In *The Truman Show*: Generating Dynamic Scenarios in a Driving Simulator

Ingo Wassink, Betsy van Dijk, Job Zwiers, and Anton Nijholt, *University of Twente*

Jorrit Kuipers and Arnd Brugman, *Green Dino Virtual Realities*

Using a movie set metaphor, this software framework increases a driving simulator's effectiveness by dynamically controlling traffic scenarios. It uses agent protocols to make the system extensible, maintainable, and easy to understand.

Imagine you're the center of the world, and everything around you is only reacting to your behavior. All the devices, animals, and people make their decisions based on what you're doing, but you don't know it or even notice it. Your world is that of Truman Burbank, from the 1998 movie *The Truman Show*.

With this idea in mind, we've taken the movie metaphor to implement a prototype simulation system where the user steps into Truman's shoes. The set of our "movie" is a driving simulator, and the user is learning to drive a car. During the driving lessons, users drive in a virtual world that lets them experience all kinds of traffic scenarios. The system generates the scenarios with the student as the focal point, and the other traffic entities respond to the student's behavior, without the student noticing.

To control the traffic scenarios and make them more effective, our prototype employs an agent-based framework. In this framework, each entity in the simulator is an actor agent playing a role. The prototype also includes a hierarchy of directors that directs the main action and the behind-the-scenes activity.

The advantage of the movie metaphor is that it helps separate scenario description from scenario playing. The agents can read their required information from a script and perform their actions based on that information. Because every agent has its own responsibilities, it's easier to introduce new elements, such as locations, actors, and roles, and the system is easier to debug. Using this framework lets us build software that's extensible, maintainable, and easy to understand.

Reinventing a driving simulator

Our prototype is based on Green Dino Virtual Realities' Dutch Driving Simulator. The original DDS supplied a virtual driving instructor that gave students negative and positive feedback and created ambient traffic, represented by software agents that drove freely through the environment. The DDS lesson structure consisted of the virtual driving instructor observing how the student driver performed in various traffic events.

The problem was the virtual driving instructor had no control over the flow of events; it only helped the student reach a place where a situation might occur. Because the agents were autonomous and hence unpredictable, the student simply drove around the simulation waiting for a useful event to coincidentally occur.

This system setup was problematic for two reasons:

- Driving-school instructors should be able to easily describe traffic scenarios using a scripting language or graphical editor.
- The system should be able to steer the agents in the environment so that scenarios will occur in the way the scenario scripts describe them.

The new DDS will be a scenario player. In this case, a scenario is a description of a traffic situation, Using scenarios to describe traffic situations has the advantage that situations occur in a more controlled way.

To provide scenario descriptions, other driving simulators use a specialized language[1,2] and have a centralized traffic director. In this case, the intelli-

1541-1672/06/$20.00 © 2006 IEEE
Published by the IEEE Computer Society

gence is in the environment, like an ant walking along a path defined by its environment. The environment controls the traffic entities using sensors. With this approach, creating new traffic entities is difficult. It requires modifications of the environment because the environment must know how to control the new entities. For example, where may these new entities move? Furthermore, the scenarios can occur only at the fixed positions in the environment where sensors are located.

Our approach uses text-based scenario descriptions similar to those that Syd Field[3] and R. Wade Allen and his colleagues[4] developed. However, their script-based languages define static locations in the environment; one can't define locations abstractly—for example, with language such as "at an intersection." These approaches use sensors and activators to send some actions to the traffic entities. The intelligence to lead the entities through the environment is contained in the environment. When a developer defines new dynamic elements with their own set of actions in the environment, the whole environment must be modified to support these actions. We developed a version of the DDS that puts the intelligence into the dynamic entities, letting developers define new entities into the environment without modifying it.

To help the driving school instructor implement scenario descriptions, we look to the movie world because it deals with similar problems. R. Michael Young also used the movie set metaphor to implement his system Mimesis,[5] but our system distributes the tasks among agents just like on a movie set.

On the set

A movie set is a clear, structured area where many people work together to create each movie scenario. Although only the actors are visible on screen, behind the scenes, many other people must work together. This metaphor clearly separates what's visible to the viewers and what occurs behind the scenes.

A scenario script[6,7] describes what should happen on the set by answering four questions:

- Where does the scenario occur?
- Who are playing the roles?
- When are they playing?
- What should they do?

This information is distributed over three parts:[4]

- information about the required location (the set)—for example, whether the scenario should occur at an intersection, a traffic circle, or somewhere else;
- a list of required actors (the cast, or what we call *role types*)—for example, actors portraying a bicyclist, pedestrians, and even traffic lights; and
- descriptions of the roles to be played—for example, the bicyclist forgetting to give the right of way.

This decomposition makes it easier for cast members and crew to collect the information they need. The whole scenario executes in a structured way. Because many participants help to create the movie, it's important that everyone knows what to do and when to do it.

No definition exists for when the scenarios should take place. The assistant director handles this. The time a scenario will take can differ from the movie script's chronological order, just as in real movies.

Our implementation of the movie set metaphor employs these crew and cast members:

- director,
- assistant director,
- casting director,
- location scout, and
- actors.

We use three types of actors. The star is always the student driver. The supporting cast, which might include a car coming from the right or the traffic light at an intersection, have specific tasks in a scenario. The extras are the actors on the set that simply fill the environment, such as ambient traffic on the highway's other side.

Figure 1 shows how the basic agent types communicate with one another.

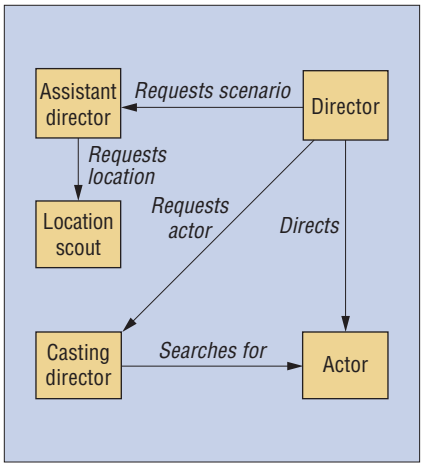


Figure 1. The basic agent types and how they communicate.

Running scenarios

A traffic scenario has the same structure as a movie scene. It contains information about the traffic location and the needed roles. It also explains what the actors should do—for example, the car driver should turn right and the traffic light should be red. We separate the role types and role descriptions because two different groups of software agents need these two kinds of information. The casting director agents use the role types to select the cast, and the actor agents use role descriptions to execute the roles.

We can describe the whole scenario setup in an object-oriented manner. The scenario is an object containing a location object, which describes the location properties. Every specific instance of a location is a subclass of the location object. In the DDS, this could include instances of intersections or roads.

The scenario object also contains one or more role types. Every role type is directly related to a group of actors. For example, the road-user role type is related to the pedestrian, bicyclist, and car driver groups of actors.

Finally, the scenario object contains the role description, which has subclasses for every kind of actor. This lets us define specific role types for every actor, defining the actions that only that actor can do. We distribute the role description over shots. Every shot is like a photograph, defining the state of every actor type. When actors know their desired state in every shot, they're responsible for improvising from their current state to the next state. The revised DDS can play more than one scenario during a driving lesson. The system stores the set of scenarios to be played in the scenario pool, which is much like a movie script. The most important difference between the DDS and a movie is that scenarios aren't directly related to each other in the DDS.[7] Rather, they execute in a series where every scenario is an episode independent of the other episodes.

Before a DDS scenario can start, the agents behind the scenes (the crew members) have much to do. The location scout agent always tries to find useful locations in the stu-

dent's virtual neighborhood. The locations it recognizes are road elements and intersections. A road might have special properties, such as access, exit, or parking lanes. An intersection might also have special properties, such as the number of roads connected to it or a specific control type (right-of-way rules or traffic lights). These properties are important because some scenarios require specific locations—for example, a T-junction with traffic lights (an intersection with sideways left and right but no opposite roads).

The assistant director agent continually monitors which scenario applies to the student's state. It makes its decision using the set of scenarios in the scenario pool, the available locations, and the student's driving skill, an extra DDS component that defines which scenarios can be played and might be useful. Because the location scout manages the set of available locations, the assistant director must request locations from it. If the location scout knows a matching location, it sends this location to the assistant director. When the director requests a useful scenario, the assistant director returns the scenario with a matching location.

When a scenario and a location are available, the casting director selects the cast. The casting director knows what kind of actors can execute each role. But sometimes it's desirable that an actor is in a certain state; for example, a car might need to approach an intersection from the right. So when the actor type matches, the casting director asks the actor whether it can execute the role description. This makes it easier to introduce new types of actors because an actor is the only one that must know whether it can play some role description at a certain moment.

Once the actors have accepted all the roles, the director can start the scenario by sending an "action" command to all the actors. Then, the director continually observes whether all the actors can fulfill their roles.

The actors get their role descriptions and improvise how to fulfill their role. For example, car driver agents can generate a route to fulfill their role. The director agent sends directives to the actors about how much time is left to reach a certain scenario. If necessary, the actor can reimprovise its plan to stay on schedule. If the actor knows it can't reach the next scenario in the given amount of time, it notifies the director. The director then stops the scenario and marks it as "failed" so that it can be played another time.

The director can't restart scenario execution directly because the student is continually moving through the environment. This differs from a movie set, where the same scenario can be replayed until it's recorded successfully. So, we can view the simulation scenarios as live action, where everything must be recorded successfully the first time.

Because the DDS contains ambient traffic, actors could pass a role to other actors that can play the role successfully. For example, a car driver must pass an intersection within a certain time window, but it might not be able to because it's stuck behind a slow vehicle. It could ask a car driver that's in front of the slow vehicle and that has no specific role (that is, one belonging to the ambient traffic) to execute the role. When actors exchange roles, they must notify the director so that it can maintain control over the scenario. We haven't yet implemented such role exchanges, but we believe they could increase the chance that a scenario plays successfully.

Because the Dutch Driving Simulator contains ambient traffic, actors could pass a role to other actors that can play the role successfully.

Directing the directors

The problem with having a single director and casting director is the single director and single casting director must have knowledge about every role type and every actor, respectively. So, if the developer introduces a new type of role or actor, he or she should modify the implementation of the existing director and casting director. When the number of roles and actors grows, the director and casting director implementations could become complex.

To deal with this problem, it is a better idea to define a director and a casting director for every role type. A specialized casting director can search for specific actors to play a given role. Consider a scenario requiring a road user at the right side of an intersection. Instead of one casting director checking all the actors approaching the intersection from the right, a specialized road user casting director can try to fulfill this request. Because the specialized casting director knows the subset of actors that are road users and can therefore potentially handle the role, it only needs to search this smaller set of actors.

Defining a director for every kind of role has many advantages. Specialized directors can give extra directives to the road user—for example, to increase speed. Because every director has its own set of directives to send, it's easier to introduce new kinds of roles and actors. When a developer introduces a new type of actor and its corresponding role, he should also introduce the corresponding director and casting director. For example, when a new role "traffic light" is introduced, the developer should also create the corresponding director, casting director, and actor (see figure 2).

This approach establishes a tree structure for the director agents, where every director except the master director has a parent director. The structure is based on role types. For example, car drivers, bicyclists, and pedestrians are three separate subroles of road users. So, the road user director is the parent director of the car driver director. Only the master director can start and stop a scenario; the other directors take their orders from it. Figure 3 shows a director hierarchy.

This structure lets us incrementally distribute the roles. When directing all the roles, the master director looks at each role type to determine the responsible subdirector. This approach is iteratively repeated by the subdirectors until the role type reaches its corresponding subdirector—for example, a road user role reaches the road user director and a car driver role reaches the car driver director.

Actors talking to one another

Another problem with our approach is that it's difficult for actors to communicate with one other. For example, if a scenario requires a pedestrian to pass a T-junction as the student approaches it, the pedestrian might need to ask the student agent how much time it has to reach the T-junction. But how will it know which actor is portraying the student? The student's role description defines its role name, so the pedestrian could search for the student by using that role name and asking every actor whether it's the

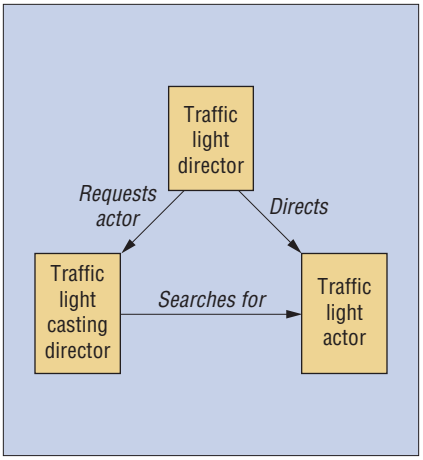


Figure 2. The agents to be created when a new role type "traffic light" is introduced with their relations.

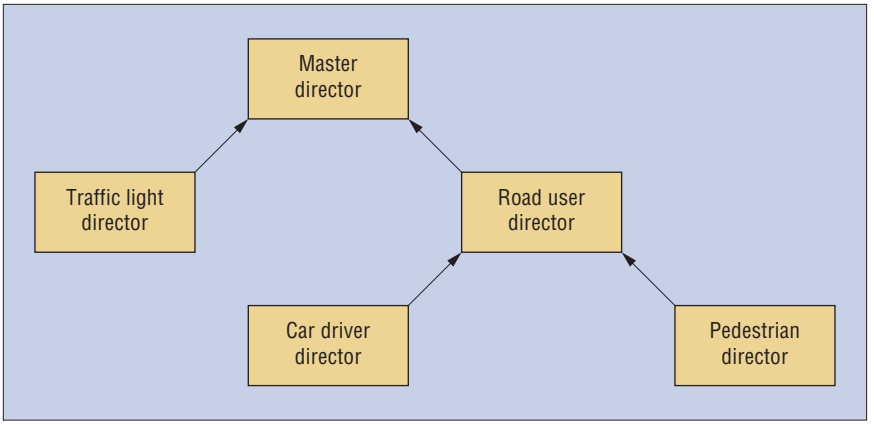


Figure 3. The hierarchical relations between the directors; for every director, the arrow points to its parent director.

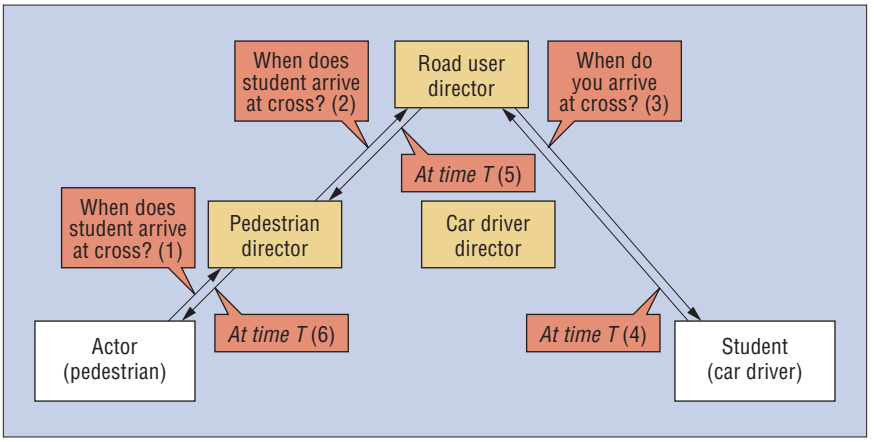


Figure 4. A pedestrian, Actor 1, can ask the student about its arrival time at the intersection. The actors don't communicate directly, but use the directors as a communication channel.

student. However, this method is inefficient and unintuitive.

To solve this problem, the actors communicate with all active actors (actors that have a role) through their director. After the casting director assigns the roles, all directors get a list of their and their subdirectors' active actors. For example, the car driver director gets a list of all active car drivers. At the same time, it sends this role allocation to its parent director, the road user director. So, the road user director knows every car driver actor that's assigned the road user role.

Now, if the pedestrian wants to know when the student (a car driver) will arrive at the T-junction, it asks its director for the information. This director sends this request to the road user director. The road user director can send the request directly to the student because it knows all the road users. The student will then send the answer to that director. Finally, the answer will be sent in reverse order back to the pedestrian. Figure 4 shows this structure.

Of course, the kind of information that an actor can request depends on the kind of information it can understand. For example, a traffic light controller doesn't know how to interpret the meaning of a car driver's clutch. However, all the actors can ask another actor when they need to be present at the next spot because they all understand the concept of time.

Extending and debugging the system

Our implementation of the movie set metaphor makes the whole DDS system extensible. For example, the location scout is the only agent that must interpret a scenario's location description. So, if a developer wants to add a new location type, he or she only needs to extend the location scout to recognize that type.

Owing to the system's clear structure and distribution of responsibilities, a developer can more easily add new elements. For example, when adding a new role type, he or she simply creates a new actor, casting director, and director. We don't need to modify existing directors because the new roles are automatically sent to the new directors.

Debugging the whole system is also easier. For example, if something goes wrong in scenario selection, the problem lies with the assistant director. If location recognition goes wrong, the problem is related to the location scout.

The movie set metaphor isn't restricted to driving simulators. For example, in a video game, you could implement every living element as an actor. The location scout could recognize the location for playing a game scenario. On the basis of this and the player's game experience, the assistant director could select a scenario. The director could distribute the roles among the actors, who would then behave according to their role. So, the game level and game play become configurable at runtime. A game could be different every time you play it.

This metaphor is also suitable for implementing other simulations, such as emergency rescue training. The location scout can search for certain locations. The assistant director can choose a suitable scenario for creating an emergency situation and make up the set, which might include a fire. Then, the director selects the required cast, such as

injured people, animals, or even part of the rescue team the student must work with. Finally, the director could examine whether the student and his team do their rescue tasks successfully. ⬒

References

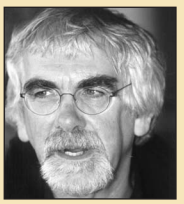
1. J. Cremer, J. Kearney, and Y. Papelis, "HCSM: A Framework for Behavior and Scenario Control in Virtual Environments," *ACM Trans. Modeling and Computer Simulation*, vol. 5, no. 3, 1995, pp. 242–267.
2. J. Cremer and J. Kearney, "Scenario Authoring for Virtual Environments," *Proc. Image VII Conf.*, 1994, pp. 141–149.
3. S. Field, *Screenplay: The Foundation of Screenwriting*, 3rd ed., Dell, 2003.
4. R.W. Allen et al., "Scenarios Produced by Procedural Methods for Driving Research, Assessment and Training Application," *Proc. Driving Simulation Conf.*, paper no. 621, 2003.
5. R. Young, "An Overview of the Mimesis Architecture: Integrating Intelligent Narrative Control into an Existing Gaming Environment," *Proc. AAAI Spring Symp. Artificial Intelligence and Interactive Entertainment*, AAAI Press, 2001, pp. 78–81.
6. P. Davidsson, "Multi Agent Based Simulation: Beyond Social Simulation," *Multi Agent Based Simulation*, LNCS 1979, Springer, 2000.
7. J. Kearney, "Scenario Languages for Driving Simulation," *Proc. Driving Simulation Conf.*, 1999, pp. 123–133.

The Authors

Ingo Wassink is a PhD student in the Human Media Interaction group of the University of Twente's Department of Computer Science. His main research interests are artificial intelligence, multiagent systems, computer vision, and computer visualization. He holds an MS in computer science from the University of Twente. Contact him at the Univ. of Twente, Postbus 217, 7500 AE Enschede, Netherlands; wassinki@ewi.utwente.nl.

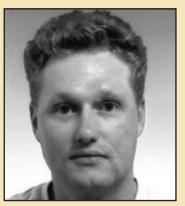
Betsy van Dijk is an assistant professor in the Human Media Interaction group of the University of Twente's Department of Computer Science. Her research is in human-computer interaction, particularly in personalization and evaluation methodologies for multiparty interaction. She received her PhD in teaching methodology in computer science from the University of Twente. Contact her at the Univ. of Twente, Postbus 217, 7500 AE Enschede, Netherlands; bvdijk@ewi.utwente.nl.

Job Zwiers is associate professor in the Human Media Interaction group of the University of Twente's Department of Computer Science. His interests include human-computer interaction, computer graphics, and multiagent systems. He holds a PhD in computer science from Eindhoven University. Contact him at the Univ. of Twente, Postbus 217, 7500 AE Enschede, Netherlands; zwiers@ewi.utwente.nl.



Anton Nijholt is a full professor and the chair of the Human Media Interaction group of the University of Twente's Department of Computer Science. His main research interests are multiparty and multimodal interaction, virtual environments, and social and intelligent (embodied) agents. He received his PhD in computer science from the Vrije Universiteit of Amsterdam. Contact him at the Univ. of Twente, Postbus 217, 7500 AE Enschede, Netherlands; anijholt@ewi.utwente.nl.

Jorrit Kuipers is a manager at Green Dino Virtual Realities and a part-time researcher at the Technical University of Delft. His main research interests are virtual reality and driving simulations. He received his MS in agriculture architecture and philosophy from the University of Agriculture, Wageningen. Contact him at Green Dino Virtual Realities, Agro Business Park 10, 6708 PW Wageningen, Netherlands; jorrit@greendino.nl.



Arnd Brugman is a software manager at Green Dino Virtual Realities, where he's responsible for the software and model development for projects such as the Dutch Driving Simulator. His main research interests are virtual reality, driving simulation and graphics, and artificial intelligence. He received his BS in computer science from Hogeschool Eindhoven. Contact him at Green Dino Virtual Realities, Agro Business Park 10, 6708 PW Wageningen, Netherlands; arnd@greendino.nl.