A Player/Stage System for Context-Aware Intelligent Environments

Matthias Kranz[1] Radu Bogdan Rusu[2] Alexis Maldonado[2]
Michael Beetz[2] Albrecht Schmidt[1]
[1]{matthias,albrecht}@hcilab.org [2]{rusu, maldonad,beetz}@cs.tum.edu

[1]Research Group Embedded Interaction, University of Munich
Amalienstrasse 17, 80333 Munich, Germany
[2]Technische Universität München (TUM)
Boltzmannstrasse 3, 85748 Garching b. München, Germany

Abstract. The effective development and deployment of complex and heterogeneous ubiquitous computing applications 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 sustain the development of robust systems that make use of sensor data fusion.
In this paper, we propose the use of Player/Stage, a middleware commonly used as a de facto standard by the robotics community, as the backbone of a heterogeneous ubiquitous system. Player/Stage offers many features needed in ubicomp, mostly because dealing with uncertainty and many different sensor and actuator systems has been a long term problem in robotics as well.
We emphasize they key features of the Player/Stage project, and show how ubicomp devices can be integrated into the system, as well as how existing devices can be used. Additionally, we present our sensor-enabled AwareKitchen environment which makes use of automatic data analysis algorithms integrated as drivers in the Player/Stage platform, of which we are active developers.

1 Introduction

Intelligent sensor-equipped environments can be much more helpful if they are capable of recognizing the actions and activities of their users, and inferring their intentions. Understanding human activities and characterizing them into expressive and detailed activity models is one of the key issues of today's current pervasive computing systems.

Integrated ubiquitous computing environments are still rare to find. By integrated we mean that the ubiquitous technology is seamlessly interwoven within a real-world setting and not put into an artificial laboratory or a single dedicated ubiquitous computing room. Examples of highly integrated sensor-enriched environments are e.g. the Georgia Tech Aware Home [1] or MIT's PlaceLab [2].

Those research facilities enable researchers to develop, build and test context-aware applications in a real-world setting. Building and maintaining these environments is expensive, in terms of time and money, and thus, not many researchers are able to work in them. While an invaluable source for researchers, systems and software are often not publicly available, e.g. Mites as novel sensor platform currently cannot be used by other researchers to reproduce research results. Annotated sensor histories are one solution to share data and context information and allow for algorithms and applications development without the need for the original infrastructure. Again, those annotated histories are costly to produce and much effort for labelling is required.

To deal with the above stated problems, we propose the usage of a proven and tested open-source middleware for sensor and actuator systems. This middleware, Player/Stage [3], is a de facto standard in the robotics research community. After a short introduction of Player/Stage which is so far not known to have been used in a ubiquitous computing context, we show its potentials and use it in our research scenario, the AwareKitchen. We developed interfaces that enable several ubicomp sensor platforms to be connected to Player/Stage, as well as software subsystems that can process raw data and extract relevant features automatically. Work concerning feature selection and semi-automatic gesture classification is in progress and will be made available free of charge as part of the Player/Stage open source software package. After placing our work in context to related work, we conclude by giving an outlook on our work and the potentials of Player/Stage as suitable middleware for ubiquitous computing.

2 Proposed System Overview

We propose the use of Player/Stage, a widely used middleware in the robotics community, as the backbone of a heterogeneous ubiquitous system.

We begin by explaining how the Player/Stage system works, then illustrate how ubicomp devices can be integrated into the system. Next we talk about the possibility of using processing algorithms integrated as Player drivers, and then finally about the existing high-fidelity simulation capabilities of the system.

2.1 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 userbase 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 [3].

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:

- 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 arbitrary 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. to name some. 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; usage of standardized communication protocols and high modularity.

2.2 Player interfaces and drivers for UbiComp Platforms

To better illustrate the advantages of using Player/Stage as a middleware platform in ubiquitous computing, we will show that:

- binding of common 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** - RCores and Particles from Particle Computers, Mica2 and Mica2dot from Xbow;
- **RFID readers** - Skyetek M1/M1-mini, Inside M300.

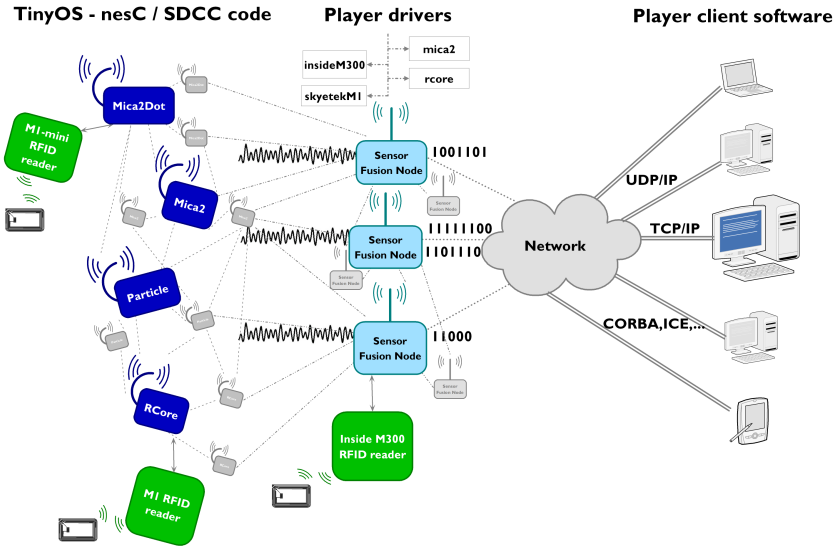

Figure 1. Player/Stage overview for UbiComp Platforms

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 enables us also to substitute devices with the same interface e.g. when they provide location information. This is an important advantage: researchers can reuse the software without changes with their devices as long as the just use the same interface proving the same context information. This will greatly speed up system and application development and is an issue so far neglected by nearly all existing systems. 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.

2.3 Advanced processing/learning algorithms as Player drivers

In our work, feature extraction is one of the most important issues that we need to address.

Because we deal with so many different types of sensors and application scenarios, a manual annotation and feature extraction process would be very cumbersome.

Therefore, we took advantage of the flexibility of the Player/Stage system, and came up with an automatic feature extraction system, that can take raw data from sensors, and output various features or coefficients calculated through a variety of methods, such as: Principal Component Analysis (PCA), Independent Component Analysis (ICA), Wavelet analysis, Fourier analysis, etc.

The features driver acts as a virtual gateway between the raw data received from the sensors and various different more useful interpretations of it. Based on input configuration options given by the user (full configuration of parameters for each algorithm is possible, without writting a single line of code), the driver spawns as many threads as needed to perform real-time feature extraction and calculus.

An example of a Player configuration file for an automatic acceleration feature extractor driver is given below:

```
driver (
    name "accelfeatures"
    plugin "libaccelfeaturesdriver"
    provides ["features:0" "wsn:1"]
    requires ["wsn:0"]
    window_size 16
    queue_size 10000
    overlapping 50
    feature_list ["wavelet_coeff" "ica"]
    wavelet_params ["daubechies" 20]
)
... ↓ ...
```

```
... ↓ ...
driver (
    name "accelfeatures"
    plugin "libaccelfeaturesdriver"
    provides ["features:1" "wsn:2"]
    requires ["wsn:1"]
    window_size 16
    queue_size 10000
    overlapping 50
    feature_list ["energy" "rms" 11
        "skewness" "magnitude" 15 18]
)
```

In this case, the accelfeatures driver will spawn two different threads, using the output of the first as the input of the second.

Therefore, the first thread will receive acceleration data via the *wsn:0* interface, calculate wavelet coefficients using Daubechies 20 and perform Independent Component Analysis (ICA) in parallel, and finally pack the resulted values in the *wsn:1* interface. The second thread will receive the values via the *wsn:1* interface, and then calculate more standard features such as: energy, rms, mean_deviance (11), skewnewss, magnitude, cross correlation (15) and kurtosis. As can be seen from the example, the feature list can be given as both indexes or string names. For a better understanding of how the features extraction mechanism works, refer to Figure 2.

By providing data through the above mentioned interfaces, the client software can easily access any information from the system, at any given point. Several instances of the same driver can be linked together, so that standard features such as mean, variance, et al., can be calculated from raw data or Fourier coefficients or even the inverted/filtered wavelet transformation of the function, using exactly the same code. The number of features that can be calculated is easily extendable

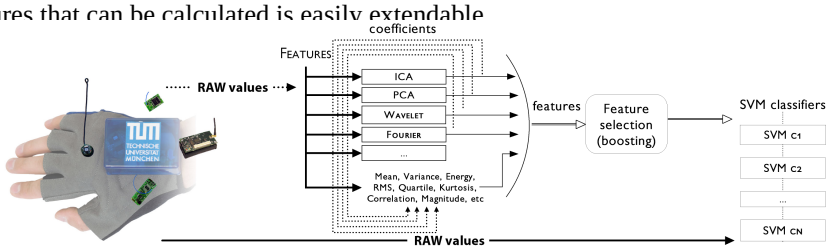

Figure 2. Features Extraction System and its usefulness in learning SVM models

Besides the automatic feature extractions system, we are currently developing an automatic feature selection driver using boosting as well as a semi-automatic learning and classification system using Support Vector Machines (SVM).Preliminary work already shows that we can easily switch between training and testing sessions, using additional sensor data (eg. while the user is holding an RFID-tagged object, record acceleration and angular data and use it to learn a motion blueprint using SVM). As soon

as the above mentioned drivers are ready, we will make them available through the Player/Stage open source project.

3 AwareKitchen

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. Several Hokuyo URG-04LX laser scanners allow us to locate people in the kitchen and follow their movements. Each cupboard door is equipped with a magnetic sensor to determine if it is opened or closed. There are also numerous RFID-Readers that detect the tags placed on the objects of interest, like glasses, pots, and other cooking equipment. One RFID reader is placed on a glove, which allows us to detect which objects are taken or manipulated (see Figure 2).Additionally, a great number of wireless sensor nodes are available, including Particles, RCores, Mica2s, Mica2Dots and Gumstix. The wireless nodes have accelerometers, magnetic sensors, light sensors, and microphones. Placed on objects or people, they offer a very easy and flexible way to collect data for new experiments.

Figure 3. AwareKitchen in the Gazebo simulator (left) and real life (right)

Using Gazebo [4], the whole AwareKitchen environment can be simulated, and thus, develop and test algorithms without using the real hardware. The simulator also provides an easy way to visualize the captured data through the use of the replay virtual Player drivers.

4 Related Work

Phidgets [5] abstract and package input and output devices, hiding construction and implementation details and expose a well-defined API. This is similiar to the interfaces used in Player/Stage, and it's especially needed for fast application development and device substitution. Papier-Mâché [6] enables programmers to develop user interfaces by shielding low-level details. It provides technology-independent input abstractions, e.g. prototyping an UI with computer vision and deploying it with RFID. It also simulates hardware by offering Wizard-of-Oz like generation of input events. The simulation of sensor or context events in a ubiquitous computing system enables the developers to concurrently develop low- and high-level parts of the system and thereby reduce the overall development time. The virtual drivers provided by Player/Stage fulfill exactly this function.

Intille et al. [7] showed the importance of having a real scenario (PlaceLab [2]) for research. This environment may be open to be used by selected researchers, but will not

be used by a greater community. Also, the Mites hardware that was used is currently not available to researchers to enable them to reproduce the results. Therefore, Intille et al. provide annotated log files of relevant sensor events. The Player/Stage platform makes it easy to log all data via a Player interface component. The sharing of huge amounts of sensor and context information among researchers becomes easy and practical. The data can be played back into the system via a virtual driver. This enables researchers to test and develop their algorithms without the need of a real environment - all that is needed is the open source Player/Stage platform.

5 Conclusion and Future Work

We presented Player/Stage, an open source middleware for sensor/actuator systems and demonstrated how ubicomp devices and algorithms can be integrated into it. In our opinion, the overall system is more suitable in comparison to other existing middlewares, and we strongly encourage the ubicomp community to use it.

We are currently developing additional interfaces and drivers to include more wireless sensor platforms, as well as semi-/automatic data processing tools, such as feature selection or learning models from acceleration data. As soon as they are ready, they will be integrated into the Player/Stage project and thus, made available to anyone interested.

The acquired sensor data from our experiments in the AwareKitchen, is also made available to interested researchers, and using a virtual driver component that we developed, can be replayed within the Player/Stage system. By making use of the logged data and simulation components, certain scenarios can be reconstructed in other laboratories, without the need to have the real hardware.

All developed code and interfaces will soon be available as open source to the Player/Stage system for both the robotics and ubicomp communities.

References

1. Kidd, C.D., Orr, R., Abowd, G.D., Atkeson, C.G., Essa, I.A., MacIntyre, B., Mynatt, E.D., Starner, T., Newstetter, W.: The aware home: A living laboratory for ubiquitous computing research. In: CoBuild '99: Proceedings of the Second International Workshop on Cooperative Buildings, Integrating Information, Organization, and Architecture, London, UK, Springer-Verlag (1999) 191–198
2. Schilit, B.N., LaMarca, A., Borriello, G., Griswold, W.G., McDonald, D., Lazowska, E., Balachandran, A., Hong, J., Iverson, V.: Challenge: ubiquitous location-aware computing and the "place lab" initiative. In: WMASH '03: Proceedings of the 1st ACM international workshop on Wireless mobile applications and services on WLAN hotspots, New York, NY, USA, ACM Press (2003) 29–35
3. Collett, T.H., MacDonald, B.A., Gerkey, B.P.: Player 2.0: Toward a Practical Robot Programming Framework. In: Proceedings of the Australasian Conference on Robotics and Automation (ACRA 2005). (2005)
4. Müller, A., Beetz, M.: Designing and Implementing a Plan Library for a Simulated Household Robot. In: AAAI 06 Workshop on Cognitive Robotics, 2006. (2006)
5. Greenberg, S.: 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, New York, NY, USA, ACM Press (2004) 161–161
6. Klemmer, S.R., Li, J., Lin, J., Landay, J.A.: Papier-mâché: toolkit support for tangible input. In: CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems, New York, NY, USA, ACM Press (2004) 399–406
7. Intille, S.S., Larson, K., Tapia, E.M., Beaudin, J., Kaushik, P., Nawyn, J., Rockinson, R.: Using a live-in laboratory for ubiquitous computing research. In: Pervasive. (2006) 349–365