages*, pages 212–229. Springer, 2012.