And all the robots merely Players

Geoffrey Biggs, Radu Bodgan Rusu, Toby Collett, Brian Gerkey and Richard Vaughan

***Abstract*—When robot researchers talk about robot middlewares, inevitably Player will enter the discussion. This piece of software and its related tools have become one of the most popular software tools in robotics research. Its range of hardware support and the flexibility it offers users, as well as its ease of use and shallow learning curve, have ensured its success. It is now over a decade since Player was conceived, and yet its popularity continues. In this article, we take a retrospective look at Player. We discuss its origins, the history of its development, and consider why it became so popular.**

***Index Terms*—Robotic software engineering, robot architectures, Player**

I. Introduction

All the world's a *stage*
and all the men and women merely *players*

OVER a decade ago, the Player robot software middleware was released to the world. What at first was just a small project to make the use of one laboratory's robots easier became one of the world's most popular robotics research tools, particularly outside of industrial robotics. The middleware has been downloaded over 100,000 times, while the Stage simulator has over 70,000 downloads and the Gazebo simulator has more than 35,000 downloads[1]. The project as a whole still averages over a thousand downloads every month.

How did this small set of tools created by a few young researchers rise to become such a commonly-used piece of software for developing both real and simulated robots? In this article, we take a look at the history of Player and Stage - where they came from and why. We discuss why Player became and has remained popular for so long, while so many other robot software projects have come and gone. We also discuss the Player project as it stands today, and where it may go in the future. Throughout this article, we will refer to events that are shown in Figure 1.

II. An evolving design

The design of Player has evolved and changed considerably over the years. However, the core principles of the architecture remain the same. It can still perform its original task, acting as a device server, but it can also compete, feature-wise, with the latest component-based architectures. In this section we discuss the evolution of Player's design from its humble origins as a side-project to make one lab's robots easier to use.

A. Origins of a robot architecture

In 1998, the Interaction Lab at the University of Southern California used Pioneer robots from ActivMedia (now known as MobileRobots). These robots ran on the Ayllu middleware [1]. Ayllu is a behaviour-based architecture targeted at distributed multi-robot systems. It builds on the design of single-robot behaviour-based control architectures, in particular the Subsumption[2] style of controller. However, programming robots using it requires following its controller design. This was quite limiting for researchers who wanted to try different controller styles. It also required the use of its own C-like custom language, adding another hurdle for developers.

[1]Statistics obtained from the SourceForge download tracker.

What eventually became Player was developed both to replace this somewhat-restrictive control architecture and to allow controllers written for the new simulator developed in the lab, named "Arena," to be executed on real robots without changes.

At first, controllers written for Arena were compiled in directly, but it soon gained a socket interface. This allowed a separation between the simulator and robot controllers. It also led to Arena's developers writing large pieces of robot controller code for use in Arena that they also wanted to try on the lab's Pioneer robots. Rather than porting their controllers to Ayllu, they developed ArenaServer.

ArenaServer ran on a Pioneer robot and presented the same socket interface as the Arena simulator. It was tied to both Ayllu and ActivMedia code, but it allowed robot control code designed for Arena to run, unchanged, on the lab's real robots.

This experience led the developers of Arena and ArenaServer to design "the generic robot interface of their dreams," based on some simple ideas and UNIX concepts of abstraction:

- Be as unprescriptive as possible
- Present a socket interface to the robot
- Hide all the details of dealing with the hardware
- Be free and open
- Work well with a simulator.

A new server for the Pioneer robots was developed based on these ideas. This time, however, a custom driver was written for interacting with the robot's controller. The new server was named "Golem." An interface between Golem and Arena was created, allowing controllers written for one to work with the other. Golem was soon renamed to Player and Arena to Stage.

B. The early days

The first publication concerning Player appeared in late 2000: the reference manual for version 0.7.4a [3]. At this point, Player was a device server for Pioneer robots, but already many of the concepts that the architecture is still based on today were beginning to develop.

The early Player protocol, shown in Figure 2(a), used a "small set of simple messages" [3] between clients and the server, communicating over TCP. The presence of the protocol

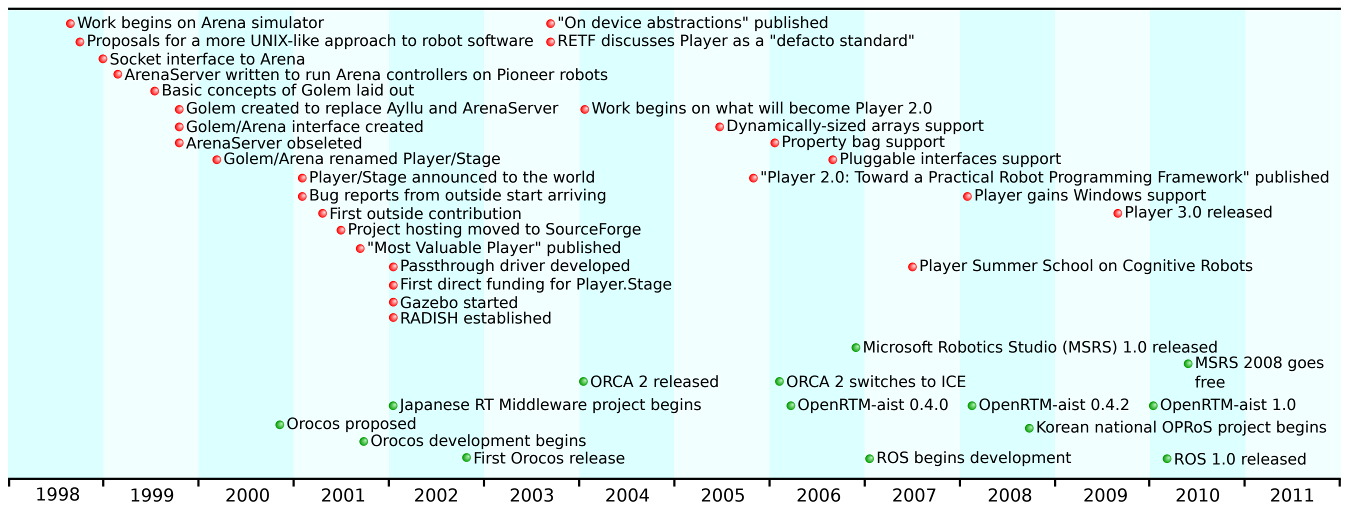


Fig. 1: A timeline showing major points in the development of Player, contrasted with other events in robot software.

even at this point illustrates the emphasis that was placed on the two benefits it brings to robotics: language/operating system independence and location independence. The use of a network connection also granted concurrent access to any number of clients.

The protocol was both simple and Pioneer-centric. In UNIX style, clients first had to request access to a device (a feature that still exists today and can be found in various forms in modern middlewares such as ROS and Orocos). A single letter was used to indicate the device to access, imitating the semantics of the UNIX "open" system call. Seven device types were available, reflecting the hardware of the Pioneer. Command messages were sent by clients, each device expecting its own specific format of command message. To keep the implementation simple, sent commands did not receive replies. Data was sent by the server at a fixed, configurable rate, 10Hz by default. As with commands, the format varied by device type, although there was no true "data message" as we know it in Player today. Configuration messages could be sent to alter parameters of the server or the motor control. Unlike subsequent versions of Player, these did not receive a reply.

The concept of device interfaces can be traced back to this early version of Player. Although tied to Pioneer hardware and not well defined, the concept was beginning to appear. Some messages still survive today, such as the request to the motor device to enable the motors.

The original client library bears little resemblance to the client libraries now included with Player. It did not have separate device proxies; all interaction was through an instance of the `CRobot` class, effectively a robot proxy. This class stored data received from the robot. Command values were written into data members of this class before calling a function that sent all command messages for all subscribed devices. Even at this early stage, though, Player had client libraries in multiple languages. C++, Tcl and Java were provided, with Python under development.

The C++ client library was written as an example; the original intent was for the socket protocol to be the defining interface and that developers would be happy managing their own socket themselves. Over time, it was realised that the client API was just as important to end users.

One of the most important features of Player has always been that it makes no assumptions about the structure of robot control programs. This was a conscious design decision from the beginning. It is perhaps the feature that most contributed to its early popularity. Having struggled with the limitations imposed by Ayllu, the Player developers decided that an open approach using a library-style API, rather than structured controllers, would give researchers more flexibility. Player therefore took a minimalist approach when compared with other architectures of the time, and even most architectures of today. In Player, the researcher is free to create controllers of any complexity and style they choose, from highly-concurrent controllers to simple read-think-act loops.

The design of the Player architecture, shown in Figure 3, was asynchronous, allowing clients and devices to operate at their own speed. The server at this point used a large number of threads: two for each client (one reader, one writer) and one for each active device, as well as the thread listening for new clients.

Communication between clients and devices was via buffers in the server. Each device had two buffers, which stored the latest command and data. If new data or a new command arrived before the old was read, it overwrote the previous contents. This design decoupled devices from clients, allowing all entities to operate at their own rate. Internally, devices communicated by global shared memory.

The design did have its drawbacks. The fixed rate meant that clients could not determine if the data it had just received was new, or a repeat of old data (a "pull" mode was later added to give clients that needed it control over receiving data). The laser scanner operated at 5Hz, for example, meaning laser data was out of date half the time. However, Player was intentionally designed this way for simplicity; it allowed clients to perform blocking reads.

By 2001, the internal server design had changed little. [4] formally defined Player as the protocol, not the server, and introduced the motivation for Player's use of a socket

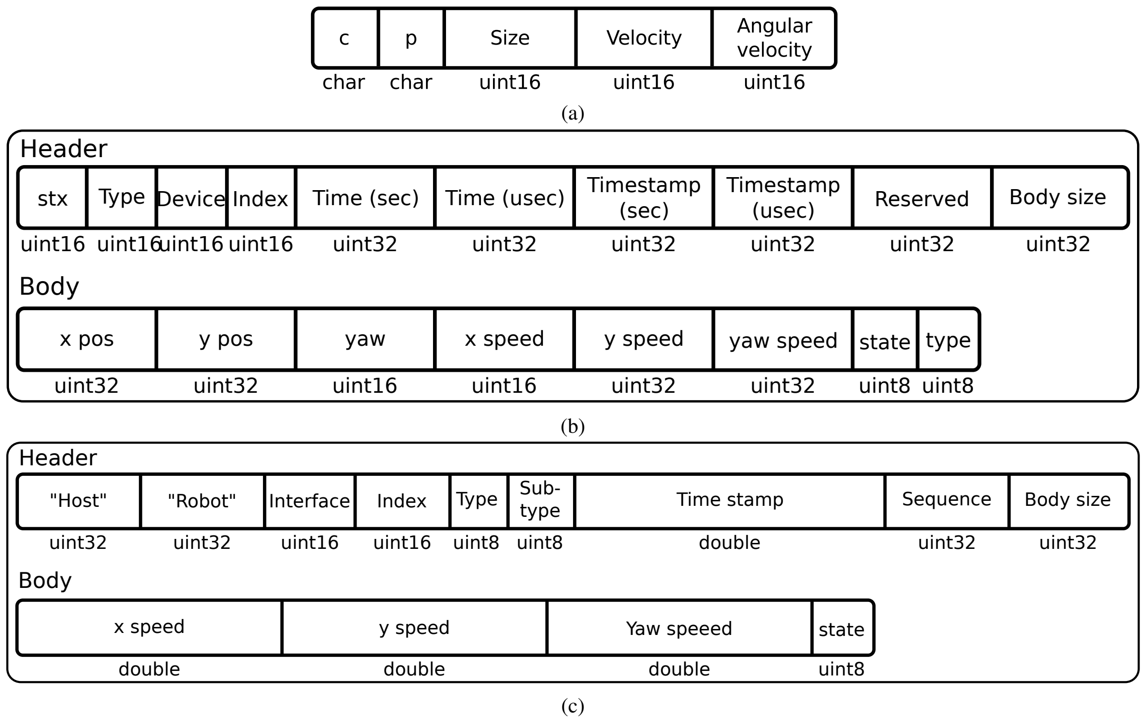


Fig. 2: The Player message structure has changed considerably since the first version. (a) The command to move the robot in Player 0.7.4a. (b) The command in Player 1.65. Note the presence of multiple commands in the message body, with a “type” field to indicate which is being used. (c) The command in Player 2 and beyond. The message header was expanded to allow more flexibility, which in turn simplified message bodies.

abstraction:

Distribution	Clients and servers could exist anywhere and in any number.
Independence	Clients could be written in any language on any operating system supporting sockets.
Convenience	Clients just had to connect and ask for access to the devices they were interested in to start receiving data.

The concept of virtual devices was first explicitly mentioned in 2001. A device could be pure software, taking data from a hardware device and transforming it, before presenting it to clients. This concept allowed algorithms to be widely shared.

In 2001, Player was beginning to grow in popularity. Compared with the state of the art at the time, Player took a radically different approach. Architectures focused on creating a development environment that suited a particular control philosophy. For example, Ayllu enforced a concurrent behaviour-based control structure [1]. COLBERT/Saphira similarly focused on behaviour-based control, using fuzzy blending [5]. Architectures at this time often restricted the programmer to a specific language designed for the architecture, such as Ayllu’s C-like language. It was also still common for research robots to come with an unmodifiable library for control, with researchers working within the constraints of this library.

C. Player 1

As Player moved through version 1, it gained an important feature: interfaces. The first clear statement on the separation of protocol, interfaces and implementation was made in a publication in 2003 [6]. This paper described the application of three well-known abstractions to robotics:

Character devices	Based on the UNIX concept that everything is a file. Robot devices provide and consume streams of data.
Interface/driver	Interfaces define the contents of the character streams. Drivers implement one or more interfaces.
Client/server	Provides location, language and controller design independence, and allows concurrent access to devices.

Through its abstractions, Player concentrated the “essential qualities” of robot devices.

The abstract interfaces grew out of extending Player’s hardware support from just Pioneer robots to also include robots from RWI. The new drivers re-used some of the structures from the Pioneer drivers. This indicated the overlap between robot device interfaces and showed that, with suitable abstractions, a robot controller could run *unchanged* on different robot platforms (RWI support initially had its own client-side support).

[6] also formalised the Player Abstract Device Interface (PADI), which by this point had grown to 29 interfaces. The Player protocol was now a well-defined, flexible, message-based protocol (shown in Figure 2(b)), and has remained message-based until the present day.

Many parts of the architecture were unchanged from earlier versions. The single data and command buffers per device,

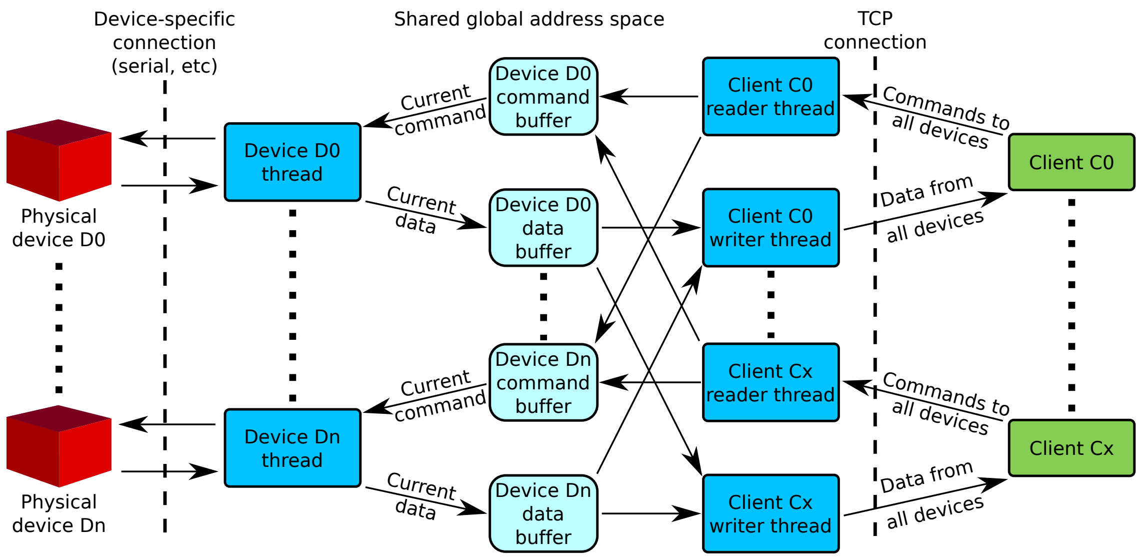


Fig. 3: The architecture of Player for versions prior to 2.0.

for example, were still intact. Drivers had been modularised, allowing unneeded drivers to be left out when compiling Player to reduce the binary size. This is especially important for embedded systems.

Virtual drivers were beginning to appear in greater numbers. Fiducial detectors and Monte Carlo localisation drivers were some of the first [7].

By version 1.3, in 2003, client libraries existed for C, C++, Tcl, Python, Java and Common Lisp, and separate device proxies had been added.

D. Player 2

Player 2.0, developed in 2004, was a major overhaul. It moved Player from a client/server-based model to a more flexible publisher/subscriber model, simplifying the architecture from the complex threads and buffers of Player 1 [8].

The new design, illustrated in Figure 4, attempted to meet several stated goals, as laid out in [9], including enhanced scalability, development process simplification, and transport independence. It had to do this while still maintaining compatibility with the by-now extensive code-base developed by Player's large community of users.

The motivations for the overhaul were many:

- Player was being used in an increasingly wide range of scenarios;
- There was an increasing number of virtual drivers, which Player was not originally designed to support;
- The community was demanding more flexibility;
- The number of contributors was growing, yet the API (particularly for developing drivers) was difficult to use and data marshaling was not robust;
- The client/server model was restricting distributivity support;
- The single data and command message types per interface were limiting;
- Only one transport was available, and it wasn't always the best choice.

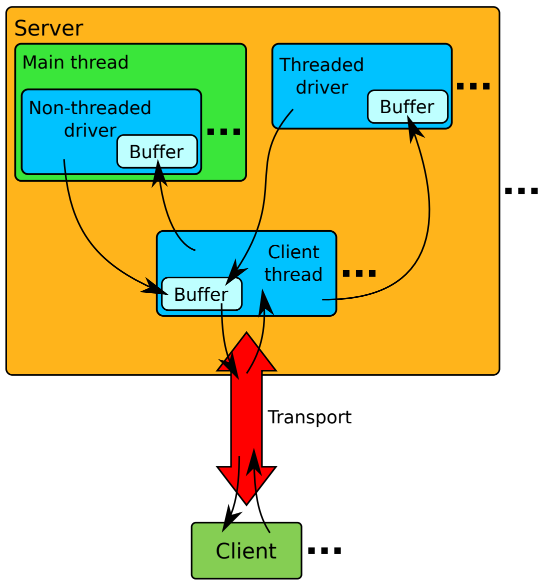


Fig. 4: Player 2's message-queue-based architecture.

The overhaul redefined drivers as software components that can produce and consume messages.

To meet the goals, several changes were made to the architecture:

- The client/server model was replaced with a publisher/subscriber model. Client/server was preventing more general arrangements of servers and clients distributed across a network. In particular, the ability for servers to talk to each other was added;
- The transport was split out of the core, with a well-defined API, allowing multiple transports to be implemented (although only TCP and UDP were provided);
- Data marshaling was made the responsibility of the transport, not the sender or receiver, and implemented using automatically-generated XDR functions;
- The message format was improved, allowing multiple

command and data messages per interface. Interface design was greatly simplified by this new feature;
- The fixed data rate was removed, allowing devices to send data as fast as they liked;
- The driver API was simplified, using a callback model.

The internal operation of the server was reworked to use generic message queues rather than the single data and command buffers per device. Each device and each client got its own message queue from which it could read at its own pace. This was an important change. It increased the flexibility of the server. Drivers could talk to each other by publishing in each other's queues. This could even occur between servers, due to the expanded address format.

The use of XDR for marshalling was first proposed in [6], and saw its first appearance in Player 2.0. This removed a common source of errors when developing with Player, both for driver and client writers.

At the time of Player 2.0's release, many more architectures were being developed and released using a variety of approaches. IPC and the Task Description Language extended C++ for task management [10]. CARMEN, based on IPC, provided a navigation toolkit [11]. CLARAty [12] used a two-tiered design, while CoolBot [13] and ROCI [14] provided component-based models.

While the above architectures, like Player, developed nearly everything themselves, others began using of-the-shelf software. Orocos [15], MIRO [16] and OpenRTM-aist [17] built on top of CORBA. ORCA2 [18] used ICE.

The trend at this time was towards more and more frameworks being developed and released. This trend has arguably not subsided today. MARIE [19] appeared at about this time, seeking to unite the frameworks.

E. Player 3

After the release of Player 2, development of Player began slowing down. Newer architectures were appearing, and the developers were beginning to move on to other interests. Despite this, Player gained important new features during this time and continued on to a version 3 release in 2009. New features added in Player 3 included:

- Pluggable interfaces, allowing users to create new interfaces and compile them as shared libraries;
- Support for dynamically-sized arrays in the PADI. This was a major improvement that finally freed it of device-created limitations on data sizes, the most common of which being the regular need to increase the maximum size of a camera frame to support higher-resolution cameras;
- Property bag support was added to provide a more device-specific form of support for the `ioctl`-style calls described in [6]. Property bags allowed individual devices to expose settings to subscribers such as baud rates that are not contained in an interface;
- Windows support for the server (at last).

When Player 3 was released, it featured 44 interfaces. 65 interfaces have existed in total since Player began, with some being deprecated and removed over time. Player currently provides over 150 different drivers covering the full range of robot hardware and software.

III. Behind the scenes

While we have described the evolution of Player's design, it is worth considering the environment in which the design originated and evolved[2].

The original creators of Player are Brian Gerkey, Andrew Howard, Kasper Stoy, and Richard Vaughan, all members of the Interaction Lab at the University of Southern California. This lab was mainly focused on multi-robot systems.

It was Brian and Richard who, in 1998, grew frustrated with Ayllu's restrictions and developed the ideas that would become Arena and Golem, such as using UNIX concepts of abstraction. Golem was designed by Brian, Kasper and Richard, based on the experience of ArenaServer, and implemented by Brian and Kasper, while Richard concentrated on Arena. At this point, the basic functionality of both Player and Stage were in place.

Andrew Howard arrived in 2000 and joined Brian and Richard in designing the interface/driver interaction framework that still underlies Player today. By this point, the majority of the Interaction Lab and Robotic Embedded Systems Lab robots at USC were running Player, and students were regularly using Stage in their work. Player/Stage was announced on the Interaction Lab's website, and bug reports started arriving from people outside the lab.

In 2001, Player/Stage was moved to its present home at SourceForge[3], making it independent of the lab, and the first contribution from a third party, the University of Massachusetts, arrived. It was in heavy use at the University of Southern California by now. In 2002, Player/Stage received direct funding and a three-dimensional simulator, Gazebo, was started by Andrew Howard and Nate Koenig. By late 2002, Player was in use in over a dozen institutes [7], and was being used in both graduate and undergraduate classes. By mid-2003, this had grown to over 20 institutes [6].

Toby Collett, who was responsible for much of the architecture overhaul in Player 2.0, joined the Robotics Research Group at the University of Auckland in 2003. After growing frustrated with the by-then antiquated system software for the group's RWI B21r robot[4], Toby began looking for replacements. Player already had a driver for the B21r, and so Toby chose to use Player to drive the robot.

However, problems soon surfaced in the design of the Player protocol when Toby realised that the B21r's sensor geometry would change every time the robot turned. Overcoming this was not a major challenge, but in order to make things easier in the future, Toby undertook a major overhaul of Player's internal architecture in 2004. This led to the release of Player 2.0 and earned him a place as a core Player developer. Geoffrey Biggs, from the same lab, soon joined him, eventually becoming the lead developer for the release of version 3. Toby

[2]Much of this history can also be found on the Player website, at http://playerstage.sourceforge.net/wiki/PlayerHistory

[3]See http://playerstage.sourceforge.net

[4]The final straw was the discovery that it would only function on Redhat 6.2, which had too-old a version of GCC for other software desired.

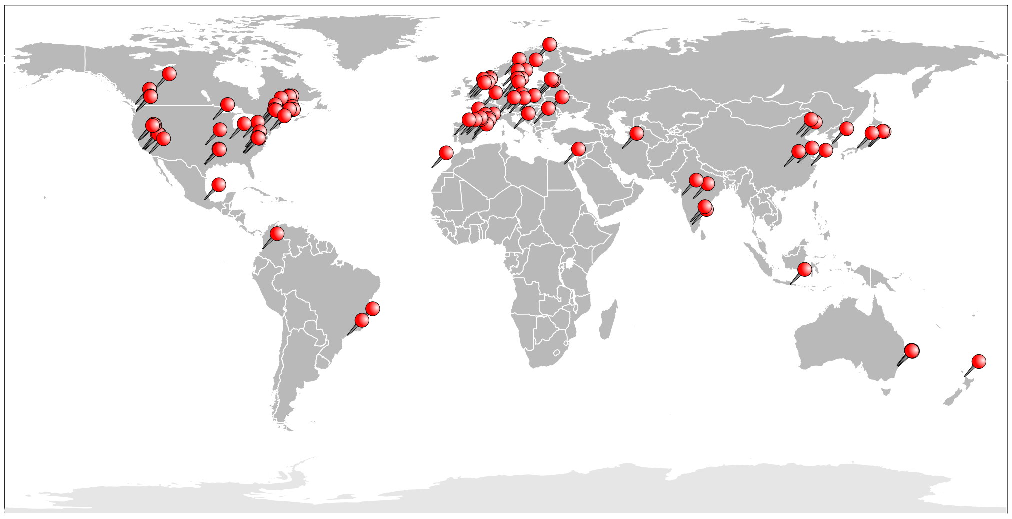


Fig. 5: The known locations of Player users; there are likely to be many more. Player has been used in major universities and research institutes all around the world.

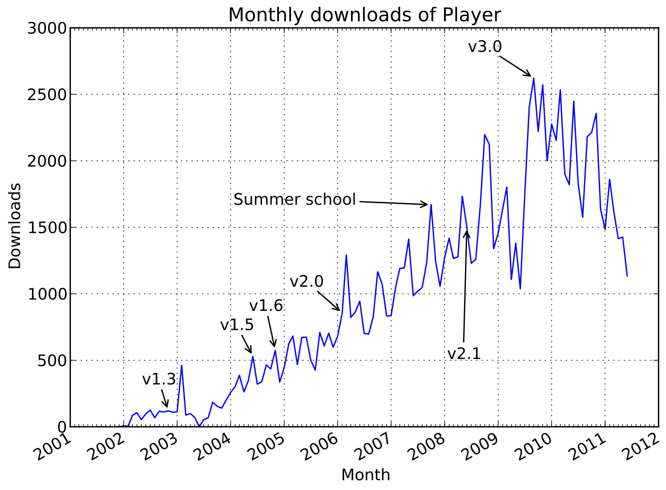


Fig. 6: The number of downloads of Player from SourceForge.

and Geoffrey passed the torch on to Rich Mattes, from Penn State University, in 2010.

IV. Making an architecture a success

Since its first release, Player has been used all over the world (see Figure 5), with a large number of downloads (see Figure 6). Initially, the factors that led to Player's popularity were flexibility, community, and Stage.

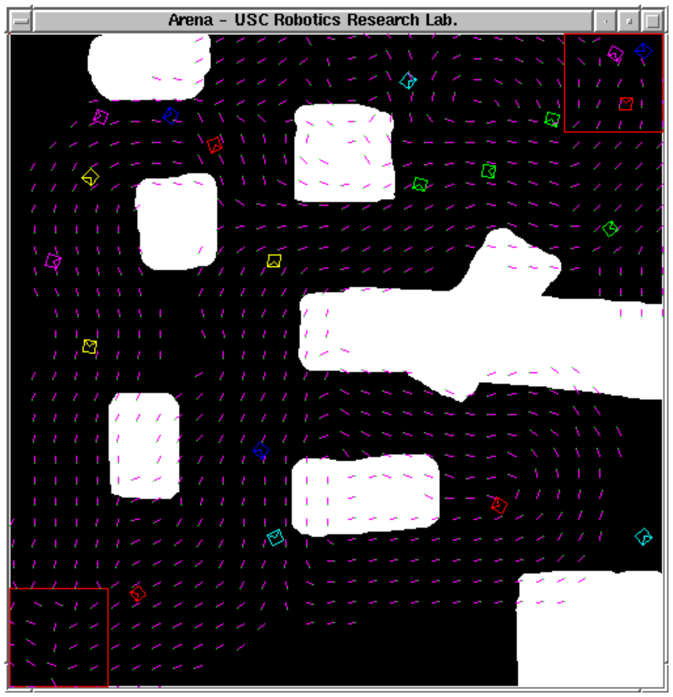
Player allowed greater flexibility in controller design. Its contemporaries promoted "One True Way of Coding," forcing the programmer into a particular controller structure. Player allowed the programmer to choose their own structure. Player also supported a variety of languages, while its contemporaries were typically limited to one, often custom-designed, language. Over the years, Player has supported C, C++, Java, Tcl, Python, Common Lisp, Matlab, and more.

It is recognised amongst open-source projects that a strong community is the key to a successful project. Player's developers cultivated a community around the project. Brian Gerkey was notable on the mailing lists for quickly answering any questions. The project as a whole was very open, with submitted patches quickly accepted into the source, irrespective of their origin, coding style, or even if they worked perfectly ("any code is good code"). The European summer school on Player was arguably a major driver in Player's growth in Europe. Player's strong community, which continues to this day, has allowed it to survive three generations of developers.

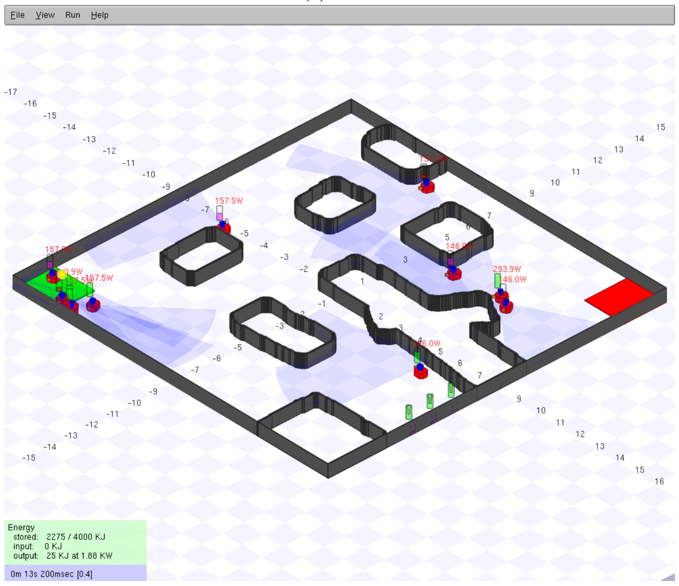
Stage was undoubtedly a major factor in Player's initial and continuing success. By providing integrated support for a robot simulator, Player greatly increased its applicability and usefulness. The appearance of the 3D Gazebo simulator in 2002 reinforced this. Stage, often assumed to be an offshoot of Player, actually predates it, as the Arena simulator (see Figure 7(a)) and was one of the inspirations for Player's development. While the models in stage began by closely emulating Pioneer robots, as Player's interfaces became more generic, so too did Stage's models. They proved to be a useful test of new interface designs, as implementing the complete interface in a Stage model revealed ambiguities and contradictions [6]. This helped improve Player for its users. Stage has changed much over the years. It is now a very powerful 2.5D simulator. Figure 7(b) illustrates Stage as it is now.

The wide range of hardware support in Player is another factor in its success. When looking for software for a robot, software that already works is more likely to be chosen. When users then added additional drivers for their extra hardware, Player's support grew more, further attracting new users.

Not every aspect of Player was a success. New technical features were sometimes ignored by the user community. For example, both the driver properties API and driver capabilities API were largely ignored. This may, in part, have been due to insufficient publicity about the features. Player also attempted to remove the choice between "push" and "pull" data modes, only to add it back at the last minute prior to Player 2's release, upon finding that it introduced a serious limitation.



(a) 2000



(b) 2010

Fig. 7: (a) The Stage simulator as it appeared in 2000, when it was still named "Arena" (from [20]). (b) Stage as it appears today, showing its 2.5D capabilities.

V. The future of Player

Player has been through two generations of main developers and a major design overhaul since it began, and development of major new features is slowing. The recently-released Player 3 does not feature any major new architectural changes. However, it continues to be supported by its current maintainer, Rich Mattes, with regular bug fixes and enhancements to drivers and interfaces. New features are still added; in one recent example, a complete scripting interface was added to the server.

Although several new middleware projects have appeared that follow a much more distributed component model, Player is still capable of providing the same concepts. Drivers can be treated as components, and Player servers provide the communications middleware and deployment control.

While it may seem that Player's popularity is waning, it still fills an important niche in robot software, and is still a popular piece of of robot software. Player 3, released in September of 2009, has over 15,000 downloads as of mid-2011. A recent survey by Robotics Business Review showed that Player is still one of the most popular tools in robotics, and is likely to continue to be so for some years [21]. The mailing list remains active, regularly receiving mail from new users requesting assistance.

Player's future most likely lies in education. Its simple programming model is well-suited to beginner and young programmers working with simple robots. Encouraging an interest is important to programming education, and if LEGO Mindstorms [22] has shown us anything, it is that robots make programming interesting.

VI. Conclusions

The first paper published about Player ended with the following statement:

> The acid test of Player will be its uptake. We hope that Player will develop over the next few years into a well-used tool. It will not suit every application, but it has proved useful in a variety of roles in our labs. [4]

We believe that Player has undoubtedly passed this acid test.

From humble beginnings as an attempt to make a lab's robots more usable, Player has grown to become the one of the most popular robot software systems outside of industrial robots. Player solved a problem in robot software that no other architecture of the time did.

Player's success can be attributed to its flexibility, which gave researchers the freedom they needed, its ease-of-use, allowing even beginner programmers to use it, the Stage simulator, and most importantly, its community support. The project is still a popular tool in robotics research today.

VII. Sidebar: Player in education at The University of Auckland

By Bruce MacDonald.

We have used Player in our robotics laboratory since 2004. We found from the start that Player was quite accessible to graduate students, reducing the time to come up to speed in programming any particular robot, and made it much easier for a student to take over a project from someone who had finished their study. Once the graduate students were comfortable with Player, we introduced it in the Robotics and Intelligent Systems course of the final undergraduate year. We ask these students to create a simple client for a following, foraging, manipulation, or navigation task. Students initially use Stage to test their clients. We use seven Pioneers, each with a simple arm, so that students can have some practical experience programming real robots. As a result they are prepared to take up graduate study without any delay in learning how to program our robots. About 70–80 students take the course each year, with backgrounds in several different engineering disciplines. The organisation of Player has encouraged students to improve the code base and to contribute ideas back. One group in their final year project improved the 2.5D version of stage, which illustrates how accessible the project is. Player has enabled a strong educational programme in robotics, with an excellent progression to research projects.

VIII. Sidebar: Player's role for European Academics

By Radu Bogdan Rusu.

We started using Player around 2004 for student projects, performing experiments in Stage with multi-agent system architectures. Its dedicated team of developers and their insatiable need to help newcomers made it very appealing for a lot of users to transition to becoming contributors, as they found the Player project compatible with their own needs. There was simply nothing of that kind in robotics at that point: a friendly, talented, team of international researchers and engineers working together on making robots useful, without any boundaries on how the project's capabilities could be extended or what the project could be useful for.

Some of the first contributions from Europe came in the form of Player client libraries, with the Java client being one of the stronger contributions. Later on, advances in heterogenous sensor networks for robotics applications and better 3D sensing devices led to important contributions to Player in the form of new interfaces and drivers [23].

The Player Summer School took place in Munich, Germany, at the Technische Universitaet Muenchen (TUM), in the autumn of 2007. Being co-organized by TUM and SRI International, and with sponsorship from the European Robotics Network (EURON), the event was a major success. It brought together more than 50 students from all around the world for a 7-day marathon through Player for Cognitive Robotics Research. We had the pleasure of being able to assemble the entire Player core development team in one place, together with a list of top worldwide scientists and researchers, and cover topics from navigation to higher level reasoning, with a stop at Munich's famous Hofbrauhaus brewery in the middle.

It was after 2007's summer school when Player began getting an amazing response from the European community, with many contributions and fresh ideas flowing in.

References

[1] B. B. Werger, "Ayllu: Distributed port-arbitrated behavior-based control," in *Distributed Autonomous Robotic Systems*, L. E. Parker, G. Bekey, and J. Barhen, Eds. Knoxville, Tennessee: Springer-Verlag, October 2000, vol. 4, pp. 25–34.

[2] R. Brooks, "A robust layered control system for a mobile robot," *Robotics and Automation, IEEE Journal of*, vol. 2, no. 1, pp. 14 – 23, Mar 1986.

[3] B. Gerkey, K. Sty, and R. T. Vaughan, "Player robot server," Institute for Robotics and Intelligent Systems, School of Engineering, University of Southern California, Tech. Rep. IRIS-00-392, November 2000.

[4] B. P. Gerkey, R. T. Vaughan, K. S. y, A. Howard, G. S. Sukhatme, , and M. J. Mataric, "Most valuable player: A robot device server for distributed control," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2001)*, Wailea, Hawaii, October 2001, pp. 1226–1231.

[5] K. Konolige, "COLBERT: A language for reactive control in sapphira," *Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)*, vol. 1303, pp. 31–52, 1997.

[6] R. T. Vaughan, B. P. Gerkey, and A. Howard, "On device abstractions for portable, reusable robot code," in *Proceedings of the 2003 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems (IROS03)*, vol. 3, Las Vegas, Nevada, October 2003, pp. 2421–2427.

[7] B. Gerkey, R. T. Vaughan, and A. Howard, "The Player/Stage Project: Tools for multi-robot and distributed sensor systems," in *Proceedings of the 11th International Conference on Advanced Robotics (ICAR 2003)*, Coimbra, Portugal, June 2003, pp. 317–323.

[8] T. Collett, B. MacDonald, and B. Gerkey, "Player 2.0: Toward a practical robot programming framework," in *Proceedings of the Australasian Conference on Robotics and Automation*, University of New South Wales, Sydney, Australia, December 5–7 2005.

[9] Y. hsin (Oscar) Kuo and B. MacDonald, "A distributed real-time software framework for robotic applications," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA05)*, Barcelona, Spain, April 2005, pp. 1976–1981.

[10] R. Simmons and D. Apfelbaum, "A task description language for robot control," in *Intelligent Robots and Systems, 1998. Proceedings., 1998 IEEE/RSJ International Conference on*, vol. 3, 1998, pp. 1931–1937.

[11] M. Montemerlo, N. Roy, and S. Thrun, "Perspectives on standardization in mobile robot programming: the Carnegie Mellon Navigation (CARMEN) Toolkit," in *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, vol. 3, 2003, pp. 2436–2441.

[12] I. Nesnas, R. Volpe, T. Estlin, H. Das, R. Petras, and D. Mutz, "Toward developing reusable software components for robotic applications," in *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, vol. 4, November 2001, pp. 2375–2383.

[13] A. Dominguez-Brito, D. Hernandez-Sosa, J. Isern-Gonzalez, and J. Cabrera-Gamez, "Integrating robotics software," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2004)*, April 2004, pp. 3423–3428.

[14] L. Chaimowicz, A. Cowley, V. Sabella, and C. Taylor, "ROCI: A distributed framework for multi-robot perception and control," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, 2003, pp. 266–271.

[15] P. Soetens and H. Bruyninckx, "Realtime hybrid task-based control for robots and machine tools," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2005)*, Barcelona, Spain, April 2005, pp. 260–265.

[16] H. Utz, S. Sablatnog, S. Enderle, and G. Kraetzschmar, "Miro - middleware for mobile robot applications," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 4, pp. 493–497, August 2002.

[17] N. Ando, T. Suehiro, K. Kitagaki, T. Kotoku, and W.-K. Yoon, "RT-middleware: distributed component middleware for RT (robot technology)," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2005)*, 2005, pp. 3933–3938.

[18] A. Brooks, T. Kaupp, A. Makarenko, S. Williams, and A. Oreback, "Towards component-based robotics," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2005)*, 2005, pp. 163–168.

[19] C. Cote, D. Letourneau, F. Michaud, J.-M. Valin, Y. Brosseau, C. Raievsky, M. Lemay, and V. Tran, "Code reusability tools for programming mobile robots," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2004)*, vol. 2, 2004, pp. 1820–1825.

[20] R. T. Vaughan, K. Støy, G. S. Sukhatme, and M. J. Matarić, "Whistling in the dark: cooperative trail following in uncertain localization space," in *Proceedings of the Fourth International Conference on Autonomous Agents*, Barcelona, Spain, June 2000, pp. 187–194.

[21] (2010) Tools, Standards, and Platforms for Commercial Robotics Development: An Adoption Profile. [Online]. Available: http://www.roboticsbusinessreview.com/articles/newsletter_view/tools-standards-and-platforms-for-commercial-robotics-development-an-adopti/

[22] (2006) LEGO.com MINDSTORMS Robotics Invention System 2.0. [Online]. Available: http://mindstorms.lego.com/eng/products/ris/index.asp

[23] R. B. Rusu, B. Gerkey, and M. Beetz, "Robots in the kitchen: Exploiting ubiquitous sensing and actuation," *Robotics and Autonomous Systems*, vol. 56, no. 10, pp. 844–856, 2008, network Robot Systems.