Player/Stage as Middleware for Ubiquitous Computing

Radu Bogdan Rusu, Alexis Maldonado, Michael Beetz
University of Technology, Munich (TUM)
Boltzmannstrasse 3
85748 Garching, Germany
{rusu, maldonad, beetz}@cs.tum.edu

Matthias Kranz, Lorenz Mösenlechner, Paul Holleis, Albrecht Schmidt
Research Group Embedded Interaction,
University of Munich
Amalienstrasse 17
80333 Munich, Germany
{matthias, lorenz, paul, albrecht}@hcilab.org

ABSTRACT

We propose Player/Stage, a well-known platform widely used in robotics, as middleware for ubiquitous computing. Player/Stage provides uniform interfaces to sensors and actuators and allows the computational matching of input and output. Player/Stage exactly addresses the issues of dealing with heterogeneous hardware but currently only with a focus towards robotics. We show how to integrate ubiquitous computing platforms into Player/Stage and propose Player/Stage as middleware for ubiquitous computing projects.

Keywords

Middleware, Sensors, Actuators, Simulation

1. INTRODUCTION

The effective development and deployment of comprehensive and heterogenous ubiquitous computing applications, such as sensor-equipped living environments or cognitive factories is hindered by the lack of a comprehensive middleware infrastructure: interfaces to sensors are company specific and sometimes even product specific. Typically, these interfaces also do not support the development of robust systems that can support sensor data fusion.

Dealing with uncertainty and many sensor and actuator systems is nothing special to ubiquitous computing as it has been a major issue in robotics as well. But, in contrast, the autonomous robotics community has developed and used software infrastructures and libraries that successfully solved these issues.

We therefore propose Player/Stage, a middleware commonly used as a de facto standard in robotics, as middleware platform for ubiquitous computing. It offers many features that we need in ubicomp as well: a common uniform interface for accessing a great variety of sensors (acceleration, cameras, etc.) and controlling actuators (like switches or displays) and a way to exchange data between them via computational interfaces. Player/Stage also offers the possibility to use 'virtual' hardware, e.g. have a virtual location system at hand for developing context-aware applications without the need to spend hundreds or thousands of dollars for a 'real' location system.

The contribution of this work is as follows: first, we present related work on infrastructures for ubiquitous computing. Then we shortly present the key points of the Player/Stage middleware for robotics. We finally show how several ubiquitous computing platforms (Cparts, Particles, Mica2 and Mica2dots, M1/M1-mini and M300 RFID readers), as well as how automatic data processing algorithms can be easily integrated into the Player/Stage middleware.

2. RELATED WORK

In the last few years, several middleware systems and infrastructures have been proposed for ubiquitous computing. From investigating selected recently proposed middleware systems and architectures, we have extracted and compiled a list of desired functional aspects for a ubiquitous computing middleware. Due to space restrictions, we cannot go into details for each system. The following is a list of the projects we examined together with references for readers who wish to learn more about them:

- iROS [1]
- Phidgets [3]
- Equip Component Toolkit (ECT) [4]
- Real World Interfaces [6]
- Papier-Mâché [5]

The following list summarizes the list of desired properties for such an infrastructure:

- supportability
  - ↪ explicit interaction;
  - ↪ implicit interaction [8];
- adaptability
  - ↪ to roaming and nomadic devices;
  - ↪ to integrate new hardware easily;
  - ↪ to learn from events;
  - ↪ to new (operating system) platforms and devices.
- failure-tolerance and robustness
- support data exchange via APIs and interfaces with
  - ↪ internal components;
  - ↪ external systems.
- support privacy and security, e.g. by allowing
  - ↪ secure data exchange (if needed between components);
  - ↪ private and public events and messages.

3. PLAYER/STAGE – A MIDDLEWARE FOR ROBOTICS

The work on the Player/Stage project started at the University of Southern California in the late ninetees and moved to Sourceforge in 2001. Since then, the user base has grown considerably in size, and currently the pool of project developers consists of people working at universities and research institutions all around the world. Because of its highly active development, the Player/Stage project became a de facto standard in the open source robotics community [2].

The project has two major components:

- **Player**, a distributed device repository server for robots, sensors and actuators, divided into several libraries for enhanced flexibility;
- **Stage**, and **Gazebo**, a 2-D respectively 3-D simulator, which provide the user with tools that support research into multi-agent autonomous systems as well as high fidelity robot simulations.

A device, as defined by Player, is composed of a driver and an interface. Every interface is well-defined, therefore all a driver needs to do is pack the data in the appropriate interface format and provide it to the client. Within the Player concept, a driver can be many things:

- code that connects and communicates to a physical device;
- an algorithm that receives data from another device, processes it, then pushes it back through the same channel;
- a "virtual driver", which can create data when needed.

Due to the standardized interfaces, and because of the fact that Player/Stage was designed to be language and platform (POSIX) independent, various client-side utilities exist for a large variety of programming languages: C, C++, Java, Python, LISP, Ada, Octave, Ruby, Scheme, etc.

Any number of clients can connect to the Player server and access data, send commands or request configuration changes to an existing device in the repository.

Some of the key features of the project are: platform, programming languange, and transport protocol independence; enhanced scalability; open source; standardized communication; high modularity as well as a promoter of software reusability.

4. UBIQUITOUS COMPUTING PLATFORMS AND PLAYER/STAGE

To better illustrate the advantages of using Player/Stage as a standard ubicomp platform, we will show that:

- binding of standard ubicomp platforms to Player/Stage is easy and feasible;
- an ubicomp context can already make use of the existing support for several devices (cameras, ultrasonic sensors, lasers, etc) and reuse the code/algorithms already implemented in Player/Stage, due to the existing standardized, uniform interfaces.

Our work concentrated on adding support for various ubiquitous devices for Player/Stage as well as building software algorithms that could be used in feature extraction and machine learning applications.

We have already added support for a wide variety of devices, such as:

- **Wireless Sensor Nodes** - Cparts and Particles from Particle Computers, Mica2 and Mica2dot from Xbow;
- **RFID readers** - Skeytek M1/M1-mini, Inside M300.

We defined several standard interfaces (wsn, rfid, features) so that different devices can be accessed in the same way by the client, without the need to write additional code. This is a very powerful concept in Player/Stage, which we think will attract the ubicomp community.

As an example, one can scatter a number of different wireless sensor nodes (mica2 and particles for instance), yet use the same code to access the data or to configure the nodes.

An example of accessing the wireless sensor data using Javaclient[7] follows:

```
WSNInterface wsn =
    client.requestInterfaceWSN (index, accesscode);
PlayerWSNData datapacket = wsn.getData ();
float accelX = datapacket.getAccelX ();
```

As mentioned above, we have also developed drivers that act as virtual gateways between the raw data received from the sensors and various different more useful interpretations of it. With that in mind, we defined a new interface called *features* and wrote a driver that can automatically extract data features using PCA, ICA, Wavelet and Fourier techniques. Besides that, the driver can also calculate some of the classical features such as mean, variance, energy, RMS, correlation, kurtosis, etc, from both raw and filtered data.

5. ACKNOWLEDGMENTS

The work has been jointly conducted by the research project Embedded Interaction ('Eingebettete Interaktion') which was funded by the DFG ('Deutsche Forschungsgemeinschaft') and the University of Technology, Munich.

6. REFERENCES

[1] J. Borchers, M. Ringel, J. Tyler, and A. Fox. Stanford interactive workspaces: A framework for physical and graphical user interface prototyping. volume 9, pages 64–69, December 2002.

[2] T. H. Collett, B. A. MacDonald, and B. P. Gerkey. Player 2.0: Toward a Practical Robot Programming Framework. In *Proceedings of the Australasian Conference on Robotics and Automation (ACRA 2005)*, December 2005.

[3] S. Greenberg. Physical user interfaces: what they are and how to build them. In *UIST '04: Proceedings of the 17th annual ACM symposium on User interface software and technology*, pages 161–161, New York, NY, USA, 2004. ACM Press.

[4] C. Greenhalgh, H. J. Izadi S., Mathrick J., and I. Taylor. ECT: A Toolkit to Support Rapid Construction of Ubicomp Environments. In *Conference on Ubiquitous Computing (Workshop on System Support for Ubiquitous Computing UbiSys04)*, 2004.

[5] S. R. Klemmer, J. Li, J. Lin, and J. A. Landay. Papier-mâché: toolkit support for tangible input. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 399–406, New York, NY, USA, 2004. ACM Press.

[6] S. Mccrickard, D. Bussert, and D. Wirghton. A toolkit for the construction of real world interfaces. In *In Proceedings of the ACM Southeast Conference (ACMSE '03)*, pages 118–123, March 2003.

[7] R. B. Rusu, G. Lazea, R. Robotin, and C. Marcu. Towards Open Architectures for Mobile Robots: ZeeRO. In *Proceedings of the Automation, Quality and Testing, and Robotics International Conference (AQTR 2006)*, May 2006.

[8] A. Schmidt. Implicit human computer interaction through context. *Personal and Ubiquitous Computing*, 4(2/3), 2000.

Player/Stage as Middleware for Ubiquitous Computing

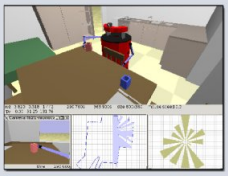
Radu Bogdan Rusu, Alexis Maldonado, Michael Beetz
{rusu,maldonad,beetz}@cs.tum.edu
Intelligent Autonomous Systems,
Technische Universität München

Matthias Kranz, Lorenz Mösenlechner, Paul Holleis, Albrecht Schmidt
{matthias, lorenz, paul, albrecht}@hcilab.org
Research Group Embedded Interaction,
University of Munich

Player/Stage - A middleware for Robotics

The project has two major components:

- **Player**, a distributed device repository server for robots, sensors and actuators, divided into several libraries for enhanced flexibility;
- **Stage**, and **Gazebo**, a 2-D respectively 3-D simulator, who provide the user with tools that support research into multi-agent autonomous systems as well as high fidelity robot simulations.



A device, as defined by Player, is composed of a driver and an interface. Every interface is well-defined, therefore all a driver needs to do is pack the data in the appropriate interface format and provide it to the client.

Within the Player concept, a driver can be many things:

- code that connects and communicates to a physical device;
- an algorithm that receives data from another device, does something with it, then pushes it back through the same channel;
- a "virtual driver", which can create data when needed.

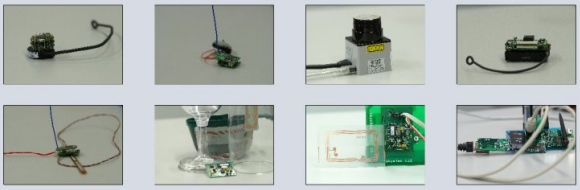
Due to the standardized interfaces, and because of the fact that Player/Stage was designed to be language and platform (POSIX) independent, various client-side utilities exist for a large variety of programming languages: C, C++, Java, Python, LISP, Ada, Octave, Ruby, Scheme, etc. Any number of clients can connect to the Player server and access data, send commands or request configuration changes to an existing device in the repository.

Key features

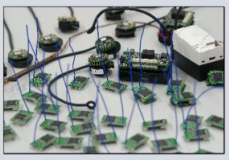
- platform, programming languange, and transport independence;
- enhanced scalability;
- open source;
- standardized communication;
- highly modular as well as a promoter of software reusability.

Infrastructure

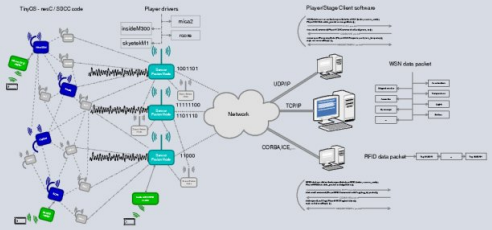
As part of our vision of a supportive and active ubiquitous computing environment, we started to build a sensor-enriched kitchen.



A variety of sensors installed in the AwareKitchen provide the system with the necessary data to study the activities that take place in it: laser sensors, rfid readers, various different wireless sensor nodes, cameras, et al.



Ubiquitous Platforms and Player/Stage



- binding of standard ubicomp platforms to Player/Stage is easy and feasible;
- an ubicomp context can already make use of the existing support for several devices (cameras, ultrasonic sensors, lasers, etc) and reuse the code/algorithms already implemented in Player/Stage, due to the existing standardized, uniform interfaces.

Our work concentrated on adding support for various ubiquitous devices for Player/Stage as well as building software algorithms that could be used in feature extraction and machine learning applications.
We have already added support for a wide variety of devices, such as:

- **Wireless Sensor Nodes** - Cparts and Particles from Particle Computers, Mica2 and Mica2dot from Xbow;
- **RFID readers** - Skeytek M1/M1-mini, Inside M300.

By defining several standard interfaces (wsn, rfid, features), different devices can be accessed in the same way by the client, without the need to write additional code. This is a very powerful concept in Player/Stage, which we think will attract the ubicomp community.
As an example, one can scatter a number of different wireless sensor nodes (mica2 and particles for instance), yet use the same code to access the data or to configure the nodes.

AwareKitchen

Using Gazebo, the whole AwareKitchen environment can be simulated, and thus, algorithms can be developed and tested without using the real hardware. The simulator also provides an easy way to visualize the captured data through the use of the virtual Player drivers.

Data processing algorithms as Player drivers

As part of the AwareKitchen system, we have also developed drivers that act as virtual gateways between the RAW data received from the sensors and various different more useful interpretations of it (eg. drivers that can automatically extract data features using PCA, ICA, Wavelet and Fourier techniques as well as other classical features such as mean, variance, energy, RMS, correlation, kurtosis, etc).