A Humanoid-Robotic Replica in USARSim for HRI Experiments

Kyle Carter and Matthias Scheutz and Paul Schermerhorn
Human-Robot Interaction Laboratory
Cognitive Science Program
Indiana University
Bloomington, IN 47406, USA
{kylcarte,mscheutz,pscherme}@indiana.edu

***Abstract*— An important set of open questions in human-robot interaction research, and to some extent cognitive science, is centered around the difference in interactions humans have with real versus simulated robots or agents. The goal of this research is to understand the effects of the agent's embodiment on human perception and cognition.**

In this paper, we present our work on providing computational tools to facilitate research in embodied situated cognition and human-robot interaction. Specifically, we introduce a simulation model of our humanoid robot CRAMER which we developed in the UNREAL game engine using the USARSim control interface. We provide details on its development and the implementation of the control interface that allows it to work seamlessly with our existing robot control architectures. We also discus potential applications of the simulation model as well as future plan to extend it.

I. INTRODUCTION

Simulated environments like USARSim [1] have become important tools for the development, testing and debugging of robot control architectures in a variety of areas, including single and multi-robot setups with and without human-robot interaction (HRI). In addition to rapid prototyping of control software, however, simulated robots can also serve an important role in psychological research: they can be used for the study of important psychological phenomena related to *embodiment* and *situatedness* of agents, which are of critical importance for human-computer and human-robot interaction. Specifically, sufficiently accurate simulations of real robots will allow us to study any possible differences in how humans interact with real versus virtual agents. These differences, then, will have significant implications for the design, testing, and deployment of robots and robotic architectures. For example, we have demonstrated that the expression of affect in a robot's voice can motivate people to perform better at a joint task when the robot is physically co-present in the same environment as opposed to just shown on a video screen [6], [7], [8]. Similarly, a warning message from a virtual character might be *less believable* than that from a physical robot [3]. Hence, one important implication for the design of robotic architectures in simulation is that HRI mechanisms that work well with simulated robots might not work well or work at all with physical robots and vice versa.

In this paper, we describe our work developing a simulated replica of our physical robot that can be used for HRI studies, specifically to explore the effects of physical embodiment. We will describe the details of the simulated robot, how it was developed, how it can be controlled, and how it can be used for future psychological experiments.

II. BACKGROUND

Multiple simulation packages are available that can be used for the development of simulated robotic replicas (including USARSim, Gazebo, ODE, etc.). Some of the packages already come with robot models and APIs for software control architectures (e.g., USARSim or Gazebo), while others provide only a core physics engine within which both simulated robot models as well as software interfaces will have to be developed. Our selection of USARSim was based on the fact that there is a fairly substantial user community with increasing support for robot models and environments.

USARSim is a collection of modifications made to *UNREAL Tournament* (a commercial "first-person shooter" game) for the purpose of allowing a robotics control architecture to interface the game engine. UNREAL Tournament is a physics-based simulation, which supports rigid-body physics and interactions. The USARSim modifications remove all game-related elements of UNREAL, leaving only the physics engine and some client software. USARSim also includes GameBots, freeware from a 3rd party developer, which provides mechanisms for communication over a TCP socket. This means of communication forms the basis for the creation and control of agents within the simulated environment at runtime. Because GameBots uses sockets for communication, developers can construct their own client software to connect to USARSim. It is through this client that one can place agents in (or remove agents from) the simulation environment. And the client also allows robotic control architectures to send commands to the simulated robots (e.g., to control a robot's wheels or actuators).

While we are using a variety of robots in our HRI studies, we are particularly interested in human-robot interactions with human-like robots. For this purpose, we employ our robot CRAMER (the "Cognitive, Reflective, Affective, Mobile, Expressive Robot"), which consists of a humanoid upper torso (manufactured by the now defunct company RoboMotio) mounted on a mobile Pioneer P3DX platform. CRAMER has two firewire cameras mounted in its eyes

and a series of eight microphones mounted in its torso. Moreover, it has movable eyes, eye brows and lips to produce facial expressions. The challenge for the simulated robot in USARSim was thus to replicate all of CRAMER's effector capabilities as closely as possible, including their timing and their degrees of freedom. Our plan was for the replica to be displayed on life-size 63in monitor so that the simulated and real CRAMER, placed side-by-side, would exhibit (close to) identical motions if controlled by the same architecture.

The development effort consisted of the integration of three primary tasks. First, we created a simulation model for CRAMER with the help of a number of 3rd party programs and tools. Second, we produced the classfile structure and configuration that UNREAL Tournament uses to recognize CRAMER as a placeable agent. Finally, we wrote a client for USARSim building upon the basic functionality already provided by USARSim and GameBots, together with additional control software for our ADE control environment, to allow us to connect our robotic architecture to the simulated CRAMER in exactly the same way it is connected to the physical robot. In the following, we will describe all three parts in more detail.

III. SIMULATION MODELS OF HUMANOIDS IN USARSIM

A fully functional simulacrum of a robot requires both a model of the physical shape and behavior of the robot as well as specifications of how some of the movable parts can be controlled. Creating the physical model in USARSim consists of "modeling the robot's parts", "adding texture to the surfaces" to make it look like the original, "organizing various configuration files" that UNREAL requires for the parts to function as a whole, and "configuring the robot's joints to function properly". Once the simulated robot is assembled within UNREAL, software can be developed to control it. Our lab's USARSim client software currently consists of three main components: the code that initializes communication with USARSim, the parser which interprets all of the incoming data, and a set of methods that implement the translation of the platform-independent action commands of our robotic control architecture into platform-specific commands of USARSim.

A. Model Creation

The possible methods for creating the model of the simulated robot are varied, and the particular combination of methods presented here is only one of many. As part of our commitment to open-source operating systems and software, our lab chose to work as much as possible with free and open-source tools available for Linux, which turned out to be more complicated and time-consuming than what model development would have been otherwise using proprietary Windows-based software. To construct the three-dimensional models of CRAMER's parts, we used the free modelling program *Blender*. The texturing was originally accomplished using the raster image editor *the GIMP*, but later we moved to the scalable vector graphics image editor *Inkscape*. Using

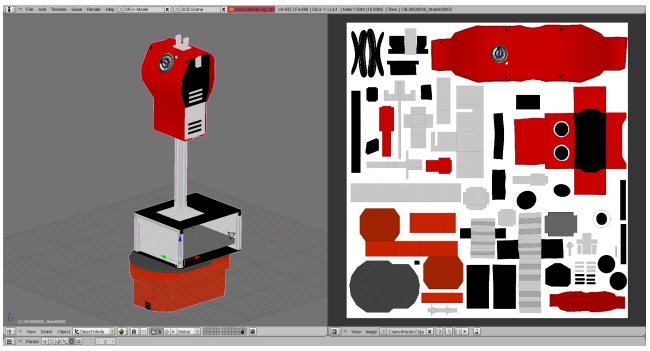

Fig. 1. Split shot of the model and texture in Blender, showing how the texture is mapped to the model.

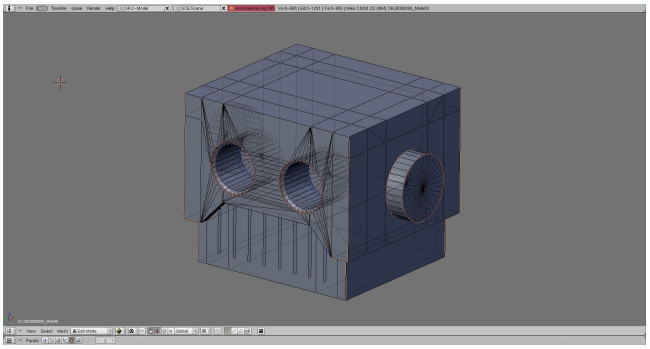

Fig. 2. Wireframe view of the CRAMER head model.

these programs (after a significant learning phase), a smooth workflow developed that proved quite efficient at producing robot models.

The modelling software, Blender, handled both the forming of the models, and their preparation for texturing, called "wrapping" (see Fig. 1 and Fig. 2). The wrapping process involves designating seams on the models in order to provide a smooth, relatively distortion-free means of wrapping a two-dimensional texture around a three-dimensional object. There do exist other open source modelling tools that perhaps could be used to create the 3D content for USARSim, but none seemed to be as mature and effective as Blender, and so their use was not considered further. Once the model

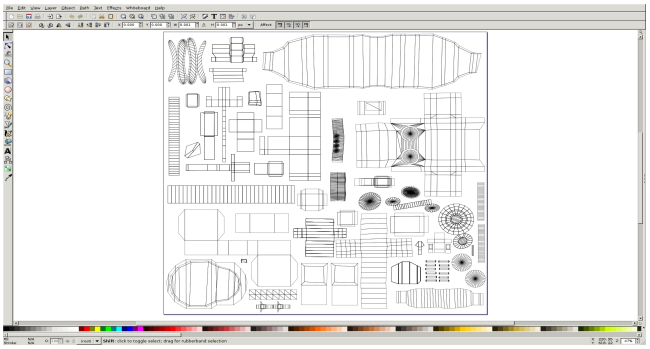

Fig. 3. The blank texture before it is filled in using Inkscape.

was prepared, the wrapping template was exported to an image editor, where it was colored and smoothened (see Fig. 3). One significant benefit that using Blender brought to this process was its ability to export the unwrapped texture directly to a file with which vector graphics software could work. Similarly to our work with Blender, there were other programs that could have been used to fill in the textures, such as Xara Xtreme LX, which is released under the Gnu Public License. Inkscape initially seemed to be more developed and robust than the alternatives, and so further exploration was deemed unnecessary. Both the model and the texture image were then exported into formats that the UNREAL Editor would accept (all content used by USARSim must be of in one of a few particular formats, according to type). In order for UNREAL to be able to use the robot models, it had to first be packaged by the UNREAL Editor. While we were able to use open-source software for physical modelling and texturing, only the proprietary UNREAL Editor can package up the collection of model, texture, and configuration files and class files, the latter of which define the robot as an entity and connect the pieces into controllable groups of joints (for details on this process, see the documentation in the USARSim manual). Configuration files give joints their limits, including range of motion, maximum speed, maximum torque, along with other parameters. With everything in place, the robot can now be initiated and manually controlled through GameBots through a simple telnet connection. For autonomous robot behavior, however, a robot control architecture is needed.

B. USARSim Client Software

In order to encapsulate USARSim's simulation capabilities for the purpose of using it with our robotic software environment ADE, we developed our own client software to provide functionality from a pre-established API that other robot interface components implement (when they interface physical robots). This functionality was achieved by wrapping the low-level joint command provided by USARSim into JAVA-based methods, such as `moveArm` or `moveHead`, that accept a number of arguments and forward the appropriate command together with the arguments to USARSim, which then performs the appropriate action.

As with all physical robots, however, the communication is two-way and sensory data needs to be received from the robot as well. In USARSim, the sensory feedback is continuously provided in special message packets, which need to be parsed accordingly and translated into a format that the robotic architecture can understand, another function performed by our USARSim client. Finally, the client is able to establish communication with USARSim and initialize the robot in the environment in right location in a specified configuration.

C. Control Implementation

The procedures of control are all built around the USARSim defined protocol. All commands are issued to GameBots over the established socket in raw line-based text format, designating which joint is to be moved as well as the magnitude and order of the movements. This order defines what type of command the message is, and can be a positional command, velocity command, or torque command. The order, therefore, determines the meaning of the magnitude, which is a number whose units are either radians, radians per second, or in a special unit used by UNREAL to measure torque, respective to the order. A significant challenge in implementing the control of the simulated robot is the inherent difference in methods of control between the simulated and real version. UNREAL Tournament accepts one of three orders of control for any joint, allowing the user to specify a joint's angle, velocity, or torque. However, in some physical robots, servo control allows for multiple orders to be issued simultaneously (e.g., specifying that a joint move to a certain angle at a particular velocity). Bridging this gap requires more sophisticated control than is natively provided by USARSim. The general solution to this problem is to have the server issue a velocity command, wait for a bit, and then send a command which sets the velocity to zero when the joint is at the correct angle. Two potential methods of ensuring correct timing for the stop command have been implemented to produce the highest fidelity between the physical and simulated robots' movements. While one method seems to be significantly more reliable than the other, there may be uses for which either is better suited.

The first method — "check to stop" — relies upon UNREAL Tournament's incoming information about the joints. Throughout the lifetime of a simulated robot, USARSim is constantly sending information about all of its joints—their respective positions, velocities, and torques. This method then waits for a return value from the parser for the appropriate joint that indicates that it is in the correct position, and so should be stopped from moving further. The method works reliably at lower velocities (0 to ~ 1.2 rad per sec) and on movements that involve longer movements ($>\sim 1$ to 1.5 rad). However, as the rotational velocity increases, this method's limitation begins to show. USARSim periodically updates its joint information every ~ 0.2 seconds. When the velocity is too high, or the angle displacement is too small, the number of updates received about the joint's position during the movement is reduced to zero, frequently causing the joint to overshoot its intended angle. Ultimately, this limitation prevents the "check to stop" method from being the preferred method.

The second method relies instead on dead reckoning. Given a joint's current angle, the desired angle, and a rotational velocity, one can simply compute the time necessary between start and stop commands using the formula $dt = v/dx$. During initial testing, this method was thought to be flawed by complications in the physical simulation entailed in UNREAL, but further development saw that these errors were in fact being produced by limitations enforced by the USARSim configuration of the robot's joints. In the configuration, certain parameters are set for each joint, including limits on range of motion, as well as limits on the joint's velocity. Because these limitations are enforced internally in UNREAL, it was not initially clear that they

were what was interfering with the calculations. If the client were to order a movement that was faster than allowed by the configured restrictions, the command would be sent without any indication that it was being executed incorrectly except for the updates from UNREAL, which include data about the joint's velocity. The client software would make the calculation based on its anticipated velocity, which would produce a significant error due to the discrepancy of velocity. This problem was bypassed by increasing the maximum velocity of the joint in question. It was found that if the joint was permitted by USARSim to move at the correct velocity, the accuracy of the calculation was restored, allowing this dead-reckoning method to be used exclusively.

It should be noted that in the process of simulating RoboMotio's Reddy robot, we strove for realism of the mechanisms. The facial expressions were a particularly significant point of interest to us, as the robot is intended for human interaction purposes. On the physical robot, the eyebrows consist of one degree-of-freedom (DOF) each, rotation in the plane parallel to the face. Recreating the eyes, likewise, was a relatively simple matter, as they consist of three DOF: individual left-right pan, and a single up-down tilt, linked to both eyes. A "dummy" object was used to link the tilt DOF to both eyes. This object was a massless object that was positioned inside of the head model, so as to be invisible during normal simulation. The mouth, however, presented a much more difficult problem. On the physical robot, the mouth consists of two "lips". Both lips were lengths of flexible rubber hose, bent into any given shape by two servos, one at each end. These four servos could then produce simple facial expressions, such as smiling, frowning, or a tilde-shaped "confused" look, by turning to particular positions, bending the rubber. The problem presented to the simulation is the difficulty of simulating flexible objects. We were also limited by the topology of the mouth, as each defined joint part in USARSim can have up to one parent part. This directly conflicts with the concept of the rubber hose mouth of the physical robot, as the shape of each lip directly depends upon not one, but two joints. To solve this, we broke each lip up into three visible components: Two end sections and one middle section. The four end sections could be rotated like normal joints, with the positions specified by the method "moveMouth(a, b, c, d)". The two middle sections actually each consist of one normal rotational joint and one prismatic joint, which has a linear up-down motion instead of rotational. The positions of these "hidden" joints could be calculated to provide the appearance of a seamless curve. These mechanisms led to the ability to simulate the robot's facial expressions through exactly the same interface of commands. For example, to frown, the robots both turn their eyebrows to tilt downward in the center of the face, and turn all four corners of the mouth upward, bending it into the appropriate shape. The only difference between physical and simulated expressions is that the physical mouth is made out of rubber, and the simulated mouth is made of extra rigid pieces which depend upon the hidden calculations to be put in the correct positions. The commands which are sent to the individual software components that control the physical and the simulated robot are exactly the same.

D. Calibration

Once the simulation was set up, the remaining task required to complete the control functions was to adjust the parameters in the simulated robot to match those on the physical robot, specifically to map the numbers used for velocity commands on the physical robot (using a PWM signal) to the numbers that USARSim uses (radians per second). The physical servos' necessary use of torque precludes a linear or simple calculation. Instead, measurements were taken on each joint, at varying speeds, to match the speeds of the physical and simulated robots, through visual similarity. These points, then, provided the data for a polynomial fitting function that can provide a smooth translation between the PWM signal and the corresponding radians per second. One problem with this strategy, of course, is that physical servos will degrade as they are used over time, and so the fitted function will become obsolete at some point in the future, and the function fitting will have to be repeated.[1]

E. Parsing Sensory and Joint Feedback

We implemented a simple parser to take the sensory output from UNREAL and translate it into a format that the robotic architecture can use. The parser runs in a separate thread from the rest of the client's operations, and primarily consists of a loop that decodes the incoming messages and attempts to ascertain which type of message it has received by checking for distinguishing tokens. For instance, a message that carries information from one of the robot's sensors will start with the token "SEN", while a message with information about the positional data of the robot's joints will start with "MISSTA". The information gleaned from these messages can be used for sensory processing, such as obstacle detection, or for motion control, as discussed in the previous section.

IV. THE USARSIM DIARC/ADE INTEGRATION

The USARSim model described above has been integrated into the *Agent Development Environment* (ADE), an infrastructure toolkit for constructing complex robotic architectures developed in our lab [5]. ADE allows developers to create modular components called *ADE servers* that can subsequently execute on any host with appropriate hardware resources. An *ADE registry* maintains information about all servers currently running in the infrastructure; whenever a new server starts, it checks in with the registry and provides information about its resource needs and the functionality it provides to the system. The registry is then able to provide a reference to that server when another server requests a server with that functionality. Some examples of other ADE servers available to architecture developers are:

- Goal Manager/Action Sequencer
- Speech Recognition

[1]Note that it is unclear how to best address this problem without a thorough model of motor degradation which is unlikely to be available to owners of physical robots.

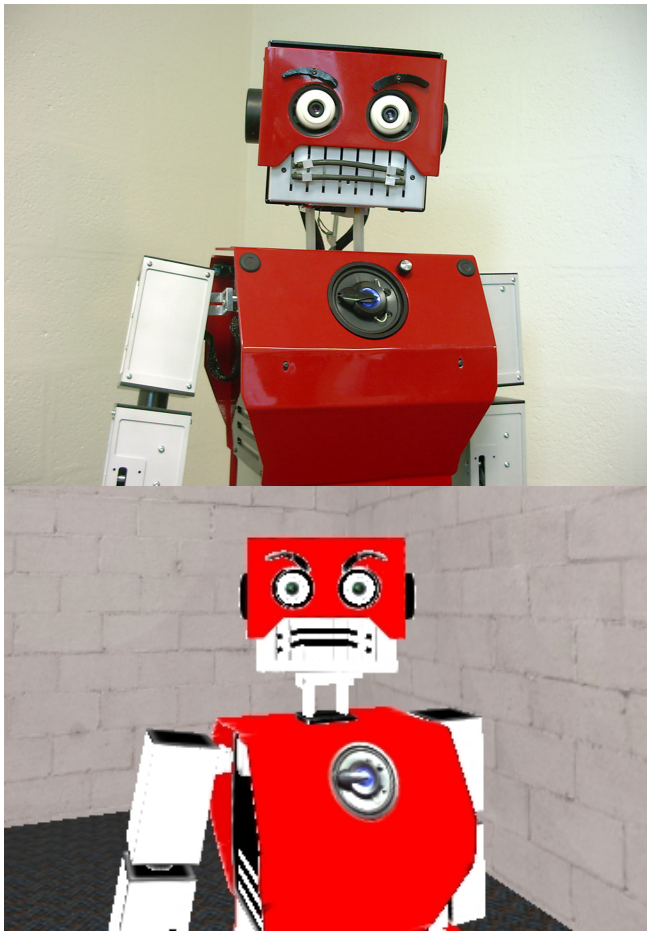

Fig. 4. Angry real CRAMER (Above), Angry virtual CRAMER (Below).

- Speech Production
- Natural Language Processing
- Planning
- Robot Base (e.g., Pioneer, Segway)

Of particular interest here is the *action manager*, ADE's goal management and action sequencing component, as it is the most frequent client of the USARSim server's services. The action manager uses the utility of goals along with information about goal deadlines to determine how resources should be allocated. This allows the system to pursue multiple goals simultaneously, so long as there are no resource conflicts (e.g., between a goal that requires the robot to remain stationary and a movement goal). When conflicts arise, the action manager gives precedence to the higher-priority goal. The action manager stores procedural knowledge in the form of action scripts that allow it to sequence multiple sub-actions together to accomplish a goal. For cases in which the action manager does not have a pre-defined script to achieve a goal, a planner component (based on the SapaReplan planner [4]) can construct scripts to achieve them.

The CRAMER server and USARSim server both implement the *HumanoidTorso* interface, which includes several methods for manipulating the arms, head, facial features, etc.

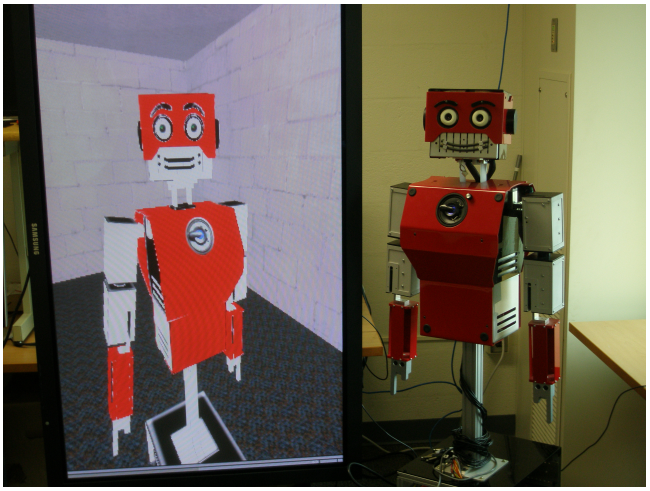

Fig. 5. Side-by-side view, as during the lab introduction described in the text.

Some of the methods are low-level, for example:

- `moveLeftArm/moveRightArm`
- `moveEyes`
- `moveHead`
- `moveEyeBrows`

whereas others are complex, higher level actions that build on the low-level interface, such as:

- `lookAt`
- `pointTo`
- `Frown`
- `Scowl`
- `Smile`.

Because the servers both implement the same interface, other ADE servers (e.g., the action manager server) need only request a reference to a *HumanoidTorso* server, and it can use whichever implementing instance (i.e., a server for the real or for the simulated robot) is returned. The USARSim server, in addition, implements the *PioneerServer* and *SICKLaserServer* interfaces, allowing access for ADE servers to the Pioneer and SICK laser range finder models included with USARSim.

V. REAL VERSUS VIRTUAL INTERACTIONS

The simulated version of the humanoid robot will allow us to explore important questions in human-robot interaction related to how people respond to simulated robots (see Section VI below). However, the validity of those experiments will depend, in part, on how closely the simulated robot models the behaviors of the real robot. We have tested the validity of the model in multiple contexts. Two of these scenarios are described below: a "dialogue" with a simulated and a real robot, and a "dance contest" in which the two robots employ the same algorithm to react (i.e., dance) to music. These scenarios allow us to evaluate the simulation by having the real and simulated robots side-by-side in the same environment; the simulated robot is displayed on a large

(63") plasma monitor, and hence can be displayed full-size, making it possible to eliminate the effect of size.

A. The Dialogue

In this scenario, the two robots perform the task of introducing the research done in the lab to visitors. The dialogue is fully scripted (very much in the way Disney animatorics work), so there is no geniune interaction with the people watching the robot. However, the two robots are programmed to respond to each other throughout their interactions. The real CRAMER begins the introduction by describing its own capabilities, explaining what its name stands for, and how it can be used in experiments in the lab. For example, because emotional expressions are very important for human-human interactions, familiar "emotion" expressions have been programmed for use by CRAMER; in the course of the dialogue, many of these emotional expressions are demonstrated. The robots behaviors are controlled by simple action sripts that are being executed by the action manager. A script for making the robot look "angry", for example, could look like this:

```
script lookAngry
moveLeftArm(5, 10, 90)
Scowl()
changeVoice(angry)
sayText("For example, I can look angry!")
```

Each of the script commands in this example invokes an action in a corresponding ADE server (the CRAMER or USARSim servers for the first two, the speech production server for the last two), producing a behavior in which the robot speaks in an angry voice while scowling and pointing angrily. `moveLeftArm` is a simple action in the robot servers, whereas `Scowl` is a compound action (also implemented in the robot servers) that builds on multiple lower-level simple actions to manipulate the robot's eyebrows, lips, and eyes.

While real CRAMER is introducing itself and the lab, virtual CRAMER "watches" it, nodding in (scripted) response to important points. When real CRAMER turns to and introduces virtual CRAMER, virtual CRAMER then takes over the introduction, demonstrating its own capabilities and elaborating on how having a virtual replica is useful for exploring the kinds of questions described in Section VI. The two robots then alternate back and forth for the remainder of the dialogue.

Note that the design of the servers plays an important role in the ease with which these behaviors can be scripted by the action manager. Because the USARSim server implements the same interface as the CRAMER server (as described in Section IV), the action manager can use the same scripts (such as `lookAngry` above) to evoke identical actions in the virtual agent. Hence, there is no need for the action manager to have any understanding of the difference between the two (or that there even is a difference).

B. The Dancing Robots

Another scenario used to demonstrate the functionality of the simulated version of CRAMER is the "dance contest." Once again, the robots are placed side-by-side (i.e., the large monitor is placed beside the read robot). The robot servers (CRAMER server and USARSim server) perform waveform analysis on the audio output of the selected song (Kraftwerk's *We Are the Robots*, naturally) to detect peaks. These are taken to approximate the beat of the music, and the servers generate random movements of various body parts to coincide with the peaks in the music. The effect is of two robots dancing to the music.

VI. DISCUSSION

The above examples of simple dialogues or synchronized behaviors between the real and the simulated robot are intended only as a proof of the functionality of the current interface, not as a demonstration of what its potential is for empirical studies of humans interacting with robots. Many empricial robot studies, particularly in the area of human-robot interaction, employ an experimental method that involves showing videos of real robots to subjects or having subjects interact with simulated robots, rather than having them interact directly with real robots. While there are certainly substantial benefits to the use of simulations in the development of complex robot architectures, it remains an open question whether experimental results obtained using simulations (or videos, images, etc.) are directly applicable to "best practices" in the design of architectures for human-robot interaction. In particular, the embodiment of the physically instantiated robot in the same physical space as the human subject is likely to have some effect on how the robot is perceived, how attention is allocated, where eye gaze focuses, etc.

For example, the simulated model of CRAMER will allow us to replicate an experiment that we conducted with CRAMER in a real environment, where the robot had to follow human eye gaze in real-time during a word learning task [2]. The interesting question then is whether the eye gaze patterns observed in humans interacting with the real robot will end up matching those to be observed in interacting with the simulated robot. If they match up, then we have learned something about eye gaze, namely that simulated robots do not necessarily have a different effect on attentional mechanisms in humans from real robots. If, however, they differ, then this effect will trigger a detailed investigation into the nature of attention and how it interacts with physical embodiment. And, of course, this is only one, immediate example of how the simulated version of CRAMER can be an invaluable research tool for psychology and HRI.

VII. CONCLUSION AND FUTURE WORK

In this paper, we presented a simulation model of our humanoid robot CRAMER in the UNREAL simulation engine using the USARSim interface. We demonstrated how a completely actuated robotic simulation model can be developed with open-source tools and how the model can

be connected to a robotic architecture in such a way that from the architecture's perspective there is no difference between controlling a simulated versus a real robot. We also briefly discussed how such a simulated replica of a physical robot that attempts to be as faithful as possible to both the visual appearance as well as the physical behavior of the robot can be a very useful tool for human-robot experiments. In particular, we believe that such a tool is necessary for the systematic exploration of the effects that the physical embodiment of a robot has on humans interacting with it, compared to possibly different effects of a two-dimensional version of the same robot on a video screen.

We plan to expand the current USARSim interface in the future, in particular, the way in which the ADE USARSim server manages the sockets and lines of communication. Currently, we can only initialize one robot inside the UNREAL environment and control it through our ADE system. The plan is to generalize the interface so that a single USARSim client can handle multiple connections. We are also planning on developing "dummy objects", which are initialized in the same way as the robot agent, but with limited actuating capabilities. A dummy object may be a box with a single hinge, for example, on which an agent may perform such actions as "open" or "close". While such interactive objects are already possible within UNREAL, there is currently no way for a remote client to manage and control them. Yet, we believe that objects with limited behavioral and actuating capabilities will be of great use (e.g., as props) in HRI studies.

VIII. ACKNOWLEDGMENTS

This work was in part funded by ONR MURI grant #N00014-07-1-1049 to second author.

REFERENCES

[1] Steven Balakirsky, Chris Scrapper, Stefano Carpin, and Michael Lewis. Usarsim: Providing a framework for multi-robot performance evaluation. In *Proceedings of PerMIS*, 2006.

[2] You-Wei Cheah, Matthias Scheutz, Chen Yu, Paul Schermerhorn, and Ikhyun Park. A multi-modal real-time interaction framework and platform for studying natural human-robot interactions. In *Proceedings of the International Conference on Multimodal Interfaces*, 2009 (Under Review).

[3] Robert Rose, Matthias Scheutz, and Paul Schermerhorn. Empirical investigations into the believability of robot affect. In *Proceedings of the AAAI Spring Symposium*. AAAI Press, 2008.

[4] Paul Schermerhorn, J. benton, Matthias Scheutz, Kartik Talamadupula, and Subbarao Kambhampati. Finding and exploiting goal opportunities in real-time during plan execution. In *Proceedings of the 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009.

[5] Matthias Scheutz. ADE - steps towards a distributed development and runtime environment for complex robotic agent architectures. *Applied Artificial Intelligence*, 20(4-5):275–304, 2006.

[6] Matthias Scheutz and Paul Schermerhorn. Affective goal and task selection for social robots. In Jordi Vallverd and David Casacuberta, editors, *The Handbook of Research on Synthetic Emotions and Sociable Robotics*. IGI Global, 2009.

[7] Matthias Scheutz, Paul Schermerhorn, James Kramer, and David Anderson. First steps toward natural human-like HRI. *Autonomous Robots*, 22(4):411–423, May 2007.

[8] Matthias Scheutz, Paul Schermerhorn, James Kramer, and Christopher Middendorff. The utility of affect expression in natural language interactions in joint human-robot tasks. In *Proceedings of the 1st ACM International Conference on Human-Robot Interaction*, pages 226–233, 2006.