The 2010 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 18-22, 2010, Taipei, Taiwan

Comparing Temporally Aware Mobile Robot Controllers Built with Sun's Java Real-Time System, OROCOS's Real-Time Toolkit and Player

Andrew McKenzie, Daniel Gay, Rahul Nori, James Davis and Monica Anderson

***Abstract*— Designing a robot controller that can optimally manage limited resources in a deterministic, real-time manner can be challenging. Behavior-based architectures, which split autonomy into levels, are very popular but neither have real-time features that enforce timing constraints nor support determinism. Even though real-time features are not included, it seems like a natural fit to make each level in the behavior-based architecture its own task or process. The only that thing that it lacks are the timing features. This has already been implemented using Suns Java Real-Time System. It has also been shown that timing constraints effect performance. This brings us to the question; why not use the more traditional language of C or C++ to implement this behavior-based real-time architecture? Are we not taught that Java is useful but slow compared to C and C++? If so why not use C++ and the features of Open Robot Control Software (OROCOS) to implement the architecture.**

This paper answers the question of does it really matter what language is used in a behavior-based real-time architecture. We implemented the architecture using OROCOS/C++ running on UBUNTU. Then compared our implementation to two other implementations of the architecture: Java/Player on Fedora; and Suns Java Real-Time System (RTS) on Solaris. Results, from experiments on a robot, show that our OROCOS/C++ implementation performed similarly to the Java RTS implementation. Both the OROCOS and Java RTS implementations performed better than the Player/Java implementation. This suggests that Java is in fact feasible for a behavior-based real-time robot architecture but it needs to be run using Java RTS not the regular version.

I. INTRODUCTION

Mobile robot controllers must be designed to concurrently handle low level and high level tasks. This leads to an added complexity in robot controller design. Behavior-based controllers have prioritized the layers of complexity by using finite state machines [1]. Although these types of behavior-based architectures are robust, real-time issues are not considered in these architectures. In certain cases, tasks need to be completed with timing constraints. Traditional real-time system development establishes frequency requirements for individual subcomponents. In behavior-based systems, the frequency would be based on the input availability according to the laws of control theory [2]. Running all behaviors as fast as possible is not sufficient because resource intensive behaviors may require more resources and potentially slow and/or block other time sensitive behaviors.

This work was supported in part by the following NSF grants: IIS-0846976, CCF-0829827 and CCF-0851824.

A McKenzie,R Nori and M Anderson are with the Department of Computer Science; D Gay and J Davis are with the Department of Electrical and Computer Engineering; University of Alabama, Tuscaloosa, Alabama 35487 USA (email: {awmckenzie, djgay, rnori, jedavis}@crimson.ua.edu; anderson@cs.ua.edu).

Typically, C or C++ is used for real-time system development because of their performance and availability of tools. However, using C or C++ poses issues with memory. It has been found that students using Java to program real-time systems performed better and enjoyed it more than other real-time modules [3]. Unfortunately the traditional garbage collector in Java can interjects a source of nondeterministic behavior. Sun's Java Real-Time System (Java RTS) is designed to have deterministic and real-time features without requiring developers manage memory [4]. Consequently, Java RTS includes a real-time garbage collector (RTGC) that provides a deterministic approach to the traditional garbage collector.

In [5], a real-time aware mobile-robot architecture [6] was mapped to Java RTS in order to show that timing affects performance. The proposed architecture was not an actual real-time system, rather it uses real-time awareness to manage timing requirements. In order to be classified as a real-time system, the proposed architecture needs to run on a real-time operating system (RTOS). Therefore, in [5] the robot architecture was rewritten in Java using Java RTS while running on a RTOS, such as Solaris. The results showed that the prevailing approach to execute behaviors "as fast as possible" is not ideal. Higher periods give behaviors more time to complete but can also cause system instability if too low.

In this paper, we compare the performance of the temporal aware mobile robot control architecture in [6] and the real-time mobile robot control architecture in [5], to the same architecture written in C++. We rewrote the proposed architecture in C++ using OROCOS [7]. OROCOS is an open source modular framework for robot and machine control. The performance of the architecture in [5] is experimentally validated against the C++/OROCOS implementation. We hypothesize that the Java RTS architecture will perform, in terms of total time to complete a given task and distance from the goal, similarly to the OROCOS architecture and the Java/Player architecture [6] will be worse than both. In Section 2, background and previous work is presented. Our approach is presented in Section 3. Section 4 describes the experimental setup. The experimental results and the analysis are presented in Section 5 and 6 respectively. The conclusion is presented in section 7.

II. RELATED WORK

A. Real-time in mobile robotics

One could claim research in mobile robotic architectures is popular, considering the various development platforms that

have come into existence. But when it comes to providing real time features, they either don't provide declarative frequency specifications or timing is managed solely by priority. Implementation of real-time specifications in robotic applications was initially done by Buttazzo et al [8]. Their architecture that was written in C and encapsulated with a set of library functions that contained four hierarchical layers. The layers were: action layer, control layer, communication layer and an interactive hard real-time kernel (HARTIK). Even though they had no experimental results, they claimed that their architecture was flexible enough for implementing new robotic applications.

Brega et al. [9] performed a case study of the XO/2 operating system, a RTOS, using the Oberon-2 programming language on a Pygmalion Robot. They presented the necessities and requirements of real time control of robots in research, education and the real world. They realized autonomy, real-time and safety could not be quantified in the mobile robots, which needed advanced requirements for hardware and software because of the increasing complexity of mobile robots. They concluded by saying that safe composition of software modules, type-safety, deadline-driven scheduling and automatic memory reclamation mechanisms can relieve the application programmer from many time-consuming implementation issues, while raising the safety-bar.

Auerbach et al. [10] built a helicopter, JAviator, to test a real-time Java application that made use of Exotasks, programming construct that achieves deterministic timing. Though time portability on different platforms was showed, most of the data was processed off-board and they could observe only a single behavior (hovering of the helicopter).

McKenzie et al. [6] proposed and implemented a real-time aware mobile robotic architecture. The architecture used Player [11] to interface to the robot hardware and Java for the control code. It was real-time aware by creating a monitor module that scheduled tasks and reported missed deadlines. Their architecture ran on Fedora, which is not a RTOS. In order to be fully real-time the architecture would need to be run on a RTOS in a real-time compatible language.

B. Real-time in non-mobile robotics

Stewart et al [12] created Chimera II, a real-time architecture based on port based objects. Port based objects account for input, output and resource ports. They are also fully competent software components that include both states and methods. It is assumed that an input can be linked with only one corresponding output and that all similar outputs are combined into one precise output. Task scheduling is manageable, which can in turn stabilize the system and reduce resource consumption.

Bruyninckx et al. [13] used the hard real-time motion control core of the OROCOS project [7] to implement a real-time motion (velocity and hybrid force) control on a KUKA 361 six-DOF robot 8. The code was written using only OROCOS bypassing the controller that KUKA traditionally inherits. It runs on the RealTime Application Interface for Linux in hard real-time and without real-time performance on Linux.

Using Java RTS on a RTOS, Robertz et al. [14] implemented a motion control system for an industrial robot. They used non-real time threads for non-critical tasks and Java RTS's real time threads for critical tasks. The real time threads were run at the highest priority and the non-RTTs were run at a lower priority. They only measured network delays. They did not give any results showing the real-time features effects on the overall system.

C. Behavior-based architectures

Subsumption-based robotic controllers build autonomy out of layers of behaviors [15]. Basic layers are complete processing units that take input and provide appropriate output. We are particularly interested in properties that enable temporal decompositions. Aria [16] is a robotic architecture that provides support for behavioral decompositions. Aria programs are composed of individual behaviors that are each given a priority that is applied to the importance of actuator output. Priority does not affect resource assignment or utilization. Because the target hardware architecture only processes commands every 100ms, all behaviors are run every 100ms. Other threads can be added by the developer outside of the framework for processing tasks that have different frequency requirements.

Carmen [17] is an open source modular architecture for mobile robot control. It features a three-tier architecture, where the first layer is the hardware interface, the second layer contains the basic robot tasks, and the final layer is the user-defined application. It uses inter-process communication to provide functionality for mobile robots and supports various platforms.

OROCOS [7] is an open source modular framework for general purpose robot and machine control. It supports four C++ libraries: the Real-Time Toolkit, the Kinematics and Dynamics Library, the Bayesian Filtering Library and the OROCOS Component Library.

Although not a behavior-based architecture, Player [11] is an open source robot architecture designed to operate with a wide range of hardware components. The Player architecture specifically avoids defining decompositions and therefore does not provide a framework for adding frequency for behaviors.

III. Approach

Our C++/OROCOS control architecture (Figure 1) closely mimics the structure of [5]'s Java RTS based architecture. In fact, to produce this implementation, we converted the Java RTS code class-by-class to a functional equivalent in C++. The most notable difference in the two implementations is the method through which real-time scheduling is facilitated. Where the Java implementation used Java RTS's Real Time Threads and the Java RTS scheduler, the C++ implementation used the real-time libraries of the OROCOS project. The OROCOS Real-Time Toolkit (RTT) is similar enough to Java

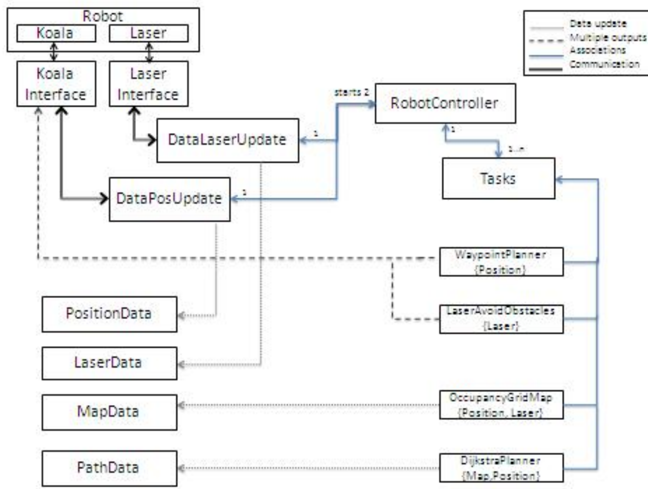

Fig. 1. Framework for the robot architecture.

RTS as to not require any structural changes to the existing architecture.

With the OROCOS RTT, we created a C++ version of the Java class TASKS where, similar to the Java implementation, real-time scheduling was operated by a proven, pre-written library. Similar to Java RTS's real time threads, OROCOS has its equivalent PERIODICACTIVITY class. One main difference is that OROCOS uses the same TIMERTHREAD to execute all instances of PERIODICACTIVITY that have the same period and priority. In this case the periodicActivities would be executed one after another. The programmer needs to be aware of this. If all tasks are set to have the same priority and period then those tasks will run sequentially. This defeats the purpose of using threads in the first place, and the resulting system is not necessarily real-time. Because of this, we had to assign the tasks different priorities. The same priorities were used in the Java implementation.

The program structure has six key components/task threads. Table I summarizes the six threads. Note that only four of the six tasks are actual behaviors. The other two, DATAPOSUPDATE and DATALASERUPDATE, are explicit behaviors because they manage sensor readings and send movement commands to the robot. The KHEPERA class takes requests from WAYPOINTPLANNER and LASERAVOIDOBSTACLES then determines what movement commands are sent to the robot. The motor control order lets safety related activities, such as stall recovery and have the first opportunity to control the motors. If both WAYPOINTPLANNER and LASERAVOIDOBSTACLES request motor control, LASERAVOIDOBSTACLES' commands take higher priority because it contains both obstacle avoidance and stall recovery behaviors. Only when LASERAVOIDOBSTACLES does not want motor control does WAYPOINTPLANNER get control.

IV. EXPERIMENTAL SETUP

Experiments were performed to compare our C++/OROCOS based real-time robot controller to a Java based real-time robot controller [5] applying different frequency requirements to each controller's behaviors. The frequencies were supplied by user input via the command line. Both controllers used a waypoint navigation controller to measure the effects of frequency and implementation on task performance.

Physical experiments employed a K-Team Koala robot. The robot was equipped with an Acces I/O ETX-Nano computer which has an Intel Core Duo 1.66GHz processor, 2GB of RAM and an 8GB compact flash card for storage. The Java RTS controller uses Sun's Solaris 10 RTOS, which requires a dual core processor [20]. The C++/OROCOS controller runs on UBUNTU (a non RTOS) using the OROCOS RTT version 1.8.5 and the Java/Player controller runs on Fedora 10 using Player version 2.0.3.

The Koala is augmented with a Hokuyo URG laser range finder, which it uses instead of the onboard IR proximity sensors. Both the Java RTS and C++/OROCOS controllers handle the interface to the robot and its devices via serial connection. The Java RTS controller uses *javax.comm* (the Java Communication API) for serial communication. Player is used for sensor and robot communication in the Java/Player controller. All code, including the logic and control code is executed onboard the robot. Because of the high power requirements of the Acces I/O ETX-Nano, the robot was tethered with a power cord and an ethernet cat5 cable for remote communication. The testing environment was a 8m x 6m room (Figure 2).

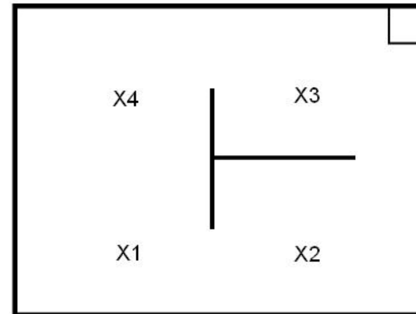

Fig. 2. The experiment room (8m x 6m) with four waypoints. The waypoints and waypoint order were chosen such that all behavior tasks would be employed.

The experiments were run on the three controllers: Java RTS using Real-Time Threads with the RTGC [5]; OROCOS using its Real-Time Threads; and Java/Player using a user written scheduler [6]. All trials were given the same four waypoints to reach (shown in Figure 2). From the start/end position X1, the robot traveled to each waypoint in a counterclockwise order until it arrived at the finish position. The waypoints were chosen such that all tasks needed to be employed to reach the ending position.

Five experiment sets were run. Each set consisted of five runs. The tasks were assigned a value that defined both its period and deadline for all sets. The robot was run with the same values as [5]. The other tasks period

TABLE I

TASK THREADS USED IN THE ROBOT ARCHITECTURE.

Tasks	Description	Provides	Requires
DATAPOSUPDATE	Sends movement commands to the robot and receives updated position information.	PositionData	
DATALASERUPDATE	Gets the laser readings from the URG laser.	LaserData	
WAYPOINTPLANNER	A high-level behavior that moves the robot through a series of pre-defined waypoints using a path created by the DIJKSTRAPATHPLANNER.		PositionData, PathData
LASERAVOIDOBSTACLES	Responds to obstacles sensed by the laser, via DATALASERUPDATE, by slowing down the robot and turning away from the obstacle, via DATAPOSUPDATE		LaserData
OCCUPANCYGRIDMAP [18]	Uses the position and laser data reported by DATAPOSUPDATE and DATALASERUPDATE and maintains an occupancy grid map. The map tracks unknown, open and occupied space.	MapData	LaserData PositionData
DIJKSTRAPLANNER [19]	Takes goal position requests and maintains a path from the current position to the goal. The requester must remove the goal when it is no longer valid (has been reached).	PathData	MapData

values were based on the Hokuyo URG laser, which runs at 10Hz [21]. Three of the five sets assigned all of the tasks the same period. The remaining two sets varied the period of LASERAVOIDOBSTACLES to see its effect on the overall performance of the experiments. DATAPOSUPDATE and DATALASERUPDATE periods were set at 75ms and 100ms respectively for all five experiment sets.

It was hypothesized that these experiments would show if one controller performed better than the others. Each controller's performance was measured based on the following metrics: course time, distance to the final goal and task's missed deadline rate. Trials were considered complete if they circled the obstacles approaching and passing each waypoint. If the robot either got stuck on an obstacle or did not reach all waypoints within six minutes, the trial was considered incomplete. Missed deadlines, overall course time and distance from the last waypoint were recorded for each run. Each controller was run using the different behavior periods (shown in Table II) to examine different period sets including some that were over and under scheduled.

TABLE II

BEHAVIOR PERIODS FOR EACH EXPERIMENT SET.

	Behavior Periods (ms)			
Set	*Waypoint Planner (W)*	*LaserAvoid Obstacles (L)*	*Occupancy GridMap (Oc)*	*Dijkstra Planner (P)*
1	10	10	10	10
2	100	10	100	100
3	50	50	50	50
4	100	50	100	100
5	100	100	100	100

V. EXPERIMENTAL RESULTS

Figure 3 shows the average time and standard deviation that the robot took to finish the course for each experiment set for each controller. Figure 5a shows the percentage of total missed deadlines for the behavior tasks (this does not include DATAPOSUPDATE and DATALASERUPDATE) for each experiment set. Since the first set of experiments and the Java/Player controller experiments, had such a large percentage of missed deadlines as compared to the others, they are removed in Figure 5b to show more detail of the other four sets. The robots final distance from the last waypoint and corresponding standard deviation for each set is shown in Figure 4. In all three controllers, the WAYPOINTPLANNER considers a waypoint reached if it is within 0.5m of it. So the closer the robot is to 0.5m from the goal the more accurate it is.

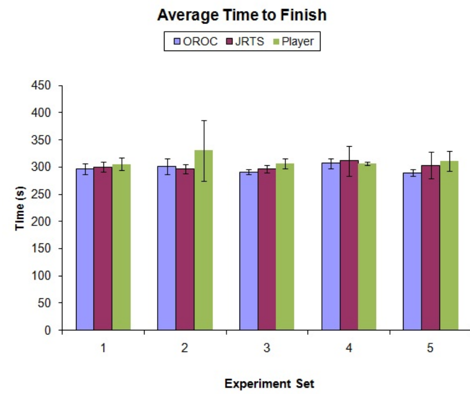

Fig. 3. The average time it took the robot to complete the course.

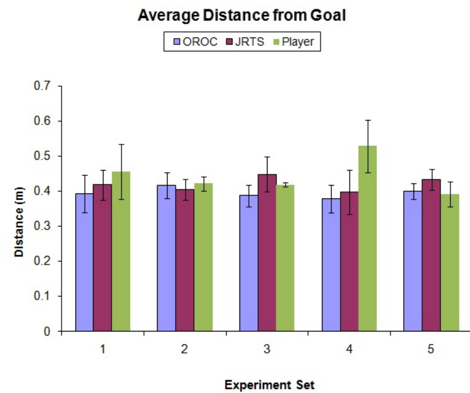

Fig. 4. The average distance the robot came near to the ending position.

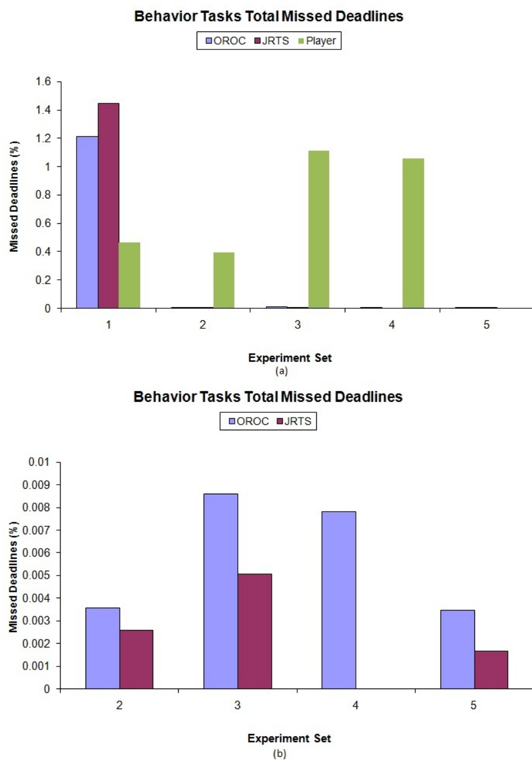

Fig. 5. Percentage of total missed deadlines (a) For all five experiment sets. (b) For experiment sets 2 - 5 (rescaled to show more detail).

VI. Analysis

It is also important to point out that the Java/Player architecture experiments had a large failure rate. While running the experiments Player would receive messages from "unknown devices" and stop running. This caused the trials to be rerun sometimes taking as many as 3 attempts to get one successful run.

Figure 5a shows that as the behavior tasks' periods increase, except in the Java/Player controller, the missed deadline rate decreases. Note that the difference between experiment sets 2 and 3 of the Java RTS architecture and experiment sets 3 and 4 of the OROCOS architecture have only one or two missed deadlines. So even though the percentage increases it is only being caused by an increase of one or two missed deadlines. Note that the computational requirements of the tasks are greatest with set 1 and decrease going from set 2 to set 5.

When looking at the average run times for each task, overall C++ is faster. However, this is not the case for DijkstraPlanner task (see Figure 6). OROCOS runs on the average around 2ms faster than the Java/Player controller. Note, that even though the Java/Player architectures' average runtime in set 1 is around 2ms, it has a standard deviation of 7ms. This makes it less predictable than the other controllers. Compared to Java RTS, OROCOS takes around 0.4 ms longer to complete the task.

The average time for the robot to complete the course and the distance from the last waypoint for all three architectures

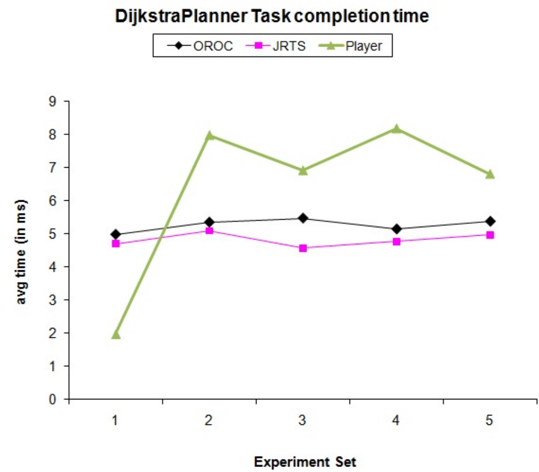

Fig. 6. Average time to complete DijkstraPlanner Task.

are almost identical (Figures 3 and 4). Therefore t-tests were used to determine if the result sets are statistically significant. The p values for the t-tests comparing the controllers' distances can be seen in Table III and the values for the time to finish can be seen in Table IV. When comparing the t-test results of OROCOS and Java RTS's time to finish values the only statistically significant data set is set 3. It is important to note the p value for set 3 of the time to finish is p=-.05, the threshold for statistical significance at 5%. The p value is 0.049, which indicates there is a 5% chance that the samples are from the same distribution. The other statistically significant sets for time to finish are set 4 comparing OROCOS and Java/Player and set 4 comparing Java/Player and Java RTS. The statistically significant sets for distance to goal are sets 3 and 5 comparing the OROCOS and Java/Player controllers. There is a control-based significance to set 3. Dorsey [2] states that the sampling rate has to be at least two times the fastest input signal. In our system, the Hokuyo laser sends out its data at a rate of 10Hz. So the periods for all of our behavior tasks need to be at least 50ms for the system to be stable.

TABLE III
T-test results comparing the three controllers' sets for distance to goal.

	T-test results for distance to goal		
Set	*OROCOS vs Java RTS*	*OROCOS vs Java/Player*	*Java RTS vs Java/Player*
1	0.618	0.243	0.434
2	0.615	0.287	0.224
3	0.135	0.010	0.117
4	0.723	0.890	0.672
5	0.264	0.042	0.622

When we look at the distance from the goal in set 3, the Java RTS architecture performs better than average. It is also worth noting that the standard deviation for OROCOS's time to finish is almost two times OROCOS's in set 3, three times OROCOS's in set 4 and almost five times OROCOS's in set

TABLE IV

T-TEST RESULTS COMPARING THE THREE CONTROLLERS' SETS FOR TIME TO FINISH.

	T-test results for time to finish		
Set	*OROCOS vs Java RTS*	*OROCOS vs Java/Player*	*Java RTS vs Java/Player*
1	0.421	0.391	0.767
2	0.618	0.935	0.686
3	0.049	0.113	0.377
4	0.603	0.011	0.037
5	0.077	0.474	0.455

5. OROCOS's and Java RTS's time to finish and the standard deviation of set 1 is almost identical. The Java/Player's time to finish is larger than the other two controllers in every set except set 4.

VII. CONCLUSION

We implemented a behavior-based real-time robot architecture using OROCOS. The system was created by converting [5]'s Java RTS based architecture into C++ and using OROCOS RTTs' PERIODICACTIVITY to be used for executing the tasks. The OROCOS system allows individual behaviors to be assigned separate execution frequencies and priorities. Unlike the Java RTS architecture, the OROCOS and Java/Player architectures run on non real-time operating systems. Experiments demonstrated how manipulating the timing of subtasks to the OROCOS and JAVA RTS architectures made their results (excluding set 3) not statistically significant. Whereas, the Java/Player controller sets were found not to be statistically significant when compared to Java RTS and OROCOSs overall time and when compared to Java RTS's distance from the goal. Also, since the Java/Player controller missed more deadlines, on an average took more time to finish, and required almost three attempts for each successful experiment one can conclude that the other two implementations are better choices. These results from testing support our hypothesis. Our hypothesis was that [5]'s Java RTS based controller will be comparable to the OROCOS RTT architecture and both would perform better than the Java/Player controller. Even though it was not the main focus of the research, we found evidence that a RTOS was not necessarily needed to implement a behavior-based real-time controller.

Future work will include user studies to determine if programmers prefer to use [5]'s Java RTS architecture over our C++/OROCOS RTT architecture. User studies that evaluate programmer proficiency on both architectures may shed light on features that are useful in teaching embedded programming techniques.

VIII. ACKNOWLEDGMENTS

The authors gratefully acknowledge the support of the following NSF grants: IIS-0846976, CCF-0829827 and CCF-0851824.

REFERENCES

[1] R. A. Brooks, "A robust layered control system for a mobile robot," *IEEE Journal of Robotics and Automation*, vol. RA-2, no. 1, pp. 14–62, March 1986.

[2] J. Dorsey, *Continuous and Discrete Control Systems*. New York: McGraw-Hill College, 2001.

[3] S. Smith, S. W. Lawson, and A. Lawson, "Can real-time software engineering be taught to java programmers?" in *Proc. of the 17th conference on software engineering education and training*, 2004.

[4] G. Bollella, T. Canham, V. Carson, V. Champlin, D. Dvorak, B. Giovannoni, M. Indictor, K. Meyer, A. Murray, and K. Reinholtz, "Programming with non-heap memory in the real-time specification for java," in *Proc. of the conference on object-oriented programming systems, languages, and applications*, http://portal.acm.org/citation.cfm?id=949344.949443, October 2003, pp. 361–369.

[5] A. McKenzie, S. Dawson, and M. Anderson, "Using sun's java real-time system to manage behavior-based mobile robot controllers," 2010, submitted (Jan 2010) to ACM Transactions in Embedded Computing Systems.

[6] A. McKenzie, S. Dawson, Q. Alexander, and M. Anderson, "Using real-time awareness to manage performance of java clients on mobile robots," *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pp. 3422 –3428, oct. 2009.

[7] H. Bruyninckx, "Open robot control software: the orocos project," in *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 3, 2001, pp. 2523–2528.

[8] G. Buttazzo, F. Conticelli, G. Lamastra, and G. Lipari, "Robot control in hard real-time environment," in *Proc. of the 4th international Workshop on Real-Time Computing Systems and Applications*, October 1997.

[9] R. Brega, N. Tomatis, and K. O. Arras, "The need for autonomy and real-time in mobile robotics: Case study of pygmalion and xo/2," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, October 2000, pp. 1422–1427.

[10] J. Auerbach, D. F. Bacon, D. T. Iercan, C. M. Kirsch, V. Rajan, H. Rock, and R. Trummer, "Java takes flight: time-portable real-time programming with exotasks," in *Proc. of the LCTES*, June 2007, pp. 51–62.

[11] B. Gerkey, R. Vaughan, K. Stoy, A. Howard, G. Sukhatme, and M. Mataric, "Most valuable player: a robot device server for distributed control," vol. 3, 2001, pp. 1226 –1231 vol.3.

[12] D. B. Stewart and P. Khosla, "The chimera methodology: Designing dynamically reconfigurable and reusable real-time software using port-based objects," *International Journal of Software Engineering and Knowledge Engineering*, vol. 6, no. 2, pp. 249–277, June 1996. [Online]. Available: http://citeseer.ist.psu.edu/18515.html

[13] H. Bruyninckx, P. Soetens, and B. Koninckx, "The real-time motion control core of the orocos project," in *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 2, Sept. 2003, pp. 2766–2771.

[14] S. G. Robertz, R. Henriksson, K. Nilsson, A. Blomdell, and I. Tarasov, "Using real-time java for industrial robot control," in *JTRES '07: Proceedings of the 5th international workshop on Java technologies for real-time and embedded systems*. New York, NY, USA: ACM, 2007, pp. 104–110.

[15] R. A. Brooks, "Intelligence without representation," ser. Artificial Intelligence, 1991, no. 47, pp. 139–159. [Online]. Available: http://citeseer.ist.psu.edu/brooks91intelligence.html

[16] K. Konolige, "Saphira robot control architecture," Technical report, SRI International, Menlo Park, CA, April 2002, Tech. Rep.

[17] M. Montemerlo, N. Roy, and S. Thrun, "Perspectives on standardization in mobile robot programming: The carnegie mellon navigation (carmen) toolkit," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, October 2003, pp. 2436–2441.

[18] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, no. 6, pp. 46–57, 1989.

[19] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[20] "Sun java real-time system precision control for the financial services market," *Sun Microsystems, Inc*, June 2008.

[21] *Range-Finder Type Laser Scanner URG-04LX Specifications*, HOKUYO AUTOMATIC CO., July 2005.