Experiments in Behavior Composition

Jana Košecká† Henrik I Christensen‡ Ruzena Bajcsy†

† GRASP Laboratory
University of Pennsylvania
3401 Walnut Street
Philadelphia, PA 19104, USA
{janka,bajcsy}@grip.cis.upenn.edu

‡ Laboratory of Image Analysis
Aalborg University
Fr. Bajers Vej 7, Bldg. D1
DK-9220 Aalborg East, Denmark
hic@vision.auc.dk

Abstract

Widespread use of mobile robots can only be achieved when frameworks that enable specification, design and implementation of systems are available. These frameworks must provide a level of abstraction that enables use of the same methods for different tasks/missions to facilitate fast prototyping and design at low cost. In this paper a Task Description Language (TDL) for task specification is outlined. The tasks are specified as network of processes. The processes are described in terms of finite state machines (FSM) and their composition is achieved via set of composition operators, common to many process algebra models. From the obtained description of the task a Discrete Event Systems supervisory controller can be synthesized. To demonstrate that such an approach is a suitable basis for description of robot tasks a set of experiments with two different platforms situated in two different laboratories is described. The elementary processes necessary to carry out these experiments are presented. Obtained results are reported for three different experiments. The results demonstrate that the presented framework has the required characteristics.

1 Introduction

In a large number of application domains it is apparent that autonomous mobile agents might be of considerable utility; examples include intelligent delivery agents, assistance to disabld, exploration and map generation for environmental cleanup, etc. So far only a very limited number of systems has been put into real-world operation. Today the ability to handle rich real world environments is almost impossible, and there is consequently a need for engineering of the environment. It has been envisaged that use of purposive sensing to a certain degree might circumvent this problem. Another problem has been the lack of suitable frameworks for specification, design and control of autonomous mobile agents.

Empirical work has demonstrated that the traditional recovery based paradigm which has dominated perception, and vision in particular, is not robust or intractable for many real-world problems [2]. This has led to design of *behavior based systems*, where a set of perception-action modules is combined to provide the functionality needed for a given application. Examples of such systems have been reported by a number of researchers [3, 6, 16, 5]. The successful systems reported in the literature are, however, all characterized by use of a small number of behaviors, and a high degree of tailoring to the task at hand. It is consequently not obvious how this approach will scale to realistic industrial applications. In spite of the claims that the behavior based approach becomes intractable for large scale problems [15] and therefore unsuitable as a model of general purpose intelligent agents, by adopting this approach incremental progress has been achieved and several robustly working systems have been developed. One of the major criticisms against the behavior-based approach has been the lack of facilities for explicit system level control and planning.

In this paper a methodology for specification and analysis of behavior based systems is presented (Section 2), and it is outlined how this methodology may be used for expressing plans in the context of such systems. The methodology is based on the use of abstraction and explicit composition rules for combination of different elementary behaviors. It is further demonstrated how such an abstraction may be compiled/converted into a set of finite state machines, that enable use of traditional control methods in a run-time system. The expressiveness of the framework is shown on examples of tasks for a navigation/delivery agent (Section 3). To demonstrate the potential of the approach a set of experiments is outlined (Section 4). These experiments has been carried out on two different platforms in two different laboratories, to justify that the methodology has a sufficient level of abstraction to enable a description of 'general' behaviors rather than embodied ones. At the end of the paper the consequences of such a representation are discussed.

2 A Framework for Description of Behavior Based Systems

2.1 Basic System Components

Before defining a framework for description of the behavior of a complex system one has to settle clearly on what are the elementary components of such a framework. In the case of autonomous mobile agents we can clearly partition these into the following categories:

Action Component With each actuator of the mobile agent we associate a set of elementary motion strategies, which can be parameterized based on the task to be accomplished.

Perception Component With each sensor we associate a set of strategies for acquisition and processing of sensory data. In case of visual behaviors, these strategies correspond to various purposive visual routines needed for a variety of tasks.

Computation Component This category comprises computational procedures that does not directly rely on or influence the perception and action components. Most of the procedures in the category can be executed both offline and online. To this category belongs, for example, global path planning and human-computer interfaces.

For the modeling purposes each elementary strategy or computation is represented as a process[1] and has a finite state machine model (FSM) associated with it. The transitions between the states of the FSM model are modeled by events, capturing clearly initiation, termination, interruption or change of the global variables of the elementary strategy. The global variables (or more specifically predicates on them) play an important role in our framework, expressing the goals the robot should achieve, maintain or prevent from happening. The set of final states of elementary strategies is partitioned into a set of successful and unsuccessful states. Communication between two processes running in parallel is modeled via shared events. If the two processes share an event a communication link between them is established.

[1]The word *process* and *strategy* will be used interchangeably throughout out the article.

2.2 Composition of Elementary Processes

To achieve tasks it is necessary to combine the processes outlined above into complete systems. To ensure generality in the design of such systems, a programming type of framework is needed. A basic characteristic of a programming/scripting facility is the ability to combine basic operations into more 'complex' ones. In the domain of intelligent agents the basic operations comprise elementary strategies and their composition is achieved using *composition operators*.

The operators (common to almost any process model) capture the temporal and structural dependencies between the processes. As we mentioned earlier, since the types of behaviors which need to be invoked depend on the task to be accomplished, we adopt the notion of task representation as a network of processes. The idea of representing tasks as networks of processes has originally been proposed by [10]. Lyons proposed the $\mathcal{RS}$ (Robot Schema) model, where both plans and the world were modeled as networks of processes. The semantics of the composition operators was modeled in terms of port automata. We propose, from an $\mathcal{RS}$-like specification of the task, how one can synthesize a finite state machine supervisor which serves as a discrete event controller for the task. The brief description of the composition operators, their semantics in terms of finite state machines and some examples follow. For more details see [8, 9].

Sequential composition $P = R\ ;\ S$. Process P behaves like R until R terminates and then behaves like S. P terminates when S terminates and has the same termination status as S.

Parallel composition $P = R \parallel S$. Process P behaves like R and S running in parallel. P terminates with the termination and status of the last terminated process[2].

Conditional composition $P = R < v > \ : S(v)$. Process P behaves like R until R terminates successfully computing v which is then used to initialize process S[3]. If R fails the composition fails.

[2]The composition of parallel processes requires synchronization of the processes in terms of shared events. The exact formulation of this is described in [9].

[3]The variable v is a global variable. In our case it is always a global variable, that may be accessed by multiple processes. Global variables are not explicitly modeled in

Disabling composition $P = R \sharp S$. Disabling composition is similar to parallel but if one of the processes terminates the other process is terminated as well. P has the same termination status as the process that caused the termination of the composition (i.e., the process that finished first).

Synchronous recurrent composition $P = R < v > :; S(v)$ is recursively defined as $R < v > :; S(v) = R < v > : (S(v) ; (R < v > :; S(v)))$. This composition terminates with the failure of process R.

Asynchronous recurrent composition $P = R < v > :: S(v)$ is recursively defined as $R < v > :; S(v) = R < v > : (S(v) \parallel (R < v > :: S(v)))$. This composition terminates with the failure of process R.

An elementary process is defined as a generator $\mathcal{G} = (Q, \Sigma, \delta, q_0, F)$, where:

Q is the set of states;

Σ is the set of events such that $\epsilon \in \Sigma$;

q_0 is the initial state;

$F \subseteq Q$ is a set of "marked" states, such that $F = F_s \cup F_u$ where $F_s \cap F_u = \emptyset$, $F_u \neq \emptyset$, F_s is a set of successful final states, and F_u is a set of unsuccessful final states;

$\delta \subseteq (Q \times \Sigma \times Q)$ is the transition relation such that $\forall (q, e, q') \in \delta.\, q \notin F$.

Elementary strategies are modeled in terms of FSM's and their composition operators are defined in the following manner [4].

our framework at this moment; but can be passed as parameters or returned as values by individual processes.

[4]Throughout the paper the strategies are identified by name in capital italics (e.g. *GoTo*). In the finite state machine description lower case italics are used to denote events. Lower case italics are also used to denote variables representing both the values returned by the strategy upon completion and the values passed to the strategy as parameters.

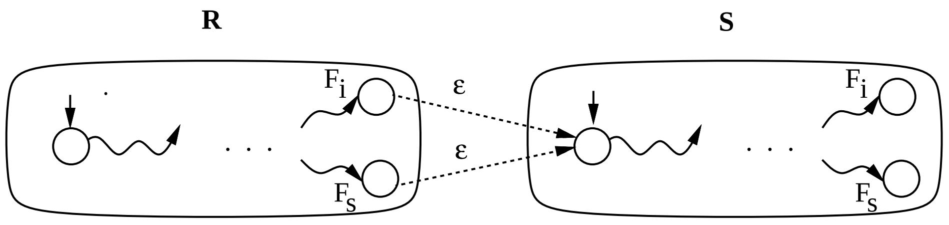

Figure 1: Sequential composition of two processes R and S.

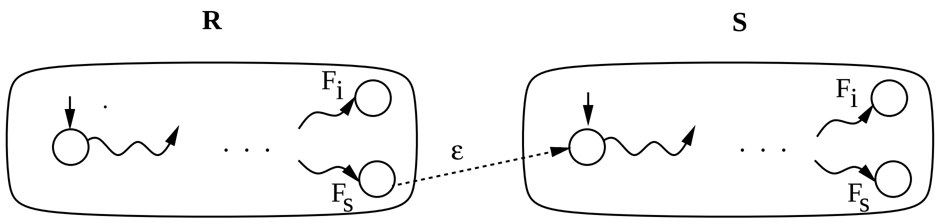

Figure 2: Conditional composition of two processes R and S.

Sequential composition: $P = R \; ; \; S$. Sequential composition of processes R and S, is achieved by connecting the final states of process R with the initial state of the process S through an ϵ-transition. The sequential composition of the two processes is depicted in Figure 1.

An example of the sequential composition could be:

$$GoTo_A(goal_1) \; ; \; GoTo_A(goal_2)$$

where $goal_i$ is (x, y) position in a global reference frame. The $GoTo_i(goal_j)$ is an elementary control strategy of the mobile base which specifies for the agent i to reach $goal_j$. In the composition above even if the first process terminates unsuccessfully the second process is still initiated and $goal_2$ can be reached.

Conditional composition: $P = R < v >: S(v)$. The conditional composition is formed by joining the successful final states of process R and the initial state of the process S via ϵ-transitions. A graphical representation of the conditional composition of two processes is shown in Figure 2.

The notion of conditional composition resembles the notion of precondition as often used in the traditional AI planning. One possible way of interpreting the conditional composition is that the first argument is a process which monitors certain condition to be true. This condition can be viewed as a precondition of the second process. Once the condition is achieved the first process terminates and the second process is initiated. An example of such a composition could be:

$$Locate < landmark > : Servo(landmark)$$

A robot, upon recognizing the landmark, heads towards it. In this case $Servo(landmark)$ is with respect to the environment a closed-loop strategy, since the landmark is beeing monitored throughout the process.

Parallel composition: $P = R \parallel S$. Parallel composition is formed as a *synchronous product*[5] of participating FSM's. An example specifying that the two mobile agents A and B should explore the environment in parallel can be expressed in the following way:

$$Explore_A \parallel Explore_B$$

Another example demonstrating the concept of parallel composition of two elementary strategies is $GoTo \parallel Detect$, corresponds to the task of reaching the desired goal while avoiding obstacles and is shown in Figure 3.

Disabling composition: $P = R \sharp S$. The disabling composition is modeled as a synchronous product of the participating FSM's. Prior to forming this synchronous product, all terminal events are relabeled to a common label to ensure synchronization and preemption of other running process. This common event then becomes the shared event between the processes and causes all participating processes to terminate when one of them terminates. An example of two mobile agents following one another can be expressed as:

$$GoTo_A(goal_1) \sharp Follow_B(agent_A)$$

If any on the agents terminates the other agent is terminated. Disabling composition of these two elementary strategies is depicted in Figure 4.

[5]Synchronous product of two finite state machines is a more general version of Cartesian product. If two FSM's share an event in their alphabet, they are synchronized on that event (i.e. the event is asserted only if it can be asserted by both processes).

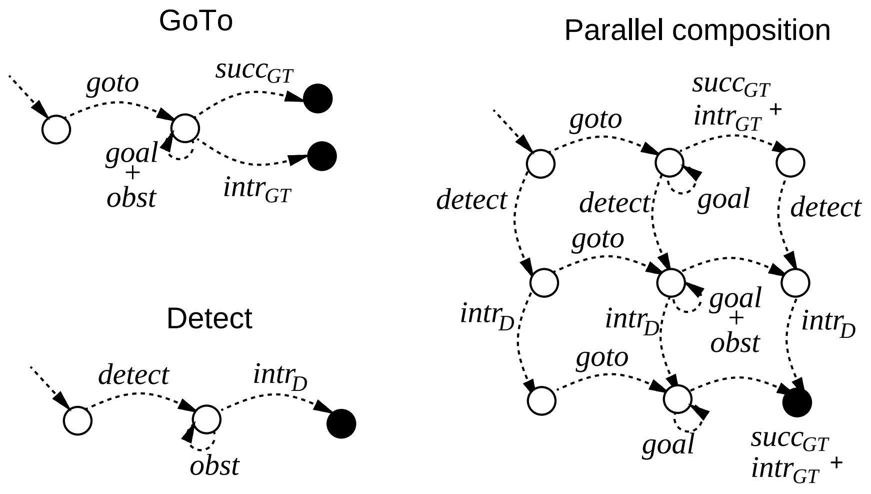

Figure 3: Parallel composition of two processes *GoTo* and *Detect* is formed as a *synchronous product* of their finite state machines. In this case processes share an event *obst*. The succesfull and unssuccessfull states of the composite finite state machines are joined together to a single final state.

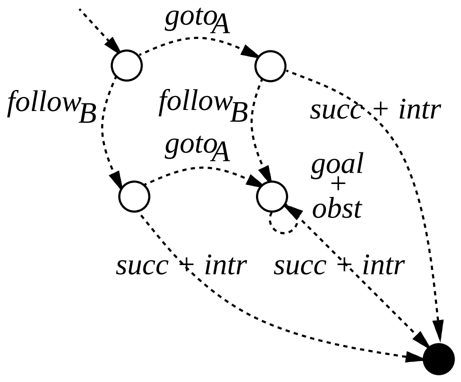

Figure 4: Disabling composition of two processes $GoTo_A$ and $Follow_B$ is formed as a synchronous product of their finite state machines after a relabeling of the events which go to the final state. In this case the processes share an events *intr* and *succ*.

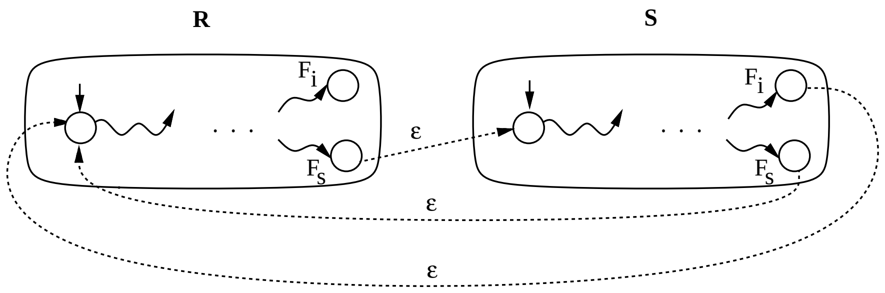

Figure 5: Synchronous recurrent composition.

Synchronous Recurrent Composition: $P = R < v >:; S(v)$. This can be recursively written as $R < v >:; S(v) = R < v >: (S(v) \; ; (R < v >: ; S(v)))$. Once process R successfully terminates process S is initiated. Upon completion of S, process R is again initiated and so on. The synchronous recurrent composition of two processes is outlined in Figure 5. To model the behavior of looking for targets and tracking them upon detection can be expressed in the following way:

$$LookFor < target > \; :; \; Track(target)$$

When the process $LookFor$ detects an ahead specified target, the process $Track$ then locks on it and tries to keep the target in the field of view. If the target is lost the target searching process is reinvoked.

Asynchronous Recurrent Composition: $P = R < v >:: S(v)$. This can be recursively written as
$R < v >:: S(v) \; = \; R < v >: (S(v) \parallel (R < v >:: S(v)))$. Once process R successfully terminates a recursive copy of process $(S(v) \parallel (R < v > \; :: S(v)))$ is created. This means that R is asynchronously initiating instances of S. For more examples see [8]. The fact that this composition operator creates multiple copies of process S brings some subtleties to the pictorial representation of the finite state machine composition.

2.3 Composition of Tasks

In the examples in the previous paragraph the arguments of the composition operators were both the elementary and composite processes. Since more complicated tasks require invocation of more than one elementary strategy, the model of the desired behavior in terms of a finite state machine is obtained by traversing a parse tree obtained from the expression in the task specification language.

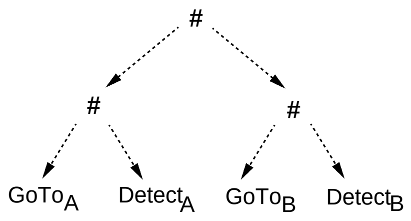

Figure 6: Task tree for the task of two mobile agents marching cooperatively while avoiding obstacles. The nodes of the tree are labeled by composition operators and leaves by elementary strategies.

The nodes of the parse tree are the composition operators and the leaves are the elementary strategies. An example of a parse tree [1] for the task of two mobile agents marching together cooperatively to a given destination *goal* is shown in Figure 6[6]. The task specification for this task is:

$$(GoTo_A(goal) \sharp\ Detect_A) \sharp\ (GoTo_B(goal) \sharp\ Detect_B).$$

From the finite state machine description of the task, we can directly obtain a finite state supervisory controller as defined in the Supervisory Control Theory of Discrete Event Systems (see [14] for a concise introduction to the subject). For a more precise description of the composition operators and the synthesis of the supervisory controller see [8].

3 Elementary Behaviors

To demonstrate the utility of the proposed methodology for robotics tasks, the domain of an "intelligent delivery agent" has been chosen. The choice is mainly motivated by a large variety of tasks that may be modeled as a delivery task. The scenario used in experiments is an indoors laboratory environment where the robot is required to perform both navigation within a room, with obstacle avoidance, as well as movement between rooms.

[6]The task specification language can be represented by so called operator grammar for which the LR parser can be effectively constructed.

3.1 Macro Behaviors

For this domain a set of "macro" level behaviors is used. A more detailed description of these behaviors is outlined below.

Bumper The *Bumper* behavior is a low level process which is responsible for stopping the robot if it comes too close to an object in the environment. The bumper is implemented either as a physical bumper connected to an electronic switch or as a low-level sonar loop that detects objects close to the platform. Once activated, the Bumper behavior continuously monitors either the switch or the sonar readings. Once the 'collision' is detected it generates a 'collision' event and terminates.

GoTo($goal, obstacles$) The $GoTo$ behavior is a process/strategy that brings the platform to a pre-specified configuration. The movement from the present configuration to the goal is currently based on the feedback control law derived from an artificial potential function, but other methods might also be used. The strategy is used for navigation within a single room. Once the strategy is invoked the control law repeatedly computes the commands to the mobile base actuators. Once the goal configuration is reached the strategy terminates successfully. The process takes as parameter information about obstacles in the vicinity of the robot, so if it is invoked in parallel with the obstacle detection process, it successfully avoids obstacles. The information about obstacles can also be provided from the outside by any other process if necessary (e.g., a global model of the world). The strategy terminates unsuccessfully, if the final goal is in an obstacle region or if the platform fails mechanically.

Detect($obst$) The *Detect* behavior is responsible for detection of obstacles on the path the robot is currently traversing. This process produces an 'obstacle' event if an obstacle is detected. The behavior is either based on vision or sonar information.

Servo($landmark$) This behavior does servoing on a landmark, to bring the platform to a specified configuration with respect to the landmark. Landmarks are defined to have a $2D$ or $3D$ structure. In the present system only information about 'doorways' (a $2D$ structure) has provided. For this landmark the robot is required to position itself centered and perpendicular to the landmark, before it tries to approach the landmark. Initially the robot is controlled to be positioned centered on the landmark, and then the motion strategy is to drive directly towards the landmark. The process will generate a 'door-completed' event when servoing terminates. The event description includes a specification of the robots present position.

Plan($goal_0, ..., goal_n$) Path planning, being a part of the computation component of the system, is used for introduction of deliberation into the behavior based architecture. The process is activated when a long-term plan (in our case we will restrict ourselves to path planning) of intermediate goals is required. The process (procedure) terminates with a 'path-available' or 'not-accessible' event depending of the availability of a path to the goal location and returns the path to be followed by the mobile base.

Localize(x, y, θ) This behavior is a periodic process that is activated for self-localization of the platform with respect to the environment. The process generates a 'robot-re-initialized' or 'unable-to-localize' event upon completion, depending on the ability to localize the robot with respect to the landmark in the environment.

Init This behavior is used for startup and self-calibration of the system. The process generates a 'robot-active' or 'robot-failure' event depending on the success of the initialization process.

3.2 Handling of errors and uncertainty

It is well known that the sensory system does not provide reliable readings, which causes problems in terms of interpretation and control. The set of behaviors outlined in section 3.1 is at a level of abstraction where the majority of uncertainty handling is internal to each of the behaviors. For example for the $Servo(door)$ strategy the servoing is based on a PID controller where the sensor noise is modeled explicitly as part of the controller design. Another type of robustness comes from the fact that for most of the tasks the elementary motion strategies are invoked in parallel with sensory strategies closing the loop with the environment. That way the unexpected deviations due to the unforeseen events, can be accommodated by the low-level controllers while executing the task. For example the *Detect* behavior encounters an obstacle in the robot's path, it will update the *obstacles* parameters of the *GoTo* behavior using using the *obst* event. The *Detect* behavior can be robustly tuned up (calibrated) for a variety of floors and lighting conditions. There is consequently no need for explicit modeling of random noise at this level of system description. In addition to that each elementary strategy has a set of predicates associated with unsuccessful termination, therefore certain types of errors due to uncertainties or unexpected dynamic interactions can be detected and particular error recovery routines invoked. An example of this is the tracking task, where when the target is lost the *LookFor* behavior is reinitiated.

4 Experiments

We verified the suitability of the framework for modeling the operation of a platform in delivery tasks. It is important to note that the framework itself has the full power of a programming language that is able to express arbitrary logical and temporal dependencies between individual processes. The types of tasks which can be reliably executed then depend on the types of available elementary motor and perceptual strategies. While the motor strategies are determined by the physical characteristics of the agent (i.e. degrees of freedom), the perceptual strategies are for now chosen purposively depending on the task at hand.

A set of experiments has been designed, which exploit the elementary behaviors outlined in the previous section. Two different scenarios are used for the experiments. The first scenario includes navigation within a single room. The robot is supposed to go from the initial location to a desired *goal* location, while avoiding obstacles. Intermediate points along the path are not specified.

4.1 Description of system controllers/behaviors

The functionality of the robots may be described by the composition

$$Agent := Init \; ; \; (Bumper \sharp (Plan < goal > \; ; \; (GoTo(goal) \sharp Detect)))$$

Similarly, for the navigation task which includes several rooms a specification as shown below may be used

$$\begin{aligned} Agent &:= Init \; ; \; Bumper \sharp \, (Plan < goal_i, door_i > \\ &:; \; ((GoTo(goal_i) \; ; \; Servo(door_i) \sharp \, Detect)) \end{aligned}$$

4.2 Outline of bases

Experiments involving two different platforms in two different laboratories (GRASP Laboratory, University of Pennsylvania and Laboratory of Image Analysis, Aalborg University) have been conducted.

The first platform is a TRC Labmate with a half circular array of ultrasonic sensors, stereo pair of cameras tilted with respect to the horizontal plane, and structured light sensor. The robot is controlled by a PC-based vehicle controller that is connected to a on-board SPARC station where most of the processing of sensory data takes place. The detailed description of the experimental platform can be found in [8].

The other platform is a RoboSoft Robuter 20, equipped with an array of 24 ultrasonic sensors and a binocular camera head. The robot has three on-board Motorola computers (two running the real-time operating system OS/9 and one running UNIX V). The robot is described in detail in [12].

The behaviors outlined in section 3.1 have been implemented on both platforms. Presently the *Servo*(*door*) behavior has only been implemented on the Robuter. Thus, experiments involving room to room navigation has only been tested on that platform.

4.3 The Experiments

To evaluate the performance of the robot systems a set of experiments has been defined:

1. Go from A to B in the same room (without meeting obstacles)
2. Go from A to B in the same room (while avoiding obstacles)
3. Go from A to B where start and goal points are in different rooms.

The first experiment is primarily carried out to verify that basic navigational capabilities can be performed.

In the second set of experiments the obstacles are introduced into the workspace obstructing the path towards the specified goal. This experiment includes the case where the goal position is occupied by another object, so it is impossible for the robot to succeed.

In the third set of experiments the robot is required to perform the same actions as in second set of experiments, but in addition the robot is required to plan a path with intermediate goals to traverse the passage between different rooms (which requires activation of the door servoing behavior).

Due to space limitations it is not possible to report the results from all of the above experiments in any detail. The results from a few test examples will, however, be presented.

In Figure 7 the basic layout of the environment for the TRC Labmate experiments is shown. The robot has no *a priori* information about the environment, so all the object elevated above the ground plane are considered to be obstacles. The desired goal is specified in the area on the other side of the laboratory behind the desks.

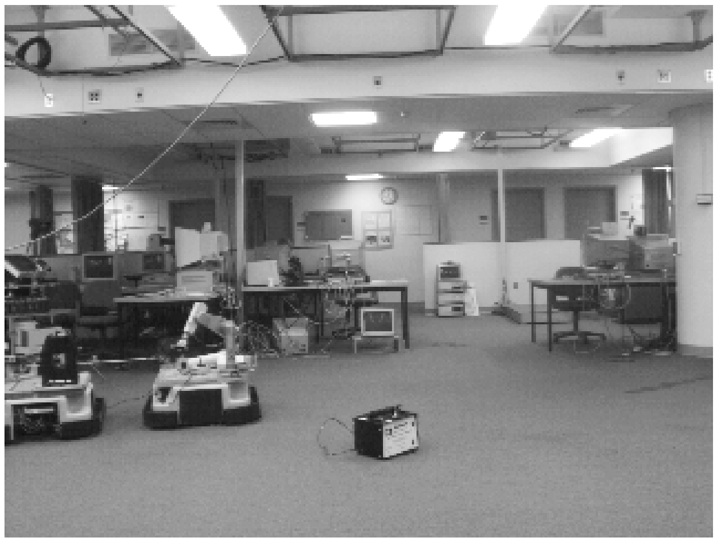

Figure 7: The laboratory layout where the experiments were carried out, as viewed from the approximate starting position of the robot.

The TRC Labmate was used for the experiment series 1 and 2. In Figure 9 the results from the series 2 experiment are shown. It is evident how the *GoTo* and the *Detect* behaviors interact with each other to arrive at the goal position. The obstacles detected by *Detect* are superimposed on the original layout of the environment, where the center of each grid cell in the common field of view of the stereo pair, mapped to the ground plane is marked by a cross. In this particular setup it appears as though the robot is running into objects/obstacles. In reality the robot successfully avoids these objects, but due to the discrepancy between the real position of the robot and the expected one, the recorded trajectories and obstacles are not superimposed correctly to the global coordinate system. This error is due to the fact that the present setup does not contain facilities for self localization of the platform. Through out the experiment the control system relies entirely on the odometric readings which given slippage is not a very reliable measure.

The result from another experiment is shown in Figure 10. Again the interaction between the different behaviors is obvious. In both cases it is worth noting the narrow field of view of the obstacle detection process. Finally the TRC platform was given an exploration command (i.e. the *GoTo* strategy was invoked without any specific goal directing the platform to move in the direction of the current heading) to explore the free space. The results are shown in Figure 11.

Similar experiments have been carried out with the Robuter platform. Some of the results have already been presented, see [13] for details. To demonstrate the full set of behaviors the third series of experiments were carried out with this platform.

To give an impression of the complexity of the environment the layout of the laboratory is shown in Figure 12.

The result from one of the experiments is shown in Figure 13. In this particular case the robot is required to move from the back of room D1-105 to the hallway. This implies that it has to go to the other side of the room, and then traverse through the door to arrive in the hallway. It is here seen how the *GoTo* and the *Detect* behaviors interact to enable avoidance of obstacles. Once the doorway is expected to be in sight the *GoTo* behavior is exchanged with the $Servo(door)$ serving behavior to enable detection and negotiation of the doorway. It appears as though the robot is running very close to the frame of the doorway, this is due to use of odometric information for plotting of the path. In reality the robot passes through the middle of

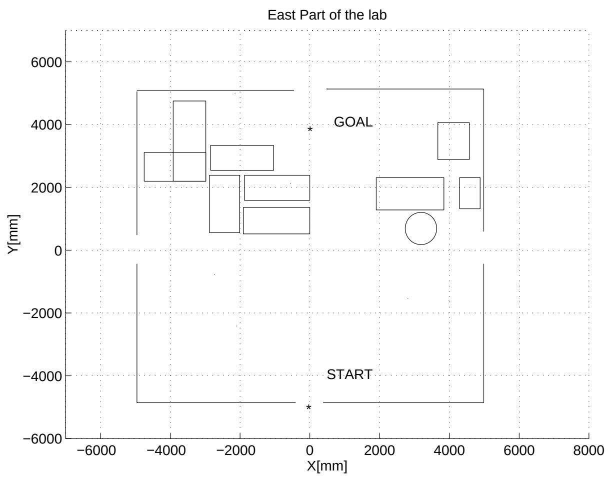

Figure 8: The graphical layout of the environment from Figure 7. The beginning of the global coordinate frame is in the middle of the room and the robot has no apriori information about the obstacles.

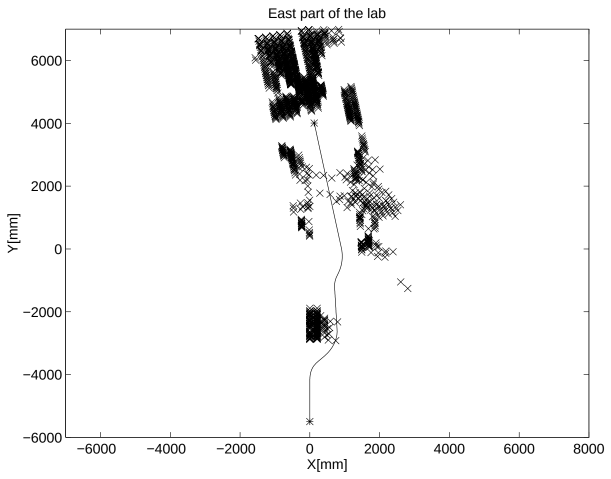

Figure 9: This series of experiments demonstrates the parallel execution of the *GoTo* and *Detect* elementary behaviors. The goals are specified in the global coordinate system. Since we currently do not use the localization procedure, the current position of the robot does not exactly correspond to its real position. In the first experiment the initial position of the mobile base is in the south part of the lab $[0, -5500]$ and the desired goal is specified between the two tables on the other side of the lab at the position $[0, 4000]$. The robot successfully avoided the obstacle in the middle of the room and reached the goal behind the tables, while detecting the tables (as obstacles). The mobile base terminated successfully at the desired destination.

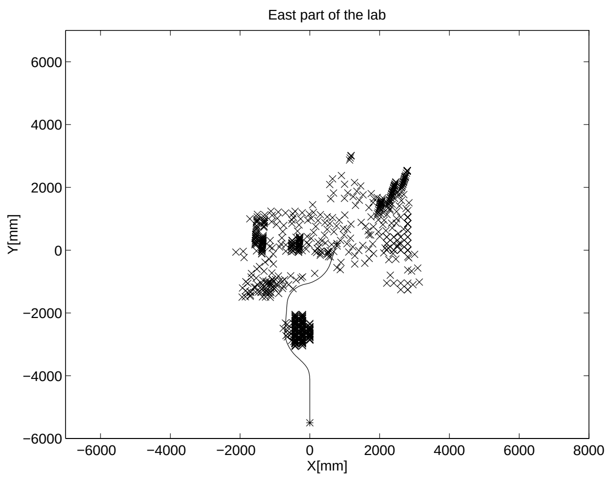

Figure 10: In the second experiment the desired goal location was the same as in the first experiment. The goal was not reached due to the collision with the table. The extent of the table was not estimated correctly by the obstacle detection routine. Since the *Bumper* behavior was not invoked, the mobile base completed the task unsuccessfully and did not retry to avoid the obstacle after detecting the collision.

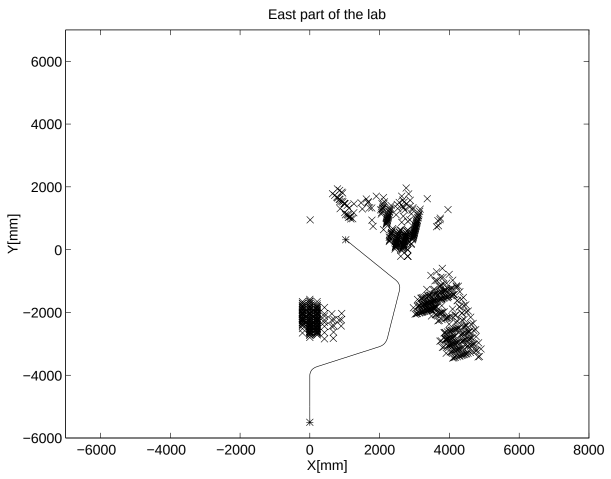

Figure 11: In the third experiment the robot was in exploration mode, with no particular goal. It successfully avoided all the obstacles and continued straight in the heading direction. The detected obstacles on the right were other mobile robots parked on the side of the room. The obstacles in the upper right part of the figure corresponds to the pillar and the tables in the right corner of the lab.

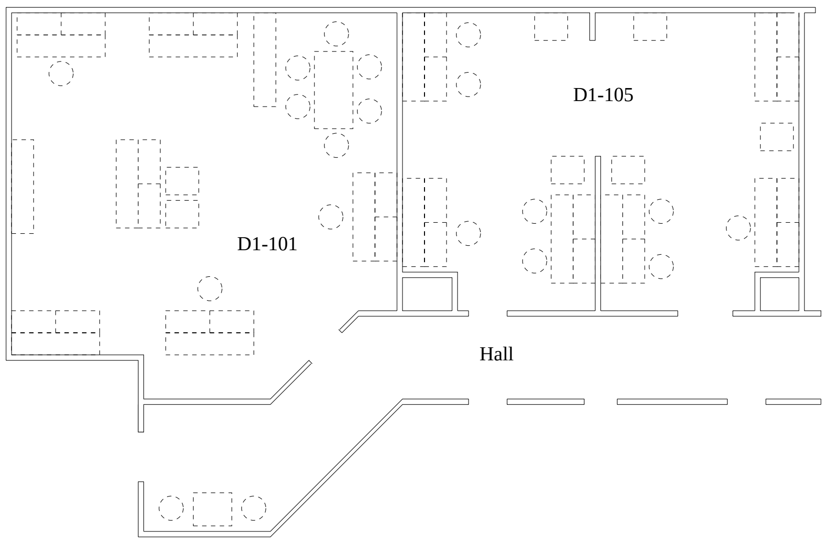

Figure 12: Layout of the environment used for the experiments with the Robuter platform.

the doorway.

To determine the robustness of the developed systems, a long series of experiments has been carried out (in total more than 100 experiments were conducted). An average of 45-50% success was obtained with both systems. The primary cause for mission failures is inadequate modeling of structures in the environment (i.e., something looking like a door is mistakenly taken to be a doorway), and inadequate fusion of the available information.

5 Discussion

In this paper a formal framework for modular composition of behaviors has been outlined and it has been indicated how it may be used for specification of robots used for delivery tasks. The presented framework provides an abstraction mechanism which enables formulation of a reconfigurable 'general' performance of a robot rather than an embodied performance, where specific facilities of the used platform must be included.

To demonstrate the characteristics of the framework for real robots a set of experiments has been carried out. The results from these experiments clearly demonstrate feasibility of the framework for definition of the same

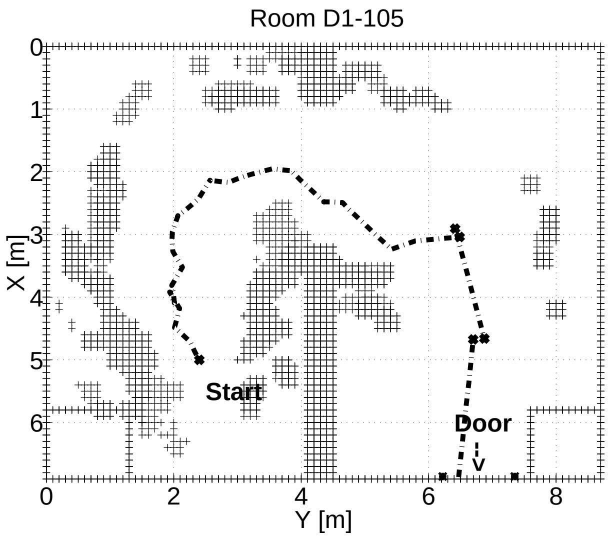

Figure 13: The robot starts out with an 'empty' world model, in which only static structures as shown in Figure 10 are represented. Having planned a path to the doorway the behavior is determined by obstacles in the vicinity of the robot. Once the door should be visible, the robot turns to face the door and servoing on the door is driving the platform.

control strategy for two quite different platforms, provided that the basic capabilities of the platforms are the same. A variety of tasks can be specified by combining some basic strategies.

In the process of defining the tasks and carrying out the experiments is has become clear that the presented framework is very powerful for describing the interaction between behaviors. Given the level of abstraction details, such as handling of noise, are at the same time hidden, which results in a concise and intuitive description. Tasks specified as networks of processes composed via composition operators can easily be compiled into FSM controllers for supervisory control in a DES formalism [8].

During the course of the study it became obvious that the formalism has an implicit assumption that the interactions between different behaviors specified by the desired tasks are correct and well captured by the composition operators. The framework does, however, not provide any guarantees in terms of logical reasoning about the correctness of the given task specification. To have a system with full reasoning capabilities, one needs to incorporate an explicit global model of the environment (database, which is updated using sensory input). This model is presently incorporated implicitly in terms of goals, targets to achieve, and obstacles to avoid. The presented research concentrated mostly at modeling of reactive interactions between behaviors, while future research will address the incorporation of explicit global model of the world and planning.

6 Acknowledgement

The experiments involving the Robuter was using the AMOR control systems developed by Paolo Pirjanian and Hans Blåsvær, AUC.

The presented research has been sponsored by the EC-US explorative action ECUS-003 “Control of Perception”, the EEC Basic Research Project EP-7108-VAP-II “Vision as Process”, and the EEC HCM network SMART, Navy Grant N00014-92-J-1647, AFOSR Grant 88-0296; Army/DAAL 03-89-C-0031PRI; NSF Grants CISE/CDA 88-22719, IRI 89-06770, and ASC 91 0813; and Du Pont Corporation.

References

[1] A. Aho and R. Sethi and J. Ullman. Compilers, Principles, Techniques and Tools. Addison Wesley, 1986.

[2] Y. Aloimonos. Purposive and qualitative active vision. In *Proc. DARPA Image Understanding Workshop*, pages 816–828, 1990.

[3] R. A. Brooks. Intelligence without reason. *Artificial Intelligence*, 1991.

[4] R. A. Brooks. Intelligence without representation. *A.I. Memo*, 1990.

[5] R. C. Arkin. Motor schema based navigation for a mobile robot. In *IEEE Proc. Intl. Conf. on Robotics and Automation*, pages 264–271, 1987.

[6] I. Horswill. A simple, cheap, and robust visual navigation system. In *From Animals to Animats II: Proceedings of the Second International Conference on Simulation of Adaptive Behaviour.* MIT Press, 1993.

[7] J. Košecká and R. Bajcsy. Discrete Event Systems for Autonomous Mobile Agents. *Journal of Robotics and Autonomous Systems*, pages 187–198, No. 12, 1994.

[8] J. Košecká. Supervisory Control Theory of Autonomous Mobile Agents, Ph.D. Thesis, GRASP Laboratory, University of Pennsylvania, February 1996.

[9] J. Košecká and H. Ben-Abdallah. An Automaton Based Algebra for Specifying Robotic Agents. *AMAST Workshop on Real-Time Systems*, Salt Lake City, 1996 (submitted).

[10] D. M. Lyons. A formal model of computation for sensory-based robotics. *IEEE Transactions of Robotics and Automation*, 5(3):280 – 293, 1989.

[11] P. Maes. Situated Agents Can Have Goals. In *Designing Autonomous Agents*, Elsevier Science Publishers, 1990.

[12] P. Pirjanian, H. Blåsvær, and H. I. Christensen, AMOR: An Autonomous Mobile Robot System. Proc. IEEE Conf. on Systems, Man and Cybernetics, San Antonio, pages 2266–2271. Oct. 1994.

[13] P. Pirjanian and H. I. Christensen, "Hierarchical Control using Heterogeneous Models", in Environmental Modeling, H. Bunke, H. Noldemeyer, & T. Kanade (Eds), World Scientific Press, Singapore, pages 344–361, Nov. 1995.

[14] P. J. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–97, January 1989.

[15] J. K. Tsotsos. Behaviorist intelligence and the scaling problem. Technical report, Department of Computer Science, University of Toronto, 1986.

[16] A. Flynn. Combining Sonar and Infrared Sensors for Mobile Robot Navigation. Intl. Journal of Robotics Research, Dec. 1988.