A Behavior-Based Architecture for Realistic Autonomous Ship Control

Adam Olenderski and Monica Nicolescu
Robotics Research Laboratory
Dept. of Computer Science and Engineering
University of Nevada, Reno
olenders,monica@cse.unr.edu

Sushil J. Louis
Evolutionary Computing Systems Lab
Dept. of Computer Science and Engineering
University of Nevada, Reno
sushil@cse.unr.edu

***Abstract*— Game environments provide a good domain for serious simulations such as those used in training Navy conning officers. Currently, a typical training scenario requires multiple personnel to play each of the boats and thus is expensive. We propose an approach to addressing this issue by developing intelligent, autonomous controllers for each boat. Significant challenges toward achieving these goals are the realism of behavior exhibited by the automated boats and their real-time response to change. In this paper we describe a control architecture that enables the real-time response of boats and the repertoire of realistic behaviors we developed for this application. We demonstrate the capabilities of our system with experimental results.**

Keywords: Training Games

I. Introduction

Virtual/game environments provide a good application area both for entertainment and for serious simulations such as those used in training. In this paper we focus on an application for training conning officers and we describe our approach to creating a robust and effective training system. The goal for such systems is to teach conning officers to drive big ships in the context of high-traffic, potentially dangerous situations. Developing such a system poses significant challenges and in this paper we will present an integrated solution to three of the major requirements for a successful training simulator.

A first challenge is the *efficiency of the training system*, in terms of the personnel required for running the system, and thus its cost. Currently, a typical training scenario requires multiple personnel to play the part of each of the traffic boats and is thus expensive and difficult to coordinate. In this paper we propose an approach to reducing the time and effort required for this training by automating the behavior of the boats in the simulation (other than the ship driven by the student officer). Taking inspiration from the field of autonomous robotics and we developed an *authoring tool* that enables the development of intelligent, autonomous controllers that drive the behavior of a large number of boats. We assume that a set of primitive behaviors (e.g., *avoidance*, *maintain station*, etc.) are available as basic navigation capabilities, and the authoring tool allows the construction of controllers for complex tasks from these underlying behaviors. With this, our system eliminates the necessity of having large number of personnel for a single student's training, significantly reducing the costs involved.

A second challenge is the *readiness of response* of the automated boats when facing changing situations as a result of a trainee's or other boats' actions. This requires that the controllers be able to act in real-time, while continuing the execution of the assigned tasks. To achieve this goal we will use Behavior-Based Control (BBC) [1], a paradigm that has been successfully used in robotics. BBC is an effective approach to robot and autonomous agent control due to its modularity and robust real-time properties. While BBC constitutes an excellent basis for our chosen domain, developing behavior-based systems requires significant effort on the part of the designer. Thus, automating the process of controller design, as our authoring mechanism will allow, becomes of key importance. In our work, the instructor uses the authoring tool to create challenging scenarios for the student conning officers, allowing for a fast and efficient transfer of knowledge from the expert to our automated system.

The third challenge is the *realism of the behavior* exhibited by the autonomous boats involved in the simulation, due to the fact that any behavior that departs from standard boat navigation techniques would have a detrimental impact on the students' training experience. Thus, this requirement imposes new constraints on how the boats' underlying behaviors are implemented, in contrast with typical behavior-based systems in which almost any behavior that achieves the desired goals is good. To implement these realistic capabilities we acquired expert knowledge of ship navigation [2] and we encoded this information within a behavior-based framework.

The implementation of the game engine and the graphics display are also main components of the entire system, but they are outside the scope of this paper. The work presented here, a part of a larger scale project, focuses on the aspects related to autonomous boat control, as previously described.

The remainder of the paper is structured as follows: Section II describes related approaches to our work and Section III describes our simulation environment. Section IV presents our behavior and controller representation and Section V presents our behavior repertoire. Sections VI and VII describe the details of our authoring tool. We present our experimental results in Section VIII and conclude with a summary of the proposed approach in Section IX.

II. Related Work

Simulation systems for training have received significant interest in recent years. Representative examples include flight simulators [3] and battlefield simulators [4]. In contrast with the above approaches, the system we propose in this paper provides an authoring mechanism to facilitate the development of autonomous controllers for the agents involved in the simulation. Our approach is inspired by the *programming by demonstration* paradigm, which has been employed in a wide range of domains, from intelligent software systems [5] , to agent-based architectures [6], to robotics [7]

In the mobile robotics domain, which is the inspiration for our system, successful approaches that rely on this methodology have demonstrated learning of reactive policies [8], trajectories [9], or high-level representations of sequential tasks [10]. These approaches employ a *teacher following* strategy, in which the robot learner follows a human or a robot teacher. Our work is similar to that of [11] , who perform the demonstration in a simulated, virtual environment. Furthermore, our work relates to teleoperation, a very direct approach for teaching a robot by demonstration. Teleoperation can be performed using data gloves [12] or multiple DOFs trackballs [13]. These techniques enable robots to learn motion trajectories [14] or manipulation tasks (e.g., [15]). Using such "lead-through" teaching approaches [16] requires that the demonstration be performed by a skilled teacher, as the performance of the teacher in demonstrating the task has a great influence on the learned capabilities. Another difficulty that may arise is that the teleoperation may be performed through instruments that are different than what the human operator would use in accomplishing the task. Also, the actual manipulation of the robot may influence the accuracy of the demonstration. In contrast with these approaches, our work uses an interface, which allows the transfer of expert knowledge through standard computer input devices.

III. Simulation Environment

We use a 3D simulation environment, called *Lagoon* (Figure 1), that was developed by a larger team at the University of Nevada, Reno. This environment allows for simulating a wide range of boats, from small cigarette boats to medium ships, such as sailboats, to large ships, such as destroyers and aircraft carriers. All boats have realistic physics, which the controllers take into account when autonomously driving the ships.

Within this architecture, each boat can be controlled via the *Authoring* panel (Figure 1, right side of screen). When an entity is selected the panel and its associated behaviors refer to that entity. Whenever a new entity is selected, the behavior information for that new entity is displayed. There are 7 primitive behaviors, as described in Section V: *approach*, *maintain station*, *ram*, *move to*, *avoid entity*, *avoid land* and *fire*. The top level of the *Authoring* panel displays information about the selected entity: name, current speed and course,

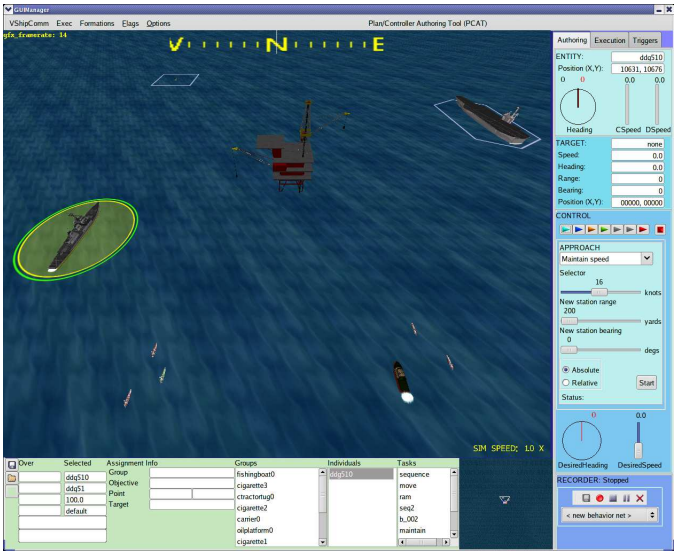

Fig. 1. Lagoon Simulation Environment

desired speed and course, and position. The lower section displays information about the target ship (when applicable - Section V): name, speed, heading, position, range and bearing to target. The bottom section of the *Authoring* panel provides manual controls for actuating the selected entity, as an alternative to behaviors.

IV. Behavior-Based Control Architecture

Behavior-based control (BBC) [1] has become one of the most popular approaches to embedded system control both in research and in practical applications. Behavior-based systems (BBS) employ a collection of concurrently executing processes, which take input from the sensors or other behaviors, and send commands to the actuators. The inputs determine the activation level of a process: whether it is on or not, and in some systems by how much. These processes, called *behaviors*, represent time-extended actions that aim to achieve or maintain certain goals, and are the key building blocks for intelligent, more complex behavior.

In this paper we use a *schema-based representation of behaviors*, similar to that described in [17]. This choice is essential for the purpose of our work, since it provides a continuous encoding of behavioral responses and a uniform output in the form of vectors generated using a potential fields approach.

For the *controller representation* we use an extension of the standard Behavior-Based Systems we developed, which provides a simple and natural way of representing complex tasks and sequences of behaviors in the form of networks of *abstract behaviors*. In a behavior network, the links between behaviors represent precondition-postcondition dependencies, which can have three different types: *permanent*, *enabling* and *ordering*. Thus, the activation of a behavior is dependent not only on its own preconditions (particular environmental states), but also on the postconditions of its relevant predecessors (*sequential preconditions*). More details on this architecture can be found in [18].

The *abstract behaviors* embed representations of a behavior's goals in the form of abstracted environmental states, which are continuously computed from the sensory data. This is a key feature of our architecture, and a critical aspect for learning from experience. *In order to learn a task the robot has to create a link between perception (observations) and the robot's behaviors that would achieve the same observed effects.*

In our system, a controller could potentially have multiple concurrently running behaviors. For such situations, our system uses the following action selection mechanism. Each behavior, including the avoid entity and avoid land behaviors, computes a speed and a heading for the actuators. If more than one non-avoidance behavior is active at one time, the speed and heading returned by each active behavior, represented as a vectors, are added by vector addition. The resulting speed and heading are passed to the actuators. However, if one of the avoidance behaviors is active along with other non-avoidance behaviors, the vector from the avoidance behavior is not fused with the other behaviors' output. In such a case, the other behaviors' vectors are summed and sent to the avoidance behavior as a *"suggestion"*. If the avoidance behavior finds that the suggested heading and speed do not create a risk of collision, then the behavior will simply pass the suggested values directly to the actuators. If, on the other hand, the avoidance behavior finds that the suggested heading and speed will cause a collision, the avoid behavior will find an alternative heading and speed that is as close to the suggested values as possible without causing a collision. These alternative values will then be passed directly to the actuators.

If both avoidance behaviors are active at the same time as other behaviors, then each avoidance behavior (land and entities) will find an appropriate set of alternative values based on the same set of suggested values from the other behaviors. However, instead of passing these values directly to the actuators, the outputs of the two avoid behaviors will be fused as described above. The result will then be passed to the actuators.

V. Behavior Repertoire

A. Description

The most important skill necessary for our behavior repertoire relates to vessel navigation, particularly where realistic navigation is concerned. In the agents/robotics domain, what is most important is to design behaviors or skills that achieve certain desired goals for the task, irrespective of how those goals are reached. However, ship handling and navigation have to obey the *"rules of the road"'* [2], thus imposing significant constraints on how the ship's basic capabilities need to be designed. An additional constraint is that the level of granularity for these skills has to be appropriate to allow for the types of tasks that the boats would perform. As a result of these requirements, the main behaviors that we identified as necessary are the following:

Fig. 2. Behavior Panels: Approach, Maintain Station, Move To, Ram

- **Maintain Station:** The goal of this behavior is to make a *maneuvering ship* (such as a destroyer) maintain a certain station (distance and bearing) with respect to a *reference ship* (such as an aircraft carrier). There are five main parameters to this behavior: 1) the reference ship, 2) the way in which the maneuver is to be performed: constant speed, constant course or in a given amount of time; 3) the value for the maneuver (i.e., speed, course or time), 4) the new stationing position in terms of distance and bearing, and 5) the type of station, i.e., if the location is relative or absolute. When executing this behavior, the *maneuvering ship* gets into station, after which it continues to track the *reference ship's* course and speed. If the *reference ship* changes course or speed, the behavior re-computes the necessary actions for the *maneuvering ship*, in order to maintain the desired station.
- **Approach:** The goal of this behavior is to get a *maneuvering ship* to reach a certain station (distance and bearing) with respect to a *reference ship*. This is similar to the *Maintain Station* behavior and has the same input parameters, the only difference being that in *approach* the *maneuvering ship* will not maintain the station after reaching it.
- **Move To:** The goal of this behavior is to get a *maneuvering ship* to a specific location, in (X, Y) coordinates, in the world. This behavior takes three main parameters: 1) the way in which the maneuver is to be performed: with a constant speed or in a given amount of time; 2) the value for the maneuver (i.e., speed or time), and 3) the new (X, Y) position.
- **Ram:** The goal of this behavior is to have a *maneuvering ship* hit a *target ship*. This behavior takes three main parameters: 1) the target ship, 2) the way in which the maneuver is to be performed: constant speed, constant course or in a given amount of time, and 3) the value for the maneuver (i.e., speed, course or time).
- **Fire:** The goal of this behavior is to direct the weapon fire from a *maneuvering ship* to a *target ship*. The sole parameter

of this behavior is the ship toward which to direct the fire.

• **Avoid Entity:** The goal of this behavior is to navigate a boat such that all collisions with other boats are avoided, in a manner consistent with the standard navigation rules.

• **Avoid Land:** The goal of this behavior is to avoid collisions with land, in a manner consistent with the standard navigation rules.

As previously mentioned, simply achieving the goals of these behaviors is insufficient if the boats do not obey the ship navigation rules. In addition, the large number of rules in the navigation domain makes very challenging the task of implementing them in a simple, modular manner. The following subsections describe the approach we took to implementing the main navigation rules into our behavior-based system.

B. Navigation

In the ship navigation domain, the course and speed of a ship is computed using a *maneuvering board*, or *moboard*. This allows the crew to obtain the course and/or speed that the ship should take to get into the desired position with respect to another boat.

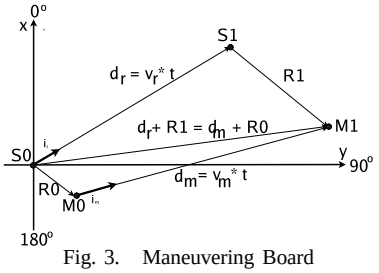

Fig. 3. Maneuvering Board

The *maneuvering ship* is placed at the center of the board, and the location, course and speed of the *reference ship* are plotted with respect to the center. With this diagram, the three modes of maneuvering can be performed: 1) constant speed (course and time to completion are computed), 2) constant course (speed and time to completion are computed) and 3) given time (course and speed are computed). The moboard is very useful for manual computation, such as that performed on the ship. For our purpose, we represent the problem as the relative motion of two objects in Cartesian coordinates, assuming that both ships maintain the same speed and course during the maneuver, as shown in Figure 3, where:

• S_0, S_1 - position of the *reference ship* at the beginning and end of the maneuver

• M_0, M_1 - position of the *maneuvering ship* at the beginning and end of the maneuver

• R_0, R_1 - displacement between the two ships at the beginning and end of the maneuver

• d_r - displacement of the *reference ship* over the course of the maneuver

• d_m - displacement of the *maneuvering ship* over the course of the maneuver

• i_m - unit vector representing the direction of the *maneuvering ship*

• v_m - velocity vector of the *maneuvering ship*

• i_r- unit vector representing the direction of the *reference ship*

• v_r - velocity vector of the *reference ship*

• t - time to complete the maneuver

The equation of motion for the *maneuvering* and *reference* ships is:

$$i_r \, ||v_r|| \, t = D + i_m \, ||v_m|| \, t \tag{1}$$

where D is the relative motion vector $(R_0 - R_1)$. From Equation 1 we can find the solutions to the three types of maneuvers, as follows:

1) Constant speed. Keeping $||v_m||$ constant, we compute the course (i_m) and the time to completion (t), by solving the system of two equations that results from projecting Equation 1 onto the (X, Y) coordinates.

2) Constant course. Keeping the course (i_m) constant, we compute the speed ($||v_m||$ and the time to completion (t), by solving the system of two equations that results from projecting Equation 1 onto the (X, Y) coordinates.

3) Constant time. Keeping t constant, we compute the course (i_m) and speed ($||v_m||$), by solving the system of two equations that results from projecting Equation 1 onto the (X, Y) coordinates.

Due to the fact that the simulated ships have realistic physics, we use a PD (proportional derivative) controller to slow down the ships as they approach their goal destination. The speed sent to the actuators is computed with the formula:

$$v_{rFinal} = v_r + K_p DiffSpeed + K_d * DiffAccel \tag{2}$$

where v_r is the speed computed from Equation 1, K_p and K_d are proportional and respectively derivative constants and $DiffSpeed$ and $DiffAccel$ are the difference in speed and acceleration between the maneuvering ship and the reference ship.

All four navigation behaviors, *approach*, *maintain station*, *move to* and *ram*, use the above equations, parameterized to fit their requirements.

C. Entity Avoidance

In typical robot/agent-based controllers, the role of the obstacle avoidance behavior is simply to avoid all obstacles. In most cases this is achieved by turning left when there is an obstacle to the right or by turning right when there is an obstacle to the left. To accurately mimic the actions of a human driving a ship, several important constraints apply.

The most important navigation rule for avoidance is that a human looks ahead in time to determine whether he will hit an obstacle, which cannot be achieved by a purely reactive controller. We implement such looking ahead capabilities into our avoidance behavior, as explained next.

For the obstacle avoidance behavior, each ship in the world (other than the avoiding ship) is represented by an ellipse that is centered about that ship's center of mass, rotated such that the major axis of the ellipse is parallel to that ship's

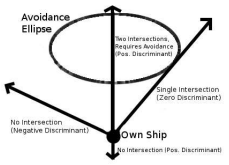

Fig. 4. Obstacle Avoidance

heading, and whose major and minor radii are proportional to the length and width of the ship, respectively (Figure 4). The avoiding ship is represented simply as a point. Since the speed and heading of the avoidance ship as well as the speeds and headings of all other ships are known (in real life, this information can be obtained through passive sensing), we can look forward in time to find the positions of the ships at some point in the future, assuming that their speed or direction does not change. Along with this information, we can also obtain the equations of the ships' corresponding ellipses. A collision is predicted to occur when the point representing the own ship falls onto the ellipse representing another ship, or, to put it another way, when the (x,y) point representing the own ship satisfies the equation of one of the avoidance ellipses. Based on this information, the avoidance behavior finds whether a collision is imminent, the time to collision, and a revised speed or heading for the own ship that will avoid collisions with other ships.

The (x, y) position of any ship in time can be expressed as a pair of parametric equations, with time as a parameter. An ellipse can be uniquely described by its center point, its minor and major radii, and its orientation. For any given ship, we can safely assume that the radii will remain constant, as they are proportional to the dimensions of the ship, which are constant. Furthermore, we assume that the orientation of an avoidance ellipse will remain constant in the future, since ships usually do not change heading without reason. If one of these assumptions turns out to be false, such as when a ship is turning, the avoidance ellipse is recalculated based on the most recent values. The center point of the ellipse, like the point representing the own ship, can be calculated for some point in the future using the ship's current heading, position, and speed as in Equation 3:

$$\begin{aligned} h &= v_t * t * cos\theta_t + x_{0t} \\ k &= v_t * t * sin\theta_t + y_{0t} \end{aligned} \tag{3}$$

where h is the x coordinate of the ellipse's center, k is the y coordinate of the ellipse's center, v_t is target ship's velocity, t is the amount of time to look into the future, θ_t is the target's heading, x_{0t} is the x coordinate of the target's current position (x coordinate of current ellipse center), and y_{0t} is the y coordinate of the target's current position (y coordinate of current ellipse center).

The equation of a boat's avoidance ellipse (centered at (h, k) and rotated by θ_t) is given by equation 4 below:

$$\begin{aligned} &\frac{((x_m-h)*cos\theta_t+(y_m-k)*sin\theta_t)^2}{a^2} + \\ &\frac{((y_m-k)*cos\theta_t-(x_m-h)*sin\theta_t)^2}{b^2} = 1 \end{aligned} \tag{4}$$

where x_m, y_m, h, k, and θ_t are the same as above, a is the major radius of the ellipse, and b is the minor radius of the ellipse.

We can find out if a point will fall on an ellipse by setting the x_m and y_m values representing the position of the points on the ellipse to the equations for the (x, y) position of the own ship and solving for time. The result is a quadratic equation in t:

$$g(t, x_{0m}, y_{0m}, v_m, v_t, \theta_t, h, k, a, b) = 0 \tag{5}$$

Based on the two solutions of Equation 5, the following cases occur:

1) If *both solutions are imaginary* (negative discriminant), then there will be no intersection at any time between the point and the ellipse, indicating no risk of collision.

2) If *both solutions are equal* (discriminant equal to 0), then there is only one point of intersection with the ellipse, meaning that the own ship is traveling tangentially to the avoidance ellipse of the ship to be avoided. In this case, there is no danger of the ships themselves colliding, as the own ship never makes its way inside the avoidance ellipse of the other ship.

3) If *the discriminant is positive*, there are three possible cases: i) If both solutions are positive, the own ship will intersect the avoidance ellipse twice: once to enter the ellipse and once more to exit it (in this case, some evasive action must be taken to avoid a collision); ii) if both solutions are negative, there is no risk of a collision (intuitively, this would mean that there was an intersection at some point in the past, but there is no danger in the future); and iii) if one solution is negative and one solution is positive, then the own ship has intersected the avoidance ellipse once in the past, and will intersect it once more in the future (this indicates that the own ship is within the avoidance ellipse of the other ship, and must take immediate and drastic evasive maneuvers to avoid a collision and leave the avoidance ellipse.)

If evasive action must be taken, the following options are considered. If the own ship is within the avoidance ellipse of another ship, this is seen as an emergency situation and the own ship will try to move behind and away from the other ship as quickly as possible. If, however, the danger of collision is sufficiently far away in the future, the obstacle avoidance behavior can make smaller adjustments to the heading or speed of the own ship to eliminate the danger of colliding with the other ship. To achieve this, the obstacle avoidance behavior computes a heading and/or speed that results in a zero discriminant for Equation 5, (case 2 above). Navigation rules favor a change speed rather than heading, thus the behavior first attempts to find a new speed that satisfies the constraint:

$$f(x_{0m}, y_{0m}, x_{0t}, y_{0t}, v_m, v_t, \theta_t, a, b) = 0 \tag{6}$$

This is a quadratic equation with respect to speed, with two solutions. If both solutions are valid speeds (positive and less than or equal to the maximum speed of the ship), the behavior returns the highest speed. If only one solution is

a valid speed, that speed is used. If neither solution is valid, then the heading must be changed to avoid a collision. To find a valid heading, the behavior first finds whether the collision would still be imminent if the own ship turned five degrees to the left. If not, then the behavior continues to test potential headings, in increments of five degrees, until it finds one that avoids a collision, at which point it uses the same process to find a corresponding heading to the right of the current heading. This results in two headings (one to the right, and one to the left) that avoid a collision. The heading closest to the current heading is the one passed to the actuators.

The solution to avoid one ship could potentially generate collisions with others. Our approach uses a mechanism to deal with avoiding multiple ships, as follows: for every ship on a collision course with the own ship, the avoidance behavior puts in a list all the valid speeds and headings that will avoid that ship. After all the ships have been analyzed and the list is built, the behavior iterates through the list and eliminates the routes that collide with other ships. The only elements remaining in the list will be the speeds or headings that avoid collisions with all the other ships in the simulation. Since speed changes take priority over heading changes, if there is at least one speed remaining in the list, that speed will be passed to the actuator with the current heading. If there is more than one potential speed in the list, the highest speed is passed to the actuators. If no speed changes remain in the list, then the potential heading that is closest to the current heading is passed to the actuators along with the current speed.

D. Land Avoidance

To determine whether or not a ship is in danger of grounding itself, the avoid land behavior uses the speed of the ship, as indicated in [2]. The behavior uses this speed to determine the approximate distance that the ship can travel in four minutes. Next, it checks if there is any land within that distance, in all directions from the ship. If no land is present in that area, there is no land to avoid. However, if there is land in front of the own ship, the behavior computes a heading that will take the ship away from land and a speed that makes the nearest land be outside of the four minute range. To achieve this, the behavior uses the distance and bearing to the nearest land in the front 180-degree field of view. The new speed is calculated to be that distance divided by four minutes, to ensure that the nearest land will lie outside of the four minute range. The new heading is calculated to be that bearing plus or minus 90 degrees, whichever is closer to the current heading. This ensures that instead of continuing to head toward land, getting slower and slower until the ship grounds itself, the ship will turn parallel to the land, following its contour until there is no more land to avoid or until the land avoidance behavior is turned off.

VI. TRIGGERS

Triggers are a mechanism that allows the user to indicate important situations in the simulation, typically with the purpose of changing a boat's behavior when that situation occurs. A trigger is created through the triggers panel of our interface (Figure 5). We provide the following types of triggers: distance between entities, entities within a certain range, hull strength of a particular ship, or an arbitrary flag. The panel allows the user to specify the parameters for each trigger, such as, for example, the entities in the simulation between which the distance is to be monitored. The name and current value (updated every tick) of all created triggers is displayed on the triggers panel main window. Triggers can be created, monitored, or deleted at any time during either authoring or execution.

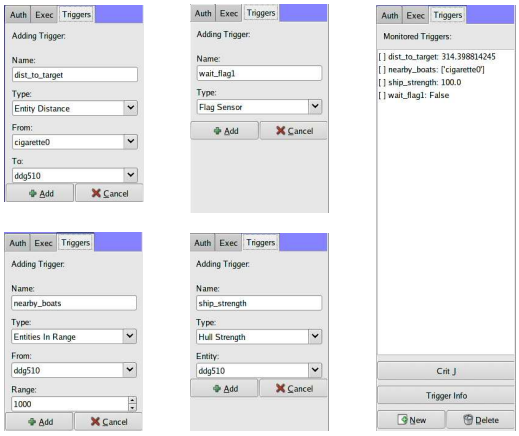

Fig. 5. Trigger types and panel

The distance between entities trigger is included for convenience, as it does not require user input during authoring and many maritime maneuvers depend on the distances between particular ships. However, for a more general event that cannot be described in terms of the distance between two entities or the damage taken by a particular ship, there are arbitrary flag triggers as well. These work similarly to the other triggers, in that they are user created and indicate when a controller should change behavior. However, these triggers are binary flags, meaning that they can only hold one of two values: true or false. Also, these triggers require user input not only during authoring, but also during execution. A flag trigger usually represents an event that the user will have to specify himself, such as the start of a scenario. For example, to add a flag trigger to start a scenario, the user would create a trigger named ”Start Scenario” before authoring. While recording, when the user is ready to begin the scenario, he would navigate to the triggers panel, click the check-box next to the ”Start Scenario” trigger, click the Critical Juncture button, and change the behavior of the ship he/she is controlling. This informs the system that that behavior should only be activated after the ”Start Scenario” flag has been set. Then, when the controller is executed, it will wait until the user explicitly activates that flag before continuing. This is done by navigating to the triggers panel, clicking the check-box next to the desired flag trigger, and clicking the Critical Juncture button.

VII. Authoring

During authoring, the instructor starts or stops the relevant behaviors using the Control Panel interface. While these behaviors are executed, the authoring tool continuously monitors the status of the behaviors' postconditions. To build the task representation (controller) we add to the network task representation an instance of all behaviors whose postconditions have been detected as true during the demonstration, in the order of their occurrence (on-line stage). At the end of the teaching experience, the intervals of time when the effects of each of the behaviors were true are known, and are used to determine if these effects were active in overlapping intervals or in sequence. Based on the above information, the algorithm generates proper dependency links between behaviors (i.e., *permanent*, *enabling* or *ordering*) (off-line stage). This one-shot learning process is described in more detail in [19]. The only difference in the work presented here is that the construction of the task representations was done off-line.

While authoring a controller, if triggers are needed, the user first creates all the triggers to be used during that session. While a scenario is being recorded, if the user wishes to indicate changes in behavior based on some particular event in the world, he/she navigates to the triggers panel, clicks on the check-box next to the appropriate trigger to indicate its relevance, clicks the Critical Juncture button, and then changes to the desired behavior. During playback of the same scenario, the system will monitor the state of the execution trigger approaches that of the user-created trigger, the system switches the boat's behavior according to the demonstration.

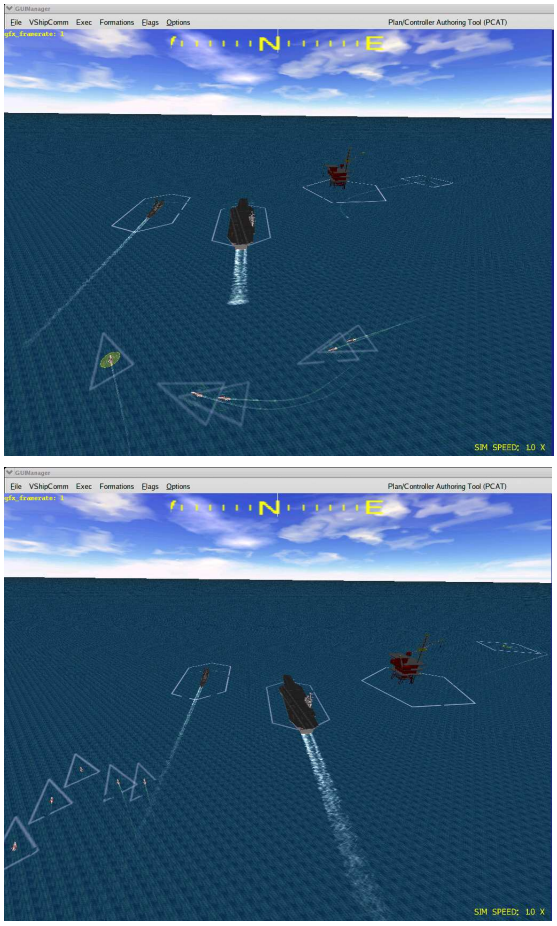

Fig. 6. Behavior performance evaluation: Destroyer maintains station with respect to aircraft carrier; V-formation avoids big ships while moving west.

VIII. Experimental Results

In this section we describe the performance of our system during behavior performance testing and during controller authoring. This experimental validation demonstrates the main capabilities of our system: autonomous control of multiple boats, compliance to navigation standards and authoring of complex controllers.

A. Behavior Performance

The behaviors that involved the underlying navigation capability (*approach*, *maintain station*, *ram*, and *move to* have been thoroughly tested throughout the experiments listed below. Their performance correctly and faithfully demonstrated compliance to the rules of ship navigation.

We successfully tested *avoid entities* in numerous situations, including the following: 1) moving own-ship from one side of stationary/moving target ship to the other side, 2) moving own-ship from one end of stationary/moving target ship to other end, 3) moving own-ship from one side of two stationary/moving ships whose ellipses overlap to the other side and 4) moving own-ship from one side of a crowded group of moving and stationary ships to the other side. These represent the most probable situations to be encountered by a boat in high-traffic areas.

We successfully tested the *avoid land* behavior in the following representative situations: 1) moving own-ship from point far away from land to point near land, 2) moving own-ship from point near land to point farther away from land, 3) moving own-ship to a point within a land mass (triggered avoidance and did not move onto land), 4) moving own-ship to a point on the other side of a land mass.

By attaching *Maintain station* behaviors to several boats, in a layout such as required by a formation, we enabled autonomous group behavior, for large number of boats. Currently, we can organize any number of boats (as allowed by the computational power of the computer) in the following formations: *line*, *row*, and *V-formation*. The boats involved in a group maintain their assigned formation while performing other tasks, such as moving to a new location, or approaching a target. While performing these maneuvers, the group avoids obstacles, keeping the formation together. Figure 6 shows a group of 5 boats in *V-formation*, moving to the left of a destroyer and aircraft carrier, while performing obstacle avoidance. The bottom figure shows the formation regrouped after avoidance. The boats can dynamically switch between formations.

B. Controller Authoring

We have performed a large number of authoring experiments (more than 10 different scenarios), with all scenarios being learned correctly. Below we list a few examples, which are most relevant for the training of conning officers.

1. ZigZag Attack: A small boat moves into position with respect to a big ship, then approaches a group that maintains distant station of that ship. The group moves in V-formation. On a flag trigger the small boat approaches the big boat and maintains station until a second trigger goes off. Next, the small boat rejoins the boat group in the distance. When a third flag trigger goes off, the small boat re-approaches the big boat and maintains close station to it. When getting with a given range (distance trigger), the small boat moves into position behind the big ship and rams it.

2. Defend: A destroyer (escort) maintains station with respect to an aircraft carrier to be defended. A small cigarette boat approaches the destroyer. When the cigarette boat comes within a buffer range of the aircraft carrier (e.g., 1000 yards) a distance trigger is activated, after which the escort's behavior changes from maintain station on carrier to ram cigarette boat.

3. Attack-Distract Two boats take turns distracting so one or both can ram a target. Boat *A* maintains a position in front and slightly to the side of the Target at a long range, as if it wants to be seen but not considered a threat. Boat *B* starts to the side of the target and slowly begins to collide with the Target. Once *B* is with in a certain range hopefully gaining the attention of the Target, *A* starts a full-speed ram. Once *A* starts the ram, *B* stops its approach and backs off.

IX. Conclusion

In this paper we presented an approach to efficient and realistic design of serious game simulators, with application to ship navigation. The goal of this system is to provide the infrastructure needed to train conning officers to drive big ships in the context of high-traffic, potentially dangerous situations. Developing such a system poses significant challenges and in this paper we presented an integrated solution to three of the major requirements for a successful training simulator: 1) the *efficiency of the training system*, 2) the *readiness of response* of the boats and 3) the *realism of the behavior* of the automated boat controllers. To address these challenges we developed an *authoring tool* that enables the development of intelligent, autonomous controllers that drive the behavior of a large number of boats.This eliminates the necessity of having large number of personnel for a single student's training, significantly reducing the costs involved. We developed a Behavior-Based Control architecture that provides responsive automated controllers, and we incorporated expert ship navigation knowledge to provide realistic behavior for the automated boats. To demonstrate our approach, we presented experimental results describing the main capabilities of our system.

X. Acknowledgments

The authors would like to acknowledge the significant contribution of Sergiu Dascalu, Chris Miles, Ryan Leigh, and Juan Quiroz, from the University of Nevada, Reno. This work was supported by the Office of Naval Research under grant number N00014-05-1-0709.

References

[1] R. C. Arkin, *Behavior-Based Robotics*. CA: MIT Press, 1998. [Online]. Available: http://www.usc.edu

[2] J. John V. Noel, Ed., *Knight's Modern Seamanship*. John Wiley and Sons, 1988.

[3] M. Tambe, L. W. Johnson, R. M. Jones, F. V. Koss, J. E. Laird, P. S. Rosenbloom, and K. Schwamb, "Intelligent agents for interactive simulation environments," *AI Magazine*, vol. 16, no. 1, pp. 15–39, 1995.

[4] R. B. Calder, J. E. Smith, A. J. Courtemarche, J. M. F. Mar, and A. Z. Ceranowicz, "Modsaf behavior simulation and control," in *Proceedings of the Second Conference on Computer Generated Forces and Behavioral Representation*, July 1993.

[5] H. Lieberman, *Human-Computer Interaction for the New Millenium*. ACM Press/Addison-Wesley, 2001, ch. Interfaces that Give and Take Advice, pp. 475–485.

[6] R. H. Angros, "Learning what to instruct: Acquiring knowledge from demonstrations and foccused experimentation," Ph.D. dissertation, University of Southern California, May 2000.

[7] S. Schaal, "Learning from demonstration," in *Advances in Neural Information Processing Systems 9*, M. Mozer, M. Jordan, and T. Petsche, Eds. MIT Press, Cambridge, 1997, pp. 1040–1046.

[8] G. Hayes and J. Demiris, "A robot controller using learning by imitation," in *Proc. of the Intl. Symp. on Intelligent Robotic Systems*, Grenoble, France, 1994, pp. 198–204.

[9] P. Gaussier, S. Moga, J. Banquet, and M. Quoy, "From perception-action loops to imitation processes: A bottom-up approach of learning by imitation," *Applied Artifi cial Intelligence Journal*, vol. 12(78), pp. 701–729, 1998.

[10] M. N. Nicolescu and M. J. Matarić, "Natural methods for robot task learning: Instructive demonstration, generalization and practice," in *Proc., Second Intl. Joint Conf. on Autonomous Agents and Multi-Agent Systems*, Melbourne, Australia, July 2003.

[11] J. Aleotti, S. Caselli, and M. Reggiani, "Leveraging on a virtual environment for robot programming by demonstration," *Robotics and Autonomous Systems*, vol. 47, pp. 153–161, 2004.

[12] R. Voyles and P. Khosla, "A multi-agent system for programming robots by human demonstration," *Integrated Computer-Aided Engineering*, vol. 8, no. 1, pp. 59–67, 2001.

[13] M. Kaiser and R. Dillmann, "Building elementary robot skills from human demonstration," in *Proc., IEEE Intl. Conf. on Robotics and Automation*, Minneapolis, Minnesota, apr 1996, pp. 2700–2705.

[14] N. Delson and H. West, "Robot programming by human demonstration: Adaptation and inconsistency in constrained motion," in *Proc., IEEE Intl. Conf. on Robotics and Automation*, Minneapolis, MN, apr 1996, pp. 30–36.

[15] J. Yang, Y. Xu, and C. S. Chen, "Hidden markov model approach to skill learning and its application in telerobotics," in *Proc., Intl. Conf. on Intelligent Robots and Systems*, Yokohama, Japan, 1993, pp. 396–402.

[16] D. J. Todd, *Fundamentals of Robot Technology*. John Wiley and Sons, 1986.

[17] R. C. Arkin, "Motor schema based navigation for a mobile robot: An approach to programming by behavior," in *IEEE Conference on Robotics and Automation, 1987*, 1987, pp. 264–271.

[18] M. N. Nicolescu and M. J. Matarić, "A hierarchical architecture for behavior-based robots," in *Proc., First Intl. Joint Conf. on Autonomous Agents and Multi-Agent Systems*, Bologna, Italy, July 2002, pp. 227–233.

[19] ——, "Learning and interacting in human-robot domain," *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans, Special Issue on Socially Intelligent Agents - The Human in the Loop*, vol. 31, no. 5, pp. 419–430, 2001.