Dynamic Scheduling of a Fixed Bandwidth Communications Channel for Controlling Multiple Robots

Paul E. Rybski, Sascha A. Stoeter, Maria Gini, Dean F. Hougen, Nikolaos Papanikolopoulos
Center for Distributed Robotics, Department of Computer Science and Engineering
University of Minnesota Minneapolis, MN 55455
{rybski,stoeter,gini,hougen,npapas}@cs.umn.edu

ABSTRACT

We describe a distributed software system for controlling a group of miniature robots using a low capacity communication system. Space and power limitations on the robots drastically limit the capacity of the communication system and require sharing bandwidth and other resources among the robots. We have developed a scheduling and resource allocation system that is capable of dynamically assigning resources to each robot.

1. MINIATURE ROBOTIC SYSTEMS

We have developed a set of extremely small robots, called Scouts [2]. A Scout, shown in Figure 1, is a cylindrical robot 11 cm long and 4 cm in diameter, which is capable of rolling over smooth surfaces and of jumping over objects 20cm in height. Their limited size restricts their of on-board computational power, forcing them to rely on off-board computing. Scouts can send and receive digital commands over an RF communications link and can transmit video from their small video camera.

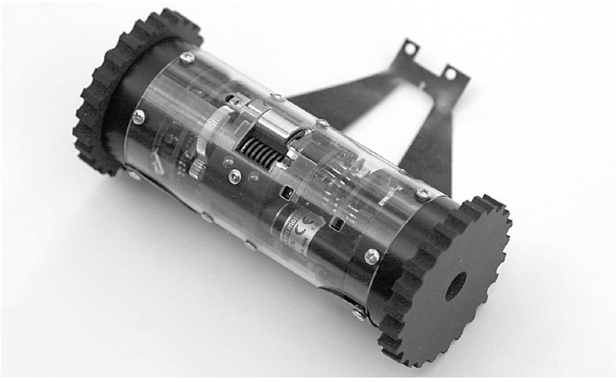

Figure 1: The Scout robot is 11 cm long and 4 cm in diameter.

AGENTS'01 May 28-June 1, 2001, Montréal, Quebec, Canada.

We have designed a distributed software control architecture which dynamically coordinates hardware resources and shares them between the clients, allowing for simultaneous control of multiple robots. All control behaviors and decision processes are organized in a hierarchical fashion, where "parent" nodes spawn off "children" to do various tasks. In this way, a task can be decomposed into specific subtasks, where each subtask is done by a specific behavior and its children. When a child behavior finishes its task, it returns status information back to its parent, which decides what actions to take from that point.

Behaviors are given priorities which are used to determine how to allocate access to resources. To access these resources, behaviors must contact components in the Resource Pool subsystem of the architecture. This subsystem controls access to robotic hardware and other computational resources through processes called RESOURCE CONTROLLERS (or RCs). Every physical resource has its own RC to manage it. If a behavior wants to use a particular resource, it must be granted access to the appropriate RC.

To control a single Scout robot, several physical resources are required. First, a robot must be selected which is not currently in use by another process. Second, a command radio is needed which has enough capacity to handle the demands of the process. If the robot is to transmit video, exclusive access to a fixed video frequency is mandatory, otherwise the interference between the video from multiple scouts would render it useless. Finally a framegrabber and tuned video receiver is required. Each of these resources is managed by its own RC.

Because control of a Scout requires having simultaneous access to groups of RCs, we use a second layer consisting of processes called AGGREGATE RESOURCE CONTROLLERS (or ARCs). Every ARC is an abstract representation of the group of RCs that it manages. This frees behaviors from the effort of managing all of the commands to each of the specific RCs. In fact, behaviors are only allowed to interface with ARCs and cannot contact the RCs directly.

Several kinds of ARCs are available, each requiring different resources to operate. These include ARCs which are capable of controlling only the actuators on the Scouts and do not allow the use of the camera, ARCs which drive the Scouts and broadcast video data, and ARCs which move the scout, broadcast data and capture it on a workstation for processing.

Within the Resource Pool subsystem, the Resource Controller Manager process oversees the distribution and access to the ARCs and RCs by running a centralized real-time schedule that dynamically adjusts itself to the current demand of all running decision processes. Behaviors send scheduling requests to the Resource Controller Manager in order to secure runtime for their ARCs. Each query is parameterized by the specific set of RCs that the ARC needs. The Resource Controller Manager takes each scheduling request and determines whether the behavior can access the ARC and RCs, or whether it has to wait until those resources are freed when a higher-priority behavior finishes.

Direct human control of the resources in the system is provided through the User Interface subsystem. Like a behavior, a User Interface component gets access to its ARCs and RCs by sending a scheduling request to the Resource Controller Manager. This allows a human to immediately take control over an autonomously controlled robot, adjust its position and actions, and then release it back to the behavior that was controlling it before.

2. SCHEDULING RESOURCES

Access to RCs must be scheduled when there aren't enough RCs to satisfy the requirements of the ARCs. The Resource Controller Manager maintains a master schedule of all active ARCs and grants each of them access to their RCs when it is their turn. When requesting access to a set of RCs, an ARC must specify a minimum amount of time that it must run to get any useful work done (generally on the order of seconds to minutes). The Resource Controller Manager uses a scheduling algorithm which tries to grant simultaneous access to as many ARCs as possible.

When activated, each ARC sends a request to the Resource Controller Manager that asks for access to its set of RCs. The Resource Controller Manager examines the ARC's request, checks that the RCs requested are running (and starts them if they are not), and then attempts to add the ARC to the schedule.

ARCs are divided into groups of equal priority. All ARCs of a higher priority must be either completed or running (not waiting on RCs) before an ARC of a lower priority can be added to the schedule. The Resource Controller Manager attempts to generate a schedule of running ARCs which allows all ARCs of the highest possible priority to run as often as they are able. If any ARCs of a lower priority can run at the same time as these higher priority ARCs without increasing the wait time of any of the higher-priority ARCs, they are allowed to do so. While this may not make full use of all available system resources, it does maintain the pre-defined priority assignments.

ARCs are divided into sets depending on the RCs they request. ARCs that ask for independent sets of RCs are put into different groups. These groups will run in parallel with since they do not interact in any way. The ARCs that have some RCs in common are examined to determine which ARCs can operate in parallel and which are mutually exclusive. ARCs which request an RC which is non-sharable cannot run at the same time and must break their operating time into slices. ARCs which have a sharable RC in common may be able to run simultaneously, assuming that the capacity requests for that sharable RC do not exceed the total possible capacity.

Once the ARC schedule has been constructed, the schedule manager takes care of notifying ARCs when their requested time slot has expired. When this occurs, the schedule manager instructs each of the running RCs to disconnect from their current ARCs and connect to the next set to be scheduled. The ARC signals its controlling process that it is no longer active and will not pass any messages to its RCs until they are available for work again.

3. SHARABLE RESOURCES

Sharable RCs, such as the Scout radio, have to manage their own schedule to ensure that each of their ARCs is given a chance to send packets to their robot at the rate they request. When requesting access to a sharable RC, an ARC must specify a usage parameter which defines how often it will make requests and, if relevant, what kinds of requests will be made. Commands sent to sharable RCs must be periodic and must have a constant interval between invocation. In addition, each request must complete before the next request for that command is made. However, because the CPU load of any given computer will vary depending on how many components are running on it, the run-time of any given request may vary. Given the first two constraints, and some assumptions on the third, a rate monotonic algorithm [1] is used to schedule access.

If all of the user-set priorities of the ARCs are equal, scheduling priority is given to ARCs with higher frequency requests (as per the rate monotonic algorithm). However, as with the Resource Controller Manager scheduler, ARCs with higher user-set priority have precedence over ARCs with lower user-set priority regardless of the frequency of the request. Once again, the relative user-set priorities of ARCs takes precedence over optimal usage of the resources. While this may seem counter-intuitive, it is an attempt to make the behavior of the overall system more deterministic.

4. EXPERIMENTS

We have done experimental studies on how the limited bandwidth affects the performance of the team while doing a surveillance task. Several scouts were placed in a hallway in order to detect the motion of people moving through it. Each robot had to share the same video frequency and so only one behavior could get data from its robot at any given time. As more robots were added to the system, the performance of the system dropped in a linear fashion. We are currently exploring methods of dynamic priority adjustment in the behaviors in an attempt to compensate for this.

Acknowledgements

Material based in part upon work supported by the Doctoral Dissertation Fellowship program at the University of Minnesota and upon work supported by DARPA/MTO, ARPA Order No. G155, Program Code No. 8H20, issued by DARPA/CMD under Contract #MDA972-98-C-0008.

5. REFERENCES

[1] C. Liu and J. Layland. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Journal of the Association for Computing Machinery*, 20(1):46–61, 1973.

[2] P. E. Rybski, S. A. Stoeter, M. D. Erickson, M. Gini, D. F. Hougen, and N. Papanikolopoulos. A team of robotic agents for surveillance. In *Proc. of the Int'l Conf. on Autonomous Agents*, pages 9–16, Barcelona, Spain, June 2000.