A Layered Architecture for Office Delivery Robots*

Reid Simmons
Richard Goodwin, Karen Zita Haigh, Sven Koenig, Joseph O'Sullivan
School of Computer Science, Carnegie Mellon University
Pittsburgh, PA 15213-3891
{reids, rich, khaigh, skoenig, josullvn}@cs.cmu.edu

Abstract

Office delivery robots have to perform many tasks. They have to determine the order in which to visit offices, plan paths to those offices, follow paths reliably, and avoid static and dynamic obstacles in the process. Reliability and efficiency are key issues in the design of such autonomous robot systems. They must deal reliably with noisy sensors and actuators and with incomplete knowledge of the environment. They must also act efficiently, in real time, to deal with dynamic situations. Our architecture is composed of four abstraction layers: obstacle avoidance, navigation, path planning, and task scheduling. The layers are independent, communicating processes that are always active, processing sensory data and status information to update their decisions and actions. A version of our robot architecture has been in nearly daily use in our building since December 1995. As of July 1996, the robot has traveled more than 75 kilometers in service of over 1800 navigation requests that were specified using our World Wide Web interface.

1 Introduction

A basic function of office delivery robots is to satisfy requests of the form "go to X, then go to Y." When the robots arrive at location X, an item, such as a document, is given to them, which is then removed when they visit location Y. To carry out such tasks, the robots must determine the order in which to visit offices, plan paths to those offices, follow paths reliably, and avoid static and dynamic obstacles while traveling. Several issues arise for such autonomous office delivery robots, the main ones being how to perform the tasks reliably and efficiently in the face of uncertainty and incomplete information. Moreover, the robots have to act in real-time to cope with a dynamic environment, such as moving people and changing delivery requests.

While many techniques exist for handling various parts of the delivery problem, comparatively little work has been done on building complete robot architectures. Only complete architectures, however, allow researchers to study interactions between the layers. Our architecture is based on layers of increasing abstraction. Upper layers in the architecture provide guidance to lower layers, while lower layers handle details that the upper layers have abstracted away. We show that not only does each individual layer provide for reliable and efficient behavior, but also that the overall architecture achieves synergistic effects by suitably partitioning system functionality.

A version of our robot architecture has been in almost daily use in our building since December 1995. As of July 1996, the robot has served over 1800 navigation requests, traveling a total of more than 75 kilometers. These experiments demonstrate that our robot architecture leads to fast and reliable navigation. The robot can travel at speeds of up to 60 centimeters per second in peopled environments. Its planning time is negligible and its task completion rate is now about 95 percent. The robot's travel speed is currently limited only by the cycle time of its sonar sensors, and tasks fail mainly due to problems with the wireless network at CMU – both problems are unrelated to the robot architecture.

Xavier, the robot used in these experiments [Nourbakhsh *et al.* 1993; O'Sullivan & Haigh 1994], is built on top of a 24 inch diameter RWI B24 base, which is a four-wheeled synchro-drive mechanism that allows for independent control of the translational and rotational velocities (Figure 1). The sensors on Xavier include bump panels, wheel encoders, a sonar ring with 24 ultrasonic sensors, a Nomadics front-pointing laser light striper with a 30 degree field of view, and a color camera on a pan-tilt head. Control, perception, and planning are carried out on three on-board 486 computers. The computers are connected to each other via thinwire Ethernet and to the outside world via a Wavelan wireless Ethernet system [Hills & Johnson 1996].

Section 2 presents an overview of our architecture and a scenario that illustrates how the various parts of the architecture work. Section 3 discusses each layer in de-

*Permission to make digital/hard copies of all or part of this material for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication and its date appear, and notice is given that copyright is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires specific permission and/or fee.
Agents '97 Marina del Rey CA USA
©1997 ACM

Figure 1: Xavier

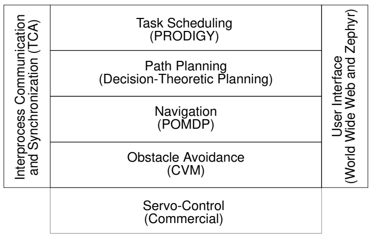


Figure 2: The Navigation Architecture

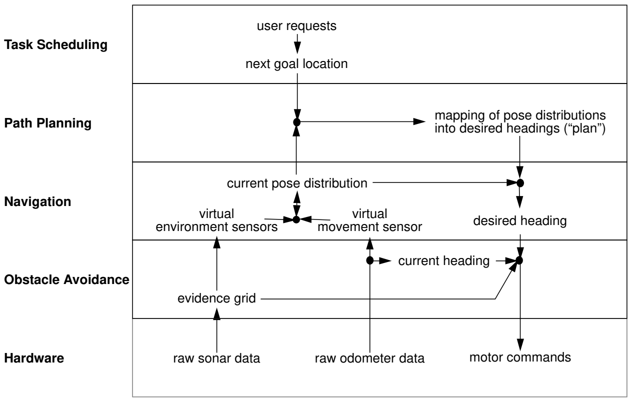


Figure 3: Flow of Information

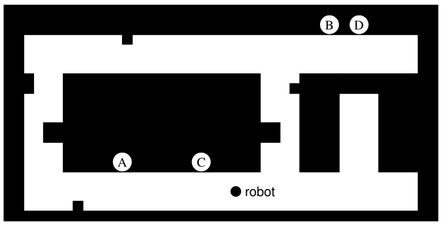


Figure 4: A Navigation Scenario

tail, including its functionality and its interface to the other layers. Section 4 presents the integration framework used and Section 5 discusses issues of user interaction with the robot. The paper concludes with performance data on the overall architecture and presents some of the lessons learned.

2 Overview of the Robot Architecture

Our robot architecture (Figure 2) consists of four layers: obstacle avoidance, navigation, path planning, and task scheduling. Each layer abstracts the raw sensor data, and higher layers work with the more abstract representations (Figure 3). The layers are implemented as separate code modules (processes). Other parts of the architecture include real-time servo control, which is provided with the commercially available hardware (robot base and pan-tilt head), an integration package that provides interprocess communication and synchronization between modules, and user interfaces for specifying requests and monitoring robot progress.

While this division of functionality is certainly not novel, each module offers novel approaches with solid theoretical foundations. Obstacle avoidance is performed by our Curvature-Velocity Method (CVM) [Simmons 1996]. Navigation is done using Partially Observable Markov Decision Process models (POMDPs) [Simmons & Koenig 1995]. Path Planning uses our decision-theoretic generate, evaluate and refine strategy that is based on ideas from sensitivity analysis [Koenig, Goodwin, & Simmons 1996]. Task scheduling is performed using our symbolic planning architecture [Haigh & Veloso 1996; 1997]. Interprocess communication and synchronization is provided by our Task Control Architecture (TCA) [Simmons 1994b]. The user interface uses both the World Wide Web and Zephyr. We illustrate the modules of our navigation architecture using a typical delivery scenario.

The **user interface** module allows users, such as secretaries, to enter delivery requests (including desired delivery times and priorities). It also provides a means for monitoring the progress of the robot, such as its current position and delivery request being carried out. Assume that there is one delivery request pending, a delivery from A to B (Figure 4), when another user enters a new delivery request: to carry a print-out from C to D.

The **task scheduling** module now has to determine the order in which to visit the four locations. Possible orders are ABCD, ACBD, ACDB, CABD, CADB, and CDAB. To make this decision, a symbolic planner (PRODIGY) consults with the path planner to determine the expected travel time between any two locations. The task scheduler integrates the new request into the current schedule, sequencing the order of the navigation tasks.

The **path planning** module determines how to travel efficiently from one location to another. Actuator and sensor uncertainty complicates path planning since the robot may not be able to follow a path accurately, and the shortest distance path is not necessarily the fastest. Consider, for example, the two paths from A to B shown

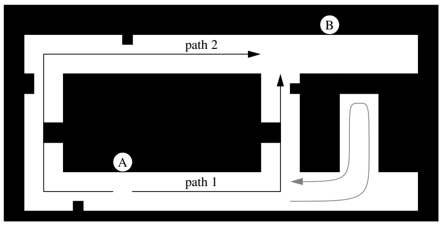


Figure 5: Two Paths from A to B

in Figure 5. Although Path 1 is shorter than Path 2, the robot could miss the first turn on Path 1 and have to backtrack. This problem cannot occur on the other path since the end of the corridor prevents the robot from missing the turn. The path planner uses a decision-theoretic approach to choose plans with high expected utility and uses sensitivity analysis to determine which alternatives to consider.

The **navigation** module generally follows the path suggested by the path planner. It may deviate from the nominal path since it, too, has to deal with sensor and actuator uncertainty. For example, if the path planner chooses Path 1 from A to B, and the robot indeed overshoots the turn, then it could mistake the dead-end corridor for the correct corridor. When it reaches the end of the dead-end and discovers its mistake, it issues corrective actions that turn the robot around and let it make progress toward the goal. Our navigation module uses a Partially Observable Markov Decision Process (POMDP) model to maintain a probability distribution of where the robot is at all times, choosing actions based on that distribution.

The **obstacle avoidance** module keeps the robot moving in the desired direction, while avoiding static and dynamic obstacles (such as tables, trash cans, and people). It provides high-speed, safe motion by taking the robot's dynamics into account and by optimizing, in real time, an objective function that combines safety, speed and progress along the desired heading.

The robot architecture is implemented as a collection of asynchronous processes. They are integrated using the **task-level control** package, which provides facilities for interprocess communication (message passing), coordination, and synchronization of the distributed, concurrent modules. Other modules use these facilities to indicate when subtasks should be active and how they should be monitored to determine whether they succeed or fail.

To a large extent, the architecture achieves reliability and efficiency by using reliable and efficient component modules. But reliability and efficiency are also achieved through the interaction of the layers:

- Higher layers can "guide" the lower layers into regions of the environment where safe and efficient navigation can take place. For instance, the path planning module takes the probability of execution failure into account when planning paths – keeping the robot away from areas where it may have difficulty navigating.
- Lower layers can take care of details abstracted out by higher layers. For example, while the navigation layer specifies headings for the robot to follow, the obstacle avoidance layer can steer the robot in a different direction, if needed to avoid local obstacles.
- Lower layers propagate failures up to higher layers when they find that they cannot handle certain exceptional situations. For example, the obstacle avoidance module can indicate when it thinks it is stuck in a local minimum. By "failing cognizantly" [Gat 1992], the lower layers can provide information that enables higher layers to determine how to handle the situation [Simmons 1994a].

In character, our architecture has many similarities to behavior-based approaches advocated in the literature [Brooks 1986; Connell 1989; Mataric 1992]. Lower layers are always running, even when higher layers are inactive, or not present. For instance, the obstacle avoidance module can keep the robot wandering safely, even without any "desired heading" input from the navigation module. In addition, lower layers are free (within some bounds) to ignore the input received by higher layers, essentially treating higher level plans and commands as "advice" [Agre & Chapman 1987]. For example, the local obstacle avoidance module can ignore the current goal heading to steer the robot around obstacles. Similarly, the navigation module has the option of choosing how to direct the robot towards the goal, if it finds that it has strayed from the path specified by the path planner (Figure 5).

The architecture differs from traditional behavior-based approaches in that it makes heavy use of models and internal representations. This actually has the effect of *improving* efficiency and reliability, since the representations explicitly model the capabilities and limitations of the robot, and take uncertainty and incomplete information into account. By combining prior information (models) with current percepts, the robot is able to maintain representations that best reflect its current belief in the state of the world, given that it receives noisy, and often incorrect, sensor information.

3 Component Modules

This section describes the modules in more detail. It also describes the interfaces between the modules and argues why the representations that the modules use are adequate for the tasks to be performed.

Obstacle Avoidance

Above all else, the robot must travel safely to protect itself and the people with which it shares the environment. In addition, we desire fast (walking speed) travel, both to make the robot useful as a delivery agent and to make it a more socially acceptable "inhabitant" of our building.

The input to the obstacle avoidance module is either a desired heading or a goal point. The obstacle avoidance module tries to either maintain this heading or head towards the goal point, while avoiding collisions. Without

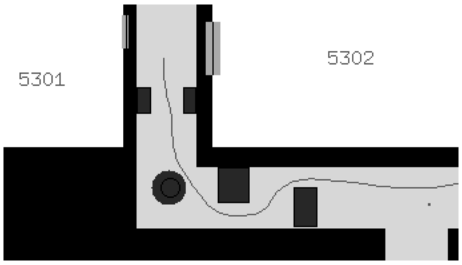


Figure 6: Local Obstacle Avoidance at 60 cm/sec

an explicit goal, the robot wanders while avoiding obstacles. If it gets stuck (trapped in local minima), the obstacle avoidance module will signal the other modules with a description of the problem encountered.

Previous obstacle avoidance schemes [Arkin 1989; Borenstein & Koren 1991] neglected dynamics by assuming the robot could turn instantaneously. While this assumption is reasonable for low-speed travel, for higher speeds one must take current velocities and feasible accelerations into account. Our curvature-velocity method (CVM) [Simmons 1996] poses the obstacle avoidance problem as one of constrained optimization in the velocity space of the robot. The *velocity space* is the space of all feasible translational and rotational velocities of the robot. Constraints are added to this space that limit the allowable velocities that can be commanded. For example, the minimum and maximum rotational velocities that can be achieved in the next time interval are limited by the current rotational velocity and the maximum rotational acceleration. Similarly, obstacles detected by the sonar and laser sensors add constraints on how far the robot can travel along an arc of a given curvature before hitting an obstacle. By *constrained optimization* we mean that the CVM method chooses velocity commands by maximizing an objective function, subject to the various velocity space constraints. The linear objective function contains terms for safety (distance to the nearest obstacle), speed, and progress (heading in the desired goal direction). By adjusting the coefficients of the objective function, we can easily enable the robot to trade off efficiency for reliability in its travels.

By making use of some reasonable approximations, the CVM method can efficiently compute the best rotational and translational velocities. The complete calculation, including sensor processing, takes 12 msecs on a 486 computer. In part, this is achieved by using the raw sensor range readings directly, doing a minimum of processing to suppress noise. The idea is to deal with sensor noise using a tight feedback loop – having the robot react quickly to any perceived obstacle and returning to the nominal (goal) heading as soon as possible. Figure 6 illustrates how the robot travels in our corridors (the goal heading is to the right – the robot travels down until it finds an opening, and then travels in the desired heading).

Navigation

The role of the navigation module is to direct the robot to a given goal location. Its input is a path plan, and it interfaces to the obstacle avoidance module by providing a series of goal headings. The navigation module works by estimating the current location of the robot, determining the direction the robot should be heading at that location to follow the path, and then sending the obstacle avoidance module any change in desired heading. It must work reliably in spite of noisy sensors and actuators ("dead-reckoning uncertainty") and incomplete knowledge of the environment (such as uncertainty about the exact lengths of corridors).

In order to avoid getting completely lost, our navigation module maintains a probability distribution over the current pose (position and orientation) of the robot. Given new sensor information and the current distribution over all possible poses, the navigation module uses Bayes' rule to update the pose distribution. The updated probabilities are based on probabilistic models of the actuators, sensors, and the environment:

actuator model: "If the pose of the robot is X and it believes that it has moved forward one meter, then its new pose is Y with probability P."

sensor model: "If the pose of the robot is X, then sensor Y will report feature Z with probability P."

distance model: "Corridor X is Y meters long with probability P."

door model: "Door X is open with probability P."

This information, together with a topological map, is automatically compiled into a Partially Observable Markov Decision Process (POMDP) model. The POMDP model produced by our system discretizes the pose of the robot: orientation is discretized into the four compass directions (relying on the rectilinear nature of most buildings) and location is discretized with a precision of one meter. There is a trade-off: a coarse discretization leads to smaller memory and runtime requirements, but at reduced precision.

Discretizing the pose allows us to abstract the raw sensor data. The raw, dead-reckoned data is discretized into virtual movement reports (e.g., "moved forward one meter" or "turned left ninety degrees"). The virtual movements abstract away low level control aspects, such as circumnavigating obstacles, by reporting the straight-line distance in the desired heading. Similarly, an evidence grid, which integrates raw sonar data over time [Moravec 1988], is used to derive virtual sensors that report on the environment. For example, we model three sensors (a front, left, and right sensor) that report features such as walls and openings of various sizes (small, medium, and large). These abstract virtual sensor reports are less noisy and more closely approximate the probabilistic independence assumptions needed by Bayes' rule.

The navigation module takes a path plan and converts it to a desired heading direction for each discrete robot location. Locations on the planned path are assigned headings along that path; locations off the nominal path are assigned headings to lead the robot back onto the path. Whenever the navigation module updates the probability distribution, it determines which heading is most likely to lead to the goal, and passes that information to the obstacle avoidance module. We currently choose the heading that has the highest total probability mass [Simmons & Koenig 1995], but choosing the heading associated with the most likely location is also a good strategy [Cassandra, Kaelbling, & Kurien 1996].

The navigation module is very reactive to unexpected sensor reports, since desired headings are maintained for *all* possible poses, not just the most likely pose. Thus, if the robot strays from the nominal path, it will automatically execute corrective actions once it realizes its mistake. Consequently, the navigation module can gracefully recover from sensor noise and misjudgments about landmarks. For example, assume that the robot takes Path 1 (Figure 5), but misses the first turn and turns into the dead-end instead. Initially, most of the probability mass is in the corridor, since the robot believes that it made the correct turn. However, when it reaches the end of the dead-end, the sensor readings will make the probability mass shift to the correct location. The desired heading within the dead end is South, so the robot turns, heads out of the dead end, and then turns right back onto the desired path. Note that, unlike landmark-based navigation schemes [Simmons 1994a], no separate mechanism is needed for error recovery – the robot is always (reactively) choosing the best direction to head in for reaching the goal.

Path Planning

The task of the path planner is to take a pair of locations and create a policy that can be used by the navigation module to guide the robot and by the task scheduler to estimate the travel time between the locations. By taking into account that some paths can be followed more easily than others, the planner can choose paths that steer the navigation module away from regions of the building where navigation is likely to fail.

The use of POMDP models for position estimation and action execution suggests using POMDP algorithms on the same models for goal-directed path planning. At present, however, it is infeasible to determine optimal POMDP solutions given our real-time constraints and the size of our state spaces (over 3000 states for one floor of our building) [Cassandra, Kaelbling, & Littman 1994; Lovejoy 1991]. Instead, our planner generates a nominal path to the goal location. Paths are generated using a modified A* search algorithm that creates a sequence of paths from shortest to longest. For efficiency, the planner operates on the topological map (augmented with metric information) rather than using the POMDP model directly. As each path is generated, it is evaluated using a forward projection to determine the expected travel time, assuming that all doors are open and no corridors are blocked. The projection takes into account that turns can be missed, such as the first turn of Path 1 in Figure 5. In cases where a path can be blocked by a closed door, a bound is calculated on the travel time that is needed to recover from a closed door. The planner then determines the range of expected travel times for the path by weighting the time needed to follow the nominal path by the probability that it is unblocked and adding the range of travel times needed to recover from each closed door, weighted by the a priori probabilities that the door is indeed closed.

The result is a set of partial plans (i.e. nominal paths augmented by incomplete contingency plans that specify how to recover from blocked doors and other obstructions) with their associated ranges of expected travel times. If none of these partial plans is clearly best, the planner selects a plan and refines it by planning a recovery route for one of the possibly closed doors, thus narrowing the range of possible expected travel times. The process of plan refinement continues until one plan is shown to be at least as good as all other plans. The planner uses methods from sensitivity analysis for meta level control to determine which plan to refine and which door recovery to plan for. This focuses its effort and allows it to identify the best plan in a fraction of a second, even for moderately complex environments.

Task Scheduling

Delivery requests may come from the users at any time. The task scheduler must consider how each task will affect the others in the queue, and then find an interleaving of the requests which maximizes the satisfaction of all users. The simple approach to handle tasks in a first-come, first-served manner leads to inefficiencies and lost opportunities for combined execution of compatible tasks [Goodwin & Simmons 1992]. In addition, the task scheduler must also know when actions fail and replan to achieve them since the robot operates in a dynamic world that is not completely known.

The task scheduler processes incoming navigation requests, prioritizes them, and identifies when different navigation requests can be achieved opportunistically. It is able to temporarily suspend lower priority tasks, resuming them when the opportunity arises, and to successfully interleave compatible requests [Haigh & Veloso 1996]. The task scheduler can also monitor the execution of the requests and compensate for failures [Haigh & Veloso 1997]. By abstracting away the details of how each request is achieved (e.g. which path the robot takes to a specified goal location), the task scheduler can more fully address issues arising from multiple interacting tasks, such as efficiency, resource contention, and reliability.

The task scheduling module is based on PRODIGY4.1 [Veloso *et al.* 1995]. PRODIGY is a domain-independent nonlinear problem solver that uses means-ends analysis and backward chaining to reason about multiple goals and multiple alternatives of achieving them. It has been extended to support real-world execution of its symbolic actions. The planning cycle involves several decision points, including which goal to select from the set of pending goals, and which applicable action to execute. Dynamic goal selection from the set of pending goals enables the planner to interleave plans, exploiting common subgoals and addressing issues of resource contention. Each time PRODIGY selects an action for execution, the task scheduler maps it into a sequence of actions supported by the lower layers, most commonly navigation, but also including vision and speech (which are not described in this paper).

When problems arise, the task scheduler can incorporate the new state information into its domain models and continue planning for the updated world. For example, each time a navigation goal is issued, the task scheduler monitors its outcome. Since the navigation module operates using probabilistic information, it may occasionally get confused and report a success even in a failure situation. Thus, the scheduler always verifies the location with a secondary test (vision or human interaction). If the scheduler detects that the robot is *not* at the correct goal location, it can either replan the task or re-analyze the set of pending tasks for newly created opportunities and plan to exploit them when possible.

By interleaving planning and execution, the scheduler can acquire additional domain knowledge to make more informed planning decisions. For example, it can prune alternative outcomes of a non-deterministic action, notice external events (e.g. doors opening or closing), monitor limited resources (e.g. battery level), and notice failures. As a result, the information lost by abstraction can be reconstructed when unexpected (infrequent) situations arise.

4 Task-Level Control

The robot architecture is implemented as a number of asynchronous processes, distributed over the three on-board computers (except for the user interface, which is off-board). The processes are integrated using the Task Control Architecture (TCA) [Simmons 1994b], a general-purpose framework for *task-level control*, by which we mean the coordination of planning, sensing and execution to achieve high level goals. TCA provides message passing facilities and facilities to support task decomposition, task sequencing, execution monitoring, and exception handling.

Other modules make use of TCA's facilities to indicate how to sequence subtasks. Modules pass messages to a central "control" module that indicate what subtasks should be executed at what times. Temporal constraints on task execution can be expressed either in absolute terms (e.g., "at noon") or relative terms (e.g., "after *subtask1* completes"). This information is maintained in a dynamically generated, hierarchical *task tree* structure, which encodes task/subtask relationships and the temporal constraints between subtasks. The task tree structure is augmented with execution monitors and exception handlers, to detect and correct anomalous situations. Modules can perform failure recovery by having the central control module modify the existing task tree – adding and deleting subgoals and temporal constraints.

Using TCA has several benefits in building autonomous systems. For one, the message passing protocols enable us to easily try out different implementations of the same functionality, as long as each implementation adheres to the same well-defined message interfaces. For example, we run the same navigation software on a second robot (named Amelia), which has a different hardware configuration, merely by running different low level software that interfaces with the hardware. All other interfaces remain the same, so the same code can run on both robots. Another benefit is that modules can be developed first in relative isolation, and then be integrated with the rest of the system by adding the appropriate task control information. Similarly, new tasks are easily coded up, by combining sequences of pre-existing behaviors, such as navigating to a given location, finding a door using vision, and detecting a person's face. While there is some overhead in maintaining the dynamic task tree structure, in our experience it is far outweighed by the flexibility provided in being able to easily create and execute task trees in a concurrent, distributed fashion.

5 User Interface

If the robot is to gain acceptance as a "delivery worker," it is important that it be easy to interact with. To increase the robot's impact on the community, interfaces were developed which allow both naïve and untrusted end users to specify tasks using communication mechanisms that are already present in the computing environment that the end users are familiar with.

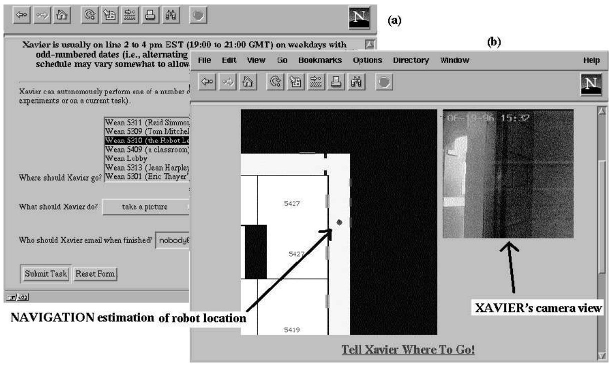
One such interface uses Zephyr, an asynchronous rapid communication mechanism with secure verification of sender [Dellafera *et al.* 1988]. With this interface, a remote user sends a textual task description to Xavier, such as:

```
% zwrite xavier -m Come to my office at 3pm.
```

The Zephyr-based interface receives the message and collects any additional information necessary to parse it (such as looking up the sender in the departmental data base to determine which room the phrase "my office" refers to). The module then coordinates with the task scheduler and finally informs the user of the status and estimated execution time for the task:

```
xavier:: [14:45:37]
      Job confirmed.
      Estimated arrival in about 15 minutes.
      (job: 4, building: weh, room: 5302)
```

Another interface, which uses the World Wide Web, was constructed to accept tasks from novice and widely distributed users. It allows arbitrary end users to specify one of 42 different non-invasive tasks, such as monitoring offices and delivering a restricted set of messages (Figure 7, left window). More importantly, the World Wide Web interface allows live monitoring of task execution. A map of the environment is annotated with the currently most likely pose of the robot. Xavier's view of the world, as seen by its on-board camera (Figure 7, right window) provides a sanity check



http://www.cs.cmu.edu/~Xavier

Figure 7: The World Wide Web Interface

for task achievement as well as a monitoring capability for users and researchers. The World Wide Web interface has been refined by comments from some of the 60,000 visitors since its initial announcement in December 1995. For example, the monitoring capability of task execution has been improved by specializing reports for different end user network bandwidths and by compressing the amount of data per report. End users are e-mailed acknowledgments of task completion, along with a mime-encoded picture that Xavier takes at its destination.

Xavier's World Wide Web existence differs from that of other "net robots" such as the USC Robotic Tele-Excavation, Chicago's Labcam, etc. First, Xavier's interface is to an autonomous agent, not simply a teleoperated robot. User requests are merely tasks to be performed; the actual decisions as to when and how to carry out these tasks are decided by the task scheduler and the other layers of the architecture. Second, Xavier operates in an uncontrolled, inhabited, bustling office building. It must not interfere with, be dissuaded by, or – worst of all – harm the people it encounters during task execution. Third, Xavier operates remotely, communicating over a wireless link. Thus it may not be accessible at all times and needs to be capable of operating without external intervention.

6 Results and Conclusions

The version of our robot architecture described in this paper has been in almost daily use since December 1995, mainly controlled by the World Wide Web interface described above. In the period from December 1, 1995 through July 1996 (Table 1), Xavier received nearly 8000 job requests. It attempted 1872 separate tasks (simultaneous requests to the same location were attempted together, and count as only one task), and reached its intended destination in 1758 cases (94%). Each job required Xavier to move 40 meters on average for a total travel distance of over 76 kilometers. The success rate is slowly climbing (from 90% to about 95%) as we find and correct bugs and refine the individual modules (December 1995 was an anomaly, since tasks in that first month were largely confined to a single corridor of the building). Many of the remaining failures are attributable to problems with our hardware (boards shaking loose) and the wireless communication – while the robot system itself runs on-board, the user interface (which includes the statistics-gathering software) operates off-board, connected by a wireless radio link.

More important than the raw data, our extensive experience with this architecture – both in development and in use – have taught us some valuable lessons about the construction of autonomous mobile robot systems:

It is important to have solid components

We have spent much of our research effort on making the individual layers as reliable as possible. Much of the effort has focused on understanding the capabilities and uncertainties inherent in real robot execution, and creating algorithms that work *with* the uncertainties, rather than assuming that the robot is omniscient and perfectly capable. While algorithms that explicitly deal with uncertainty tend to be more complex, they have great advantage in their ability to produce reliable behavior. (Although not described in this paper, our architecture also incorporates learning, both on a low level [Koenig & Simmons 1996; O'Sullivan, Mitchell, & Thrun 1996] and high level, to further increase component reliability.)

Layering increases reliability

Layered systems can be more reliable than the sum of their parts, since lower level components can deal with problems abstracted away at higher layers, while higher level components can try to keep the robot away from error-prone situations in the first place. For example, while the path planner produces routes that tend to keep the robot away from areas where it might miss turns, the navigation module can react correctly if a turn happens to be missed. Similarly, the obstacle avoidance module deals with features (such as trash cans and people) that are not modeled at the navigation layer or higher. This lesson is extremely important: Since it is (probably) impossible to produce individual modules that *always* act correctly in any situation, the modules need to do the best they can, realize when they are outside their capabilities, and signal their failure to other modules.

Design the architecture to be refined

The original Xavier system used a potential field approach for local obstacle avoidance. The CVM approach, which was developed following daily trials with our complete architecture, was easily added due to the layered approach. This methodology was common throughout the development of the architecture. Simple approaches were first implemented in a modular fashion, and each was supplanted only as experience dictated necessary, with the interfaces and data flow between layers remaining fixed.

End users want feedback...

The initial Zephyr-based interface was intended to be simple for naive end users. However, practically no

Month	Days in Use	Jobs Attempted	Jobs Completed	Completion Rate	Distance Traveled (approx.)
December 1995	13	262	250	95 %	7.65 km
January 1996	16	344	310	90 %	11.37 km
February 1996	15	245	229	93 %	11.55 km
March 1996	13	209	194	93 %	10.06 km
April 1996	18	319	304	95 %	14.11 km
May 1996	12	192	180	94 %	7.90 km
June 1996	7	179	170	95 %	8.24 km
July 1996	7	122	121	99 %	5.42 km
Total	101	1872	1758	94 %	76.30 km

Table 1: Performance Data for World Wide Web Tasks

one used it, since potential users were uncertain as to what types of tasks could be undertaken by Xavier. Also, users had little incentive to use the interface – without feedback as to what Xavier was doing, it was simpler for them to just perform the task themselves. The World Wide Web interface is more natural, providing a list of available tasks and real-time feedback, which makes people feel more in control, even though the robot remains actually autonomous.

...and people encountered want feedback
Xavier encounters people during the execution of its tasks. We found that treating people as mobile objects provokes unpleasant responses. Some people interfere with the task execution by blocking the robot, to excite a reaction, thus slowing down the task. Other people cower by walls, uncertain how to react. Current work involves examining the robust and fast detection of inhabitants in the environment, and interacting with people in a socially acceptable manner.

Acknowledgments

This research was supported in part by NASA under contract NAGW-1175 and by the Wright Laboratory and ARPA under grant number F33615-93-1-1330. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations and agencies.

References

[Agre & Chapman 1987] Agre, P., and Chapman, D. 1987. Pengi: An implementation of a theory of activity. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 268–272.

[Arkin 1989] Arkin, R. 1989. Motor schema-based mobile robot navigation. *International Journal of Robotics Research* 8(4):92–112.

[Borenstein & Koren 1991] Borenstein, J., and Koren, Y. 1991. The vector field histogram – fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation* 7(3):278–288.

[Brooks 1986] Brooks, R. 1986. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation* RA-2(1):14–23.

[Cassandra, Kaelbling, & Kurien 1996] Cassandra, A.; Kaelbling, L.; and Kurien, J. 1996. Acting under uncertainty: Discrete Bayesian models for mobile-robot navigation. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*.

[Cassandra, Kaelbling, & Littman 1994] Cassandra, A.; Kaelbling, L.; and Littman, M. 1994. Acting optimally in partially observable stochastic domains. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1023–1028.

[Connell 1989] Connell, J. 1989. A behavior-based arm controller. *IEEE Journal of Robotics and Automation* 5(6):784–791.

[Dellafera *et al.* 1988] Dellafera, A.; Eichin, M.; French, R.; Jedlinsky, D.; Kohl, J.; and Sommerfeld, W. 1988. The Zephyr notification service. In *Proceedings of the USENIX Winter Conference*, 213–219.

[Gat 1992] Gat, E. 1992. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 809–815.

[Goodwin & Simmons 1992] Goodwin, R., and Simmons, R. 1992. Rational handling of multiple goals for mobile robots. In *Proceedings of the International Conference on Artificial Intelligence Planning Systems (AIPS)*, 86–91.

[Haigh & Veloso 1996] Haigh, K., and Veloso, M. 1996. Interleaving planning and robot execution for asynchronous user requests. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*.

[Haigh & Veloso 1997] Haigh, K., and Veloso, M. 1997. High-level planning and low-level execution: Towards a complete robotic agent. In *Proceedings of the International Conference on Autonomous Agents (AA)*.

[Hills & Johnson 1996] Hills, A., and Johnson, D. 1996. A wireless data network infrastructure at Carnegie Mellon University. *IEEE Personal Communications* 3(1):56–63.

[Koenig & Simmons 1996] Koenig, S., and Simmons, R. 1996. Passive distance learning for robot navigation. In *Proceedings of the International Conference on Machine Learning (ICML)*, 266–274.

[Koenig, Goodwin, & Simmons 1996] Koenig, S.; Goodwin, R.; and Simmons, R. 1996. Robot navigation with Markov models: A framework for path planning and learning with limited computational resources. In Dorst, L.; van Lambalgen, M.; and Voorbraak, R., eds., *Reasoning with Uncertainty in Robotics*, volume 1093 of *Lecture Notes in Artificial Intelligence*. Springer. 322–337.

[Lovejoy 1991] Lovejoy, W. 1991. A survey of algorithmic methods for partially observed Markov decision processes. *Annals of Operations Research* 28(1):47–65.

[Mataric 1992] Mataric, M. 1992. Integration of representation into goal-driven behavior-based robots. *IEEE Transactions on Robotics and Automation* 8(3):304–312.

[Moravec 1988] Moravec, H. 1988. Sensor fusion in certainty grids for mobile robots. *AI Magazine* 9(2):61–74.

[Nourbakhsh *et al.* 1993] Nourbakhsh, I.; Morse, S.; Becker, C.; Balabanovic, M.; Gat, E.; Simmons, R.; Goodridge, S.; Potlapalli, H.; Hinkle, D.; Jung, K.; and Vactor, D. V. 1993. The winning robots from the 1993 robot competition. *AI Magazine* 14(4):51–62.

[O'Sullivan & Haigh 1994] O'Sullivan, J., and Haigh, K. 1994. *Xavier Manual (Version 0.2)*. School of Computer Science, Carnegie Mellon University, Pittsburgh (Pennsylvania). Unpublished Internal Manual.

[O'Sullivan, Mitchell, & Thrun 1996] O'Sullivan, J.; Mitchell, T.; and Thrun, S. 1996. Explanation based learning for mobile robot perception. In Ikeuchi, K., and Veloso, M., eds., *Symbolic Visual Learning*. Oxford University Press.

[Simmons & Koenig 1995] Simmons, R., and Koenig, S. 1995. Probabilistic robot navigation in partially observable environments. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1080–1087.

[Simmons 1994a] Simmons, R. 1994a. Becoming increasingly reliable. In *Proceedings of the International Conference on Artificial Intelligence Planning Systems (AIPS)*, 152–157.

[Simmons 1994b] Simmons, R. 1994b. Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation* 10(1):34–43.

[Simmons 1996] Simmons, R. 1996. The curvature-velocity method for local obstacle avoidance. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 3375–3382.

[Veloso *et al.* 1995] Veloso, M.; Carbonell, J.; Pérez, M.; Borrajo, D.; Fink, E.; and Blythe, J. 1995. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical Artificial Intelligence* 7(1):81–120.