Journal of Software Engineering for Robotics 5(2), September 2014, 13-24
ISSN: 2035-3928

ORCA - a physics-based robotics simulation environment that supports distributed systems development

E. Hourdakis[1] G. Chliveros[1] P. Trahanias[1,2]
[1] Institute of Computer Science, Foundation for Research and Technology, Greece
[2] Department of Computer Science, University of Crete, Greece

Abstract—Physics-based simulation is becoming an indispensable tool in robotics research, since it enables the evaluation of algorithms in silico, even when the actual robot is not available. Due to this, it is considered a standard practice, prior to any deployment on hardware platforms, and an important part of the development activities in large-scale projects. To cope with this role, simulators must address additional issues related to large-scale model development, including multi-modal processing, provisions for algorithmic integration, and fast, scalable graphics.
In the current paper we present ORCA, a versatile, physics-based robotics simulator that integrates a server and a simulation environment into one package. This type of architecture makes the sharing of resources a seamless task, promotes code re-use, and facilitates the integration of individual software modules. To demonstrate the software's applicability, we discuss three different use-cases, in which ORCA was used at the heart of their development efforts.

Index Terms—Physics simulation, robotics, sensor modeling, graphics, distributed processing.

1 INTRODUCTION

DUE to the upsurge in processing power, physics engines and CAD modeling software, robotic simulators are increasingly being used as a cost-free alternative to actual robotic platforms [1], [2]. A key reason for their popularity is that they allow testing and algorithmic evaluation upon several robots, which can be modeled at relatively high accuracy. Moreover, simulator packages offer additional benefits such as *(i) seamless sharing of platform configuration across organizations*, *(ii) altering the morphological characteristics of robots with no risk/cost* and *(iii) re-configurable environments*. Therefore, simulation is nowadays considered as a standard milestone in the workflow of activities that take place during a robotics project [3], and is used as the main development tool, prior to evaluation on actual hardware platforms.

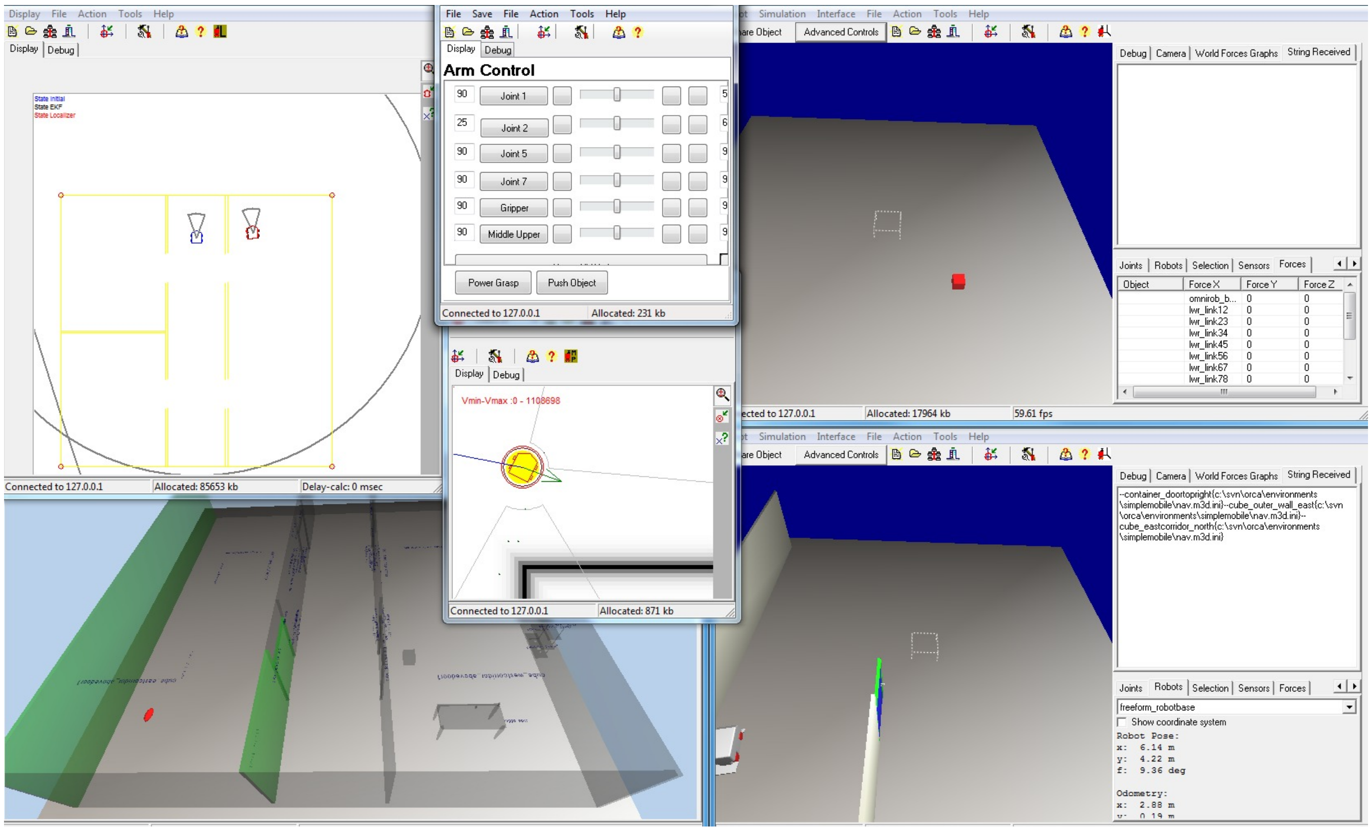
Due to above benefits, robotics simulators are gradually starting to claim an increased share of the development activities in robotics, which can involve large-scale and integrated systems. As such, they must consider issues related to the distributed nature of software workflow, for example the ability to facilitate algorithmic integration or process several complex environments/robotics structures.

In the current paper, we present ORCA (*Open Robot Control Architecture*) (Fig. 1), a physics-based robotics simulator, that integrates a *sever* and *simulation environment* into one software package. As a result of this architecture, a developer can decentralize the processing requirements of an environment into different simulation instances, and use the server to communicate information across instances. Further to that, ORCA supports a large number of modeling formats, a state of the art physics engine and features a great number of robotics related add-ons, such as navigation modules, human-robot interaction and dialogue interfaces [4]. All these render it an ideal solution for development purposes in robotics research, which is evident from the fact that it has been used in the development activities of several large-scale robotics projects (including First-MM [5], INDIGO [6], Xenios [7] and GNOSYS [8]).

The rest of the paper is organized as follows: In section 2 we discuss relevant works in the field of robotics simulation and place ORCA within the literature. Section 3 describes

Regular paper – *Manuscript received November 10, 2013; revised September 19, 2014.*

- *This work was partially supported by the European Commission (Contract No.FP7-248258, First-MM project).*



Object	Force X	Force Y	Force Z
	omnirob_b...	0	0
	lwr_link12	0	0
	lwr_link23	0	0
	lwr_link34	0	0
	lwr_link45	0	0
	lwr_link56	0	0
	lwr_link67	0	0
	lwr_link78	0	0

```
--container_doortopright{c:\svn\orca\environments
\simplemobile\nav.m3d.ini}--cube_outer_wall_east{c:\svn
\orca\environments\simplemobile\nav.m3d.ini}--
cube_eastcorridor_north{c:\svn\orca\environments
\simplemobile\nav.m3d.ini}
```

freeform_robotbase

```
Robot Pose:
x:  6.14 m
y:  4.22 m
f:  9.36 deg

Odometry:
x:  2.88 m
y:  0.19 m
```

Fig. 1. The interface of the ORCA software package, showing two simulator instances (right), server rendering a shared environment (bottom left) and add-on modules (top left).

the architecture of ORCA, focusing on its communication components and the simulator's environment, including its support for geometries, 3D meshes, kinematic constraints, and sensor models. In section 4 we demonstrate the benefits from ORCA's architecture, by setting up a common robotic scenario in which the software distributes the processing needs of an environment into different simulation instances. Section 5 presents popular use cases and add-ons that have already been developed during collaborative projects. Finally we conclude the paper, by discussing future implementation plans for the package (section 6). The current version of ORCA is freely available at http://www.ics.forth.gr/cvrl/index_main.php?l=e&c=17.

2 Related Work

Due to their increasing popularity, various simulation packages have become available in the robotics community, including Gazebo [9], MORSE [10], Webots [11], V-REP [12] and SimRobot [13]. When selecting simulation software, one usually evaluates several important factors, including its physics processing capabilities, available robotic platforms, sensor models and packages that it can support. In the current section, we focus on two of these issues: (i) physics simulation and (ii) algorithmic integration, while we discuss robot, sensor and object support in section 3.

Physics simulation: The choice of a physics engine is crucial for a robotics package, because of the complex aspects (including kinematic chains, constraints, collisions etc) of the environment that must be simulated. Amongst the aforementioned simulators, SimRobot and Webots use the Open Dynamics Engine (ODE) [14] to handle physics processing. MORSE builds upon Blenders python API [15], which in turn uses the Bullet library [16] for its physics implementation. Finally Gazebo [9], is an open source simulator that supports ODE and more recently Bullet.

In contrast, ORCA (Fig. 1) uses the Newton Dynamics Engine (NDE) [17], a powerful engine that has recently become available as open source. One of the main benefits of NDE is that it allows writing specific behaviors on how physical entities interact with each other. Consequently, one can define material properties for each object, and thus explicitly control simulation parameters such as collision, friction and elasticity.

Regarding the quality of the simulation, the choice of an engine is a matter of context. Several works have used various metrics [18] to evaluate the properties that affect the quality of physics processing [19], [20], [21], [22]. Amongst the most important include friction models, model scalability

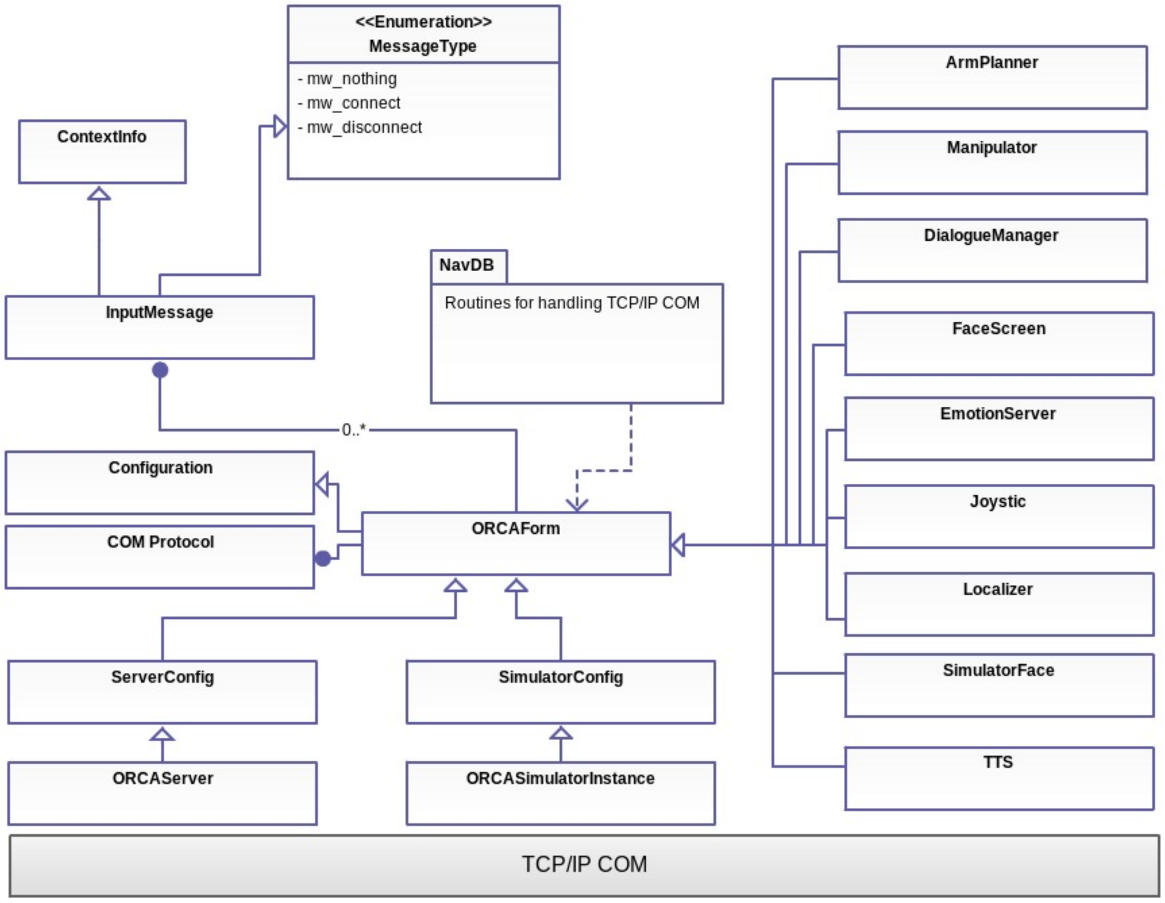


Fig. 2. UML diagram showing the modules in the ORCA package, along with their interdependencies

and energy preservation. In [20], the authors use the Physics Abstraction Layer to carry out an evaluation of 7 commercial and open-source physics engines, including ODE and NDE. They present the two engines as having similar capacities, with NDE being more accurate when modeling friction and rolling contact events. In [23] the authors present a more thorough evaluation between the two engines, in which they undergo a series of tests including bounce, stability of constraints, energy preservation and gyroscopic force modeling. Test results indicate that NDE outperforms ODE in most cases.

Algorithmic integration: To facilitate integration in software development, works have suggested various design principles that promote leverage. In [24] the authors identify several methods to facilitate integration, and note code re-use and collaboration as the most important among them. Others suggest client-server architectures [25], software modules for reuse [26], [27] and programming environments [28], [29] to confront these issues.

In robotics one common solution to these problems is to employ middleware component which can interface with many robots through one common system. In this context several works have become available, and are currently being used by robotics simulators, including ROS [30], YARP [31], Miri [32], Orocos [33], OpenRTM [34] and Player [35]. These type of components provide abstraction at the level of robotic platform, and a way to integrate the individual algorithms into one common system. A key difference of ORCA is that it integrates a middleware component along with the server and simulator, using compatible software data structures that can communicate with each other. As a result it provides the ability to interface with simulated and actual robots easily.

In the following sections we discuss ORCA's architecture and describe in detail how the software package can promote these concepts.

3 Architecture of ORCA

ORCA is implemented using a distributed architecture [36], having a *simulator*, a *middleware component*, and a *server* all integrated into one package. Communication across software modalitites is accomplished through the *ORCA communications server*, using a TCP/IP based protocol (*ORCA's communications protocol*) (Fig. 2). This setup allows the software to be easily expanded: new modules can be embedded, simply by augmenting the communication protocol of the server, and adding client instances that consume/produce the corresponding messages. Moreover, it promotes code re-use, since addons can be easily reused, through the server object, simply by enabling their corresponding messages in the server.

3.1 Communications server

The server is responsible for regulating the TCP/IP bandwidth amongst the different software components (Fig. 3). At runtime, each ORCA component registers with its protocol, and can then publish information throughout the network. This prevents from creating bottlenecks in large-scale applications, by enabling clients to seek services only when required [36]. To support a distributed infrastructure, the communication protocol provides timing stamps for each message that is communicated, and access to several functions of the software, including control, sensory and user interface messages.

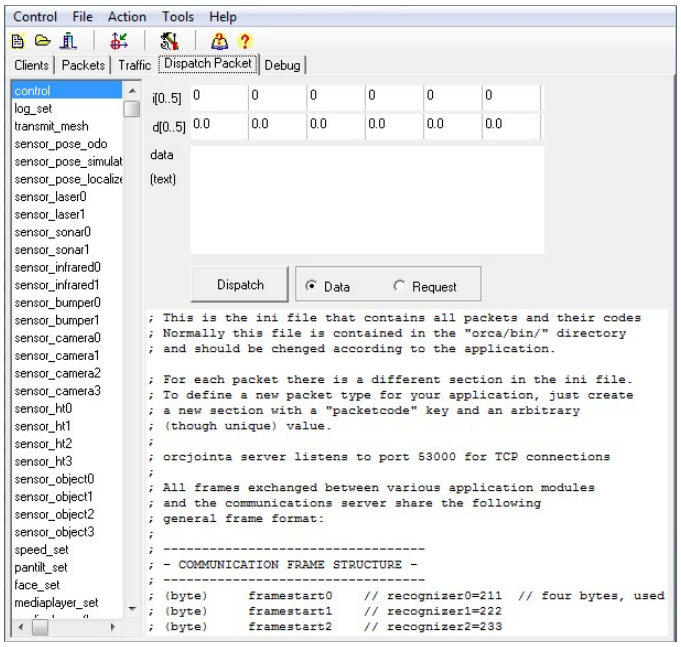
ORCA's server supports both synchronous and asynchronous message queues for real-time robot control. When it receives package from a specific connection, it reads the contents of the data field and expects to find a string and two lists of integers. These lists correspond to the codes of the packets that are produced and consumed respectively by the module at the other side of the connection. For each listed packet code, the server accordingly updates its corresponding "producers" and "consumers" lists. When a module disconnects from the server, the latter erases all references to the corresponding connection from the "consumers" and the "producers" lists of all packages and terminates the connection thread.

Packages are broadcast in a designated listening port either on demand, by publishing a message when available, or in a timely manner, where a package is posted on regular intervals. Client modules produce packages at instances, as described below:

- As a result of a specific event, e.g. a module attached to a motion sensor produces a package when motion is detected.
- Periodically, e.g. a module attached to a laser scanner produces a new frame at regular intervals.
- As a result of incoming data, e.g. a text-generator module produces annotated text, each time new text is created.
- When a request is received, e.g. a server, attached to a high-resolution camera, that produces frames containing camera images, only after an explicit request is received.

All pieces of information that need to be exchanged are assigned a different code (packet code). Packets and their codes are defined according to the application, and are stored in a configuration file. The communication server reads the configuration file and for each different packet code, the server creates two lists:

- the list of connections (modules) that will produce this particular packet (producers), and
- the list of connections (modules) that will consume this particular packet (consumers).



control
log_set
transmit_mesh
sensor_pose_odo
sensor_pose_simulat
sensor_pose_localize
sensor_laser0
sensor_laser1
sensor_sonar0
sensor_sonar1
sensor_infrared0
sensor_infrared1
sensor_bumper0
sensor_bumper1
sensor_camera0
sensor_camera1
sensor_camera2
sensor_camera3
sensor_ht0
sensor_ht1
sensor_ht2
sensor_ht3
sensor_object0
sensor_object1
sensor_object2
sensor_object3
speed_set
pantilt_set
face_set
mediaplayer_set

i[0..5]	0	0	0	0	0	0
d[0..5]	0.0	0.0	0.0	0.0	0.0	0.0

data (text)

Dispatch (● Data ○ Request)

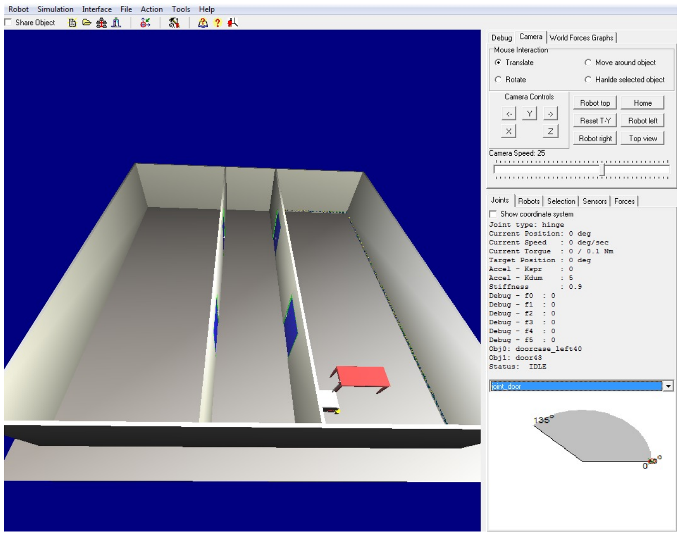
```
; This is the ini file that contains all packets and their codes
; Normally this file is contained in the "orca/bin/" directory
; and should be chenged according to the application.

; For each packet there is a different section in the ini file.
; To define a new packet type for your application, just create
; a new section with a "packetcode" key and an arbitrary
; (though unique) value.
;
; orcjointa server listens to port 53000 for TCP connections
;
; All frames exchanged between various application modules
; and the communications server share the following
; general frame format:
;
; ----------------------------------
; - COMMUNICATION FRAME STRUCTURE -
; ----------------------------------
; (byte)     framestart0    // recognizer0=211  // four bytes, used
; (byte)     framestart1    // recognizer1=222
; (byte)     framestart2    // recognizer2=233
```

Fig. 3. The interface of the ORCA server, which provides the ability to edit message data, dispatch individual messages and extent the communication protocol to fit the development purposes.

The server object also provides an interface to ORCA's communication protocol, including message monitoring, package dispatching and debug information pertaining to errors in the communication. In addition, it allows editing all required information regarding a certain package, including the applications that have registered to consume it, whether traffic is enabled for this package and monitoring of its activity.

Currently, ORCA supports more than 100 TCP/IP type packet messages that pertain to information about control commands, sensor information, physics related data, HRI interface commands and object handling messages. Moreover, to support platform independence [36], ORCA provides templates for client redistributables for Java, ANSI C++ and C. Consequently, through the server communication protocol, one can run instances of a simulation on any machine, including Windows and Linux, and use the server to handle their interaction.

More importantly, the communication protocol can be easily extended using a text-based structure. The server object scans a set of predefined directories upon initialization and registers new messages as required. The importance of this setup will become more apparent in section 4, where we demonstrate how ORCA's protocol can be used to share information processed by one instance of the physics simulation to other instances, thus allowing it to tackle the computational needs of a simulation, despite them being intensive.



```
Joint type: hinge
Current Position: 0 deg
Current Speed    : 0 deg/sec
Current Torque   : 0 / 0.1 Nm
Target Position : 0 deg
Accel - Kspr     : 0
Accel - Kdum     : 5
Stiffness        : 0.9
Debug - f0  : 0
Debug - f1  : 0
Debug - f2  : 0
Debug - f3  : 0
Debug - f4  : 0
Debug - f5  : 0
Obj0: doorcase_left40
Obj1: door43
Status:  IDLE
```

Fig. 4. The interface of the ORCA simulator provides capabilities for editing a simulation environment and inspection of object properties.

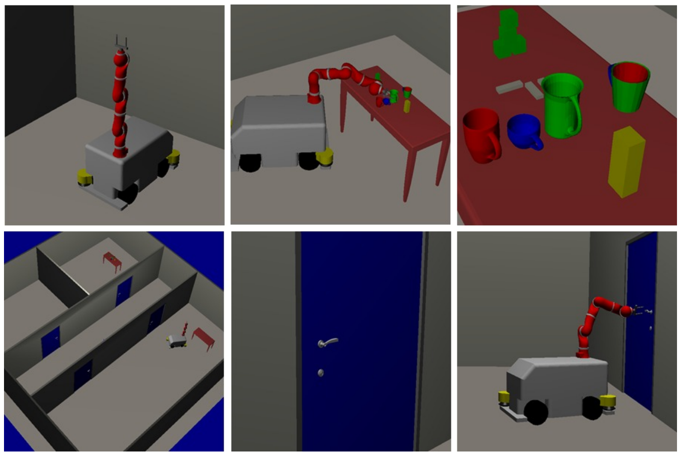


Fig. 5. The powerful modeling abilities provided by the ORCA simulator allow developers to design and simulate inherently complex environments.

3.2 Simulator

In addition to the integrated server, ORCA also includes a physics based simulator, that is able to process complex robotic structures occupying large environments. One of the main strengths of the simulator is its interface environment, which handles object properties with minimal code. The GUI (Fig. 4) allows manipulating several world parameters, including properties of objects and geometries, direct application of forces, camera movement, predefined views, environment loading, and configuration settings.

In addition, the simulator interface reports on all the physical interactions among objects, including constraint properties, robot poses, odometry and control information, forces applied to each object, and provides logging activities between the server and the client instances.

To model the interactions between different objects, the simulator integrates a physic engine into the graphics environment. Most commercial robotic packages are based on the ODE physics library [14], an SDK that can handle rigid body dynamics of three-dimensional multi body systems. Other alternatives include the AGEIA physics [37], a commercial engine that is implemented on a hardware chipset and OpenTissue [38], a physics library that can handle deformable objects. Finally, due to its GPU support, Bullet [16], a rigid body dynamics library which uses ODE's solver, is increasingly starting to gain amongst developers. ORCA employs the Newton-Dynamics engine, an up-to-previously commercial engine that recently released an open source SDK [17]. One of the main benefits of Newton is that it allows writing specific behaviors on how physical entities interact with each other.

In addition, ORCA includes several features which enable developers to generate new world scenarios seamlessly, without requiring any computing skills (Fig. 5). Rendering, handled by OpenGL, and physics processing, by Newton Dynamics, are coupled together through a tree-based hierarchy, called environment tree, which can be setup using configuration files. Currently, ORCA's environment tree supports the insertion of (i) objects, (ii) kinematic constraints, (iii) sensor models, (iv) robot models and (v) interaction behaviors. These are outlined in the following sections.

3.2.1 *Object definition*

Upon initialization the simulator loads an environment file, which contains the definitions and physics properties of each object. Several different types of objects are supported, including (i) primitive geometries (cube, polygon, capsule and triangles), (ii) containers, that bundle entities together, as well as (iii) freeform objects, which allow importing meshes from CAD software directly into a simulated world (*ORCA supports the .3ds, .stl and .obj formats, used by several modeling packages, including Blender*). Through the Newton Dynamics physics engine, ORCA provides the largest number of primitive objects than any other engine. In addition, it supports the definition of extrude objects, i.e. abstract geometries defined by a series of 3D point edges, as well as various light sources. Mesh and boundary information are generated automatically for each object that is inserted in the simulation environment, providing the option to select what type of bounding boxes will be generated from each object entered into the world. Additional properties include the initial position of an object, orientation vectors, object materials, whether it will be subject to gravitational forces, its mass and density.

3.2.2 *Kinematic constraints*

ORCA also supports the definition of various different types of constraints, which can be used to limit the movement of

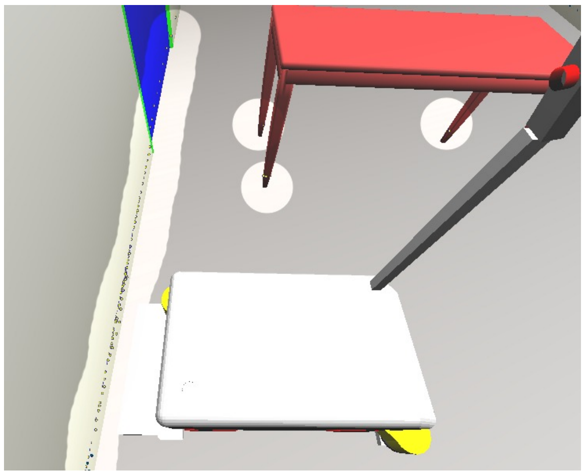


Fig. 6. Graphic output of the sonar sensor model provided by the simulator.

kinematic structures in one or more axes. These are also specified upon initialization, using the environment tree hierarchy, and can be used to restrict the motion of a body in relation to another. Currently various different types of constraints are supported, including ball, hinge and socket joints. ORCA exports all the properties supported by the physics engine, type and stiffness of the joint, pivot points and transformation axis, as well as acceleration bounds.

3.2.3 Sensor models

Most hardware robotic structures have a large number of mounted sensors, which must be modeled accurately by a simulator that is intended for research based purposes. ORCA has a great variety of sensor models available, which can be inserted directly into its environment tree.

Currently the software supports sensor objects, used to monitor the forces that different geometries exert upon each other, bumping sensors, responsible for detecting the collision among objects, as well as virtual cameras, that simulate the 2D projection of a scene using RGB and depth data (Fig. 6). The simulator also supports the definition of extrinsic and intrinsic camera parameters, as well as specific camera models including the Kinect, PointGrey and Logitech Quickcam Pro 5000 camera.

Each sensor model can be easily inserted to a simulation, by defining a set of initial parameters. For cameras, available properties include the image width/height and pixel per unit, models for distortion and noise, as well as intrinsic and extrinsic parameters. To facilitate the realistic simulation we have also incorporated appropriate sampling intervals for each sensor, which can be specified upon initialization. In addition, ORCA supports range object sensors, such as laser and sonar. Available parameters for these objects include the number of measurements sampled at each time interval, the minimum and maximum radius of the range sensors, as well as noise models relative to the distance of the measurement.

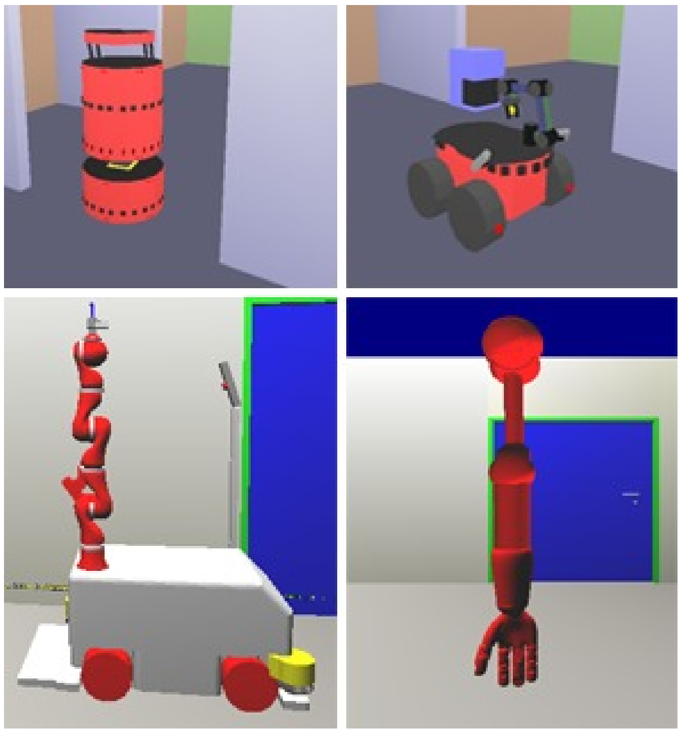


Fig. 7. Robot models included in the current release of ORCA's software package: iRobot3ADm (top left), Pioneer 3-AT (top right), Kuka LWR arm (bottom left) and custom 27-DoF manipulator (bottom right).

3.2.4 Robot models

Using the aforementioned three components, a developer can re-construct any type of robotic platform. ORCA also comes with a number of pre-defined structures for some popular robots (Fig. 7), including the Pioneer 3-AT platform, the iRobot3ADm, the Kuka LWR and a generic hand and arm model with 27-DoFs.

In addition, we are working towards integrating additional robotic models, which will made available upon ORCA's next release.

3.2.5 Interaction behaviors

An extremely useful feature of ORCA, is that it defines explicitly how objects and physical entities will interact with each other. This is supported through Newton's ability to define materials and how these interact with each other. Properties of the interaction include the use of gravity, collision mode, convex collision tolerance, net force limit, linear damping, translation and rotational impulse velocity. Moreover, each behavior can be set to run based on different collision primitives, be it mesh, or primitive bounding boxes, and can assign different mass, and density values to interacting entities. Finally, torque and collision estimation can be performed locally for each behavior, through the implementation of appropriate callback functions.

This feature gives complete control on the physical representation of objects in the world. One can use it to describe how different materials will interact together, and as a result define explicitly cross-object interactions, instead of describing a set of simple physics properties, as it is the case with other simulators. Additionally, one can turn off and on the physics processing of an object on demand, by freeing its assigned behaviors at any time, thus improving the speed of simulation by eliminating redundant entities. In the following section we demonstrate an example in which we make use of this property, along with the simulator's server, to segment the physics processing of a simulation environment into different workstations.

4 Distributed processing

The benefits from using ORCA become apparent when tackling computational intensive environments. The server works together with the simulation instances in order to distribute the computational load, by decentralizing the processing of a complex environment.

To provide this feature, ORCA has been designed to distribute the processing requirements of a simulation, through its server object, to various client instances. This architecture is based on the following principle: A complex robotic structure can be processed locally by a simulator instance, have its feature and state vectors computed, and make them available through the server object. The latter is then responsible to communicate those, on demand, to whichever simulation instance requests them. Consequently, it can tackle the computational costs that are inherent in complex robotic structures and large environments, by sharing and managing resources through its server object. To facilitate this feature, one simply has to enable the ”shared” property whilst defining an environment. Then the server object will transmit those properties, through appropriate messages, to the client instances that register to the messages. The functionality is embodied through three different properties in the simulator's environment: (i) robot sharing, (ii) environment sharing, (iii) object sharing. These are outlined below.

4.1 Robot sharing

A main characteristic of a simulated robot is that it utilizes resources that are modular, and can be described by an I/O convention. Consequently, in most cases, rendering a simulated robot is a computationally intensive task that outputs only a small set of parameters.

Consider for example the case where a “high degrees of freedom” manipulator has to be processed within a large environment. At any given time, the simulator has the duty to process its kinematic chain, integrate the appropriate equations, and update the position of the joints of the hand to simulate it. Even though this process contains a large number of computations, it is only used to output a small amount of information, such as the end-point location of the hand. Consequently, when interacting with such a robot, it is usually important to have only this piece of information available. Due to its design, ORCA can facilitate processing of such robotic structures locally, and distribute outputs from the simulation on demand, only when required, using predefined messages in its server object (Fig. 9). To setup such simulation paradigm, one has simply to augment ORCA's server communication protocol, with the additional messages that will be produced by the simulation instance, and the data structures each message will hold. New client instances will be able to consume those messages (e.g. pertraining to the location of the manipulator) whenever they become available to the server.

4.2 Environment sharing

Another important feature of ORCA is its ability to simulate large environments, by activating or deactivating processing in those segments that are not subject to change. Such feature is becoming increasingly important across research projects, and finds applications in a large number of fields including outdoor robotics, search and rescue simulations, swarm robots, etc.

In large environments (e.g. buildings with corridors etc), the robot is located within a small space, while other parts of the environment must be available only on demand (for example, if they contain an active robot within them). Upon initialization of an experiment, the server loads a virtual world file which contains the definitions of all objects that will be processed by simulation instances. These are created as physical entities and entered into the world without processing any of their properties (Fig. 8).

ORCA includes appropriate messages for this purpose. As soon as a new simulation instance is loaded into the new world, it can use these messages to communicate its location to the server. The server, which must be started with the environment sharing option enabled, will respond to the request of the instance by transmitting the environment and object properties that are in the vicinity of the robotic client. The user will only have to include options regarding the radius in which the environment must be transmitted, and properties regarding the intervals of transmission. As a result, ORCA is able to handle any type of environment, by processing only those parts that are subject to change (e.g. as the result of having an active robot within them).

4.3 Object sharing

Further to the above, an important feature that was implemented to augment environment sharing, is object sharing. ORCA utilizes the flexibility of the NDE engine to limit the processing done to single objects. More specifically, by

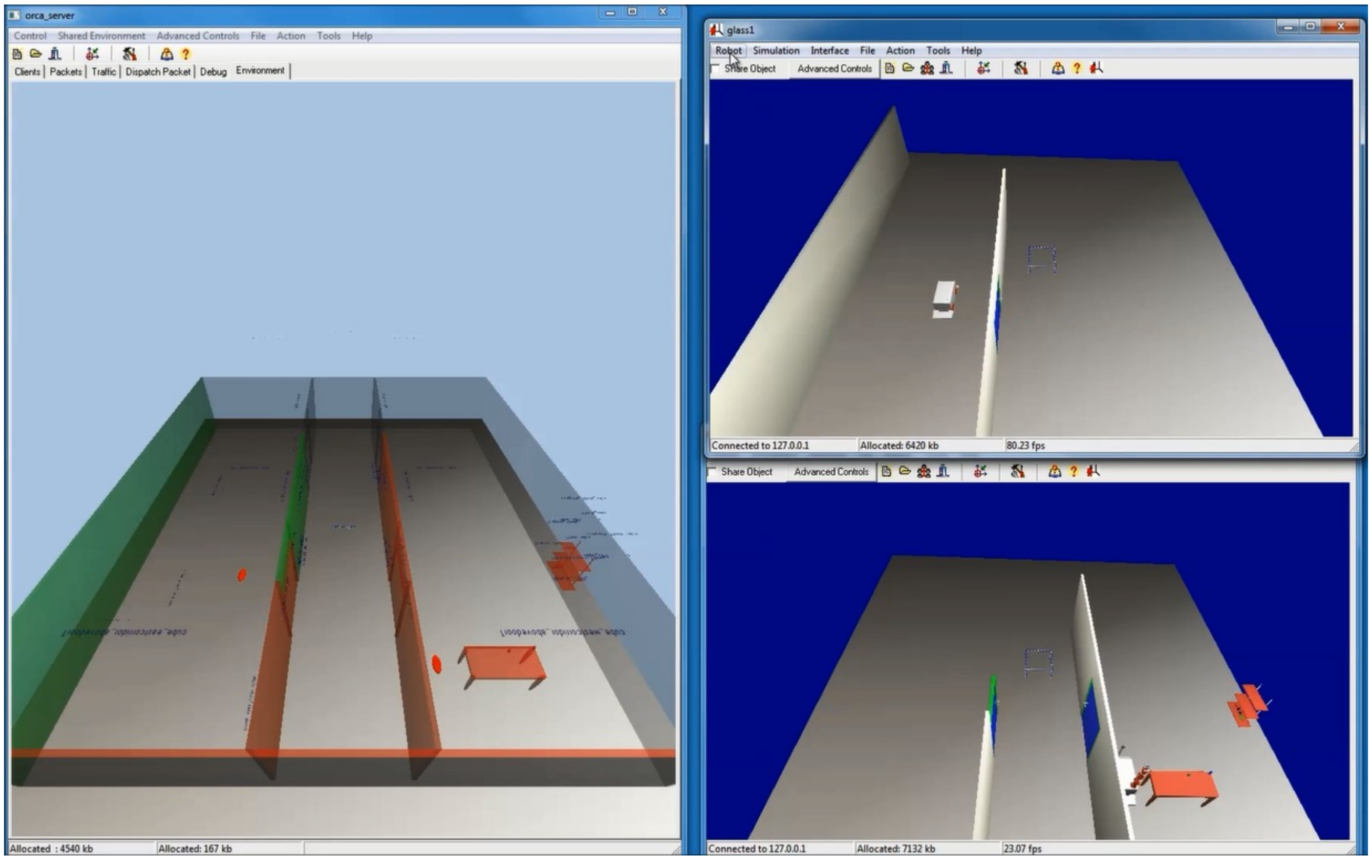

orca_server
Control Shared Environment Advanced Controls File Action Tools Help
Clients Packets Traffic Dispatch Packet Debug Environment
Allocated : 4540 kb
Allocated: 167 kb
glass1
Robot Simulation Interface File Action Tools Help
Share Object
Advanced Controls
Connected to 127.0.0.1
Allocated: 6420 kb
80.23 fps
Allocated: 7132 kb
23.07 fps

Fig. 8. World sharing feature of the ORCA simulator. The server object (left) can load an environment centrally, and communicate individual parts to different simulation instances (top and bottom right, respectively) based on the location of their main robot.

configuring its environment file, ORCA allows a developer to transfer the processing of an object into a remote simulation instance.

When a simulation instance receives a certain set of published properties, it has the ability to render them statically, i.e. without considering the object geometries, physical properties and mesh information, or load the object in its own right, and only embed the published properties provided by the server. This feature becomes particularly important when sharing environments that are cluttered with many objects. In this case the server provides the ability to publish specific object properties, which can then be shared across simulation instances.

Herewith, we describe how ORCA's communication protocol can be setup across different simulation instances, in order to communicate processed properties regarding an object. To illustrate this, we setup a simulation in which two robotic models interact with each other with a common object. For the current simulation purposes we employ the Kuka LWR arm and the generic mobile platform (Fig. 9). The following scenario has been included in the demonstration video which can be found at ORCA's url (http://www.ics.forth.gr/cvrl/index_main.php?l=e&c=17). In an environment consisting of two robots, we place one cube object on the table. We setup ORCA to process the two robots on two different simulation instances. The first instance is responsible for processing the environment and the Kuka arm model (model A), while the second the environment and the mobile platform (model B). Moreover, a cube is placed on top of the object, and model A must pass the cube to model B. The above experimental paradigm has one important characteristic. It consists of an environment with multiple robots, each being processed in its own instance. The two models interact with the environment, while objects can be shared across instances.

To setup this paradigm we use ORCA's network protocol (shared mesh messages), used to share physics properties among different simulation instances. During the simulation, model A interacts with the object, whilst model B receives information about the object's position, orientation and net forces acting on it. These are delivered by the server in a timely manner, through ORCA's messaging system. When model B detects that the location of the cube object is closer than a predefined location, it activates the physics of the object and starts processing it as required. Figure 9 illustrates the concept.

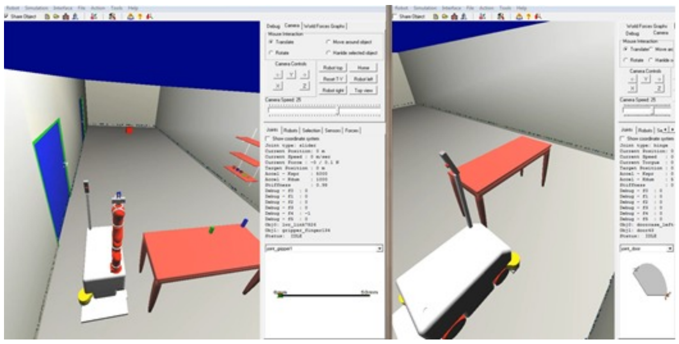


Fig. 9. Different simulation instances can share objects distributively. In the example shown, two robots (left and right) interact with an object, in their own simulation instance, and communicate changes using the server.

One major benefit from this setup is that ORCA can computationally handle large environments by distributing simulation costs across different instances. That is, different parts of the environment can be modeled in different simulators, and only the objects that cross-interact between them will be processed. In the robotics literature one can find a wide range of topics that this feature is important, including team-robots, such as robotic football, or swarm robotics.

5 ORCA USE CASES

A valuable feature of the ORCA simulator is that it is easily extensible. This is very important for robotics projects, since it is a usual case for software and hardware modules to be subject to change [29]. In ORCA, developers can augment the software at any time to support additional functions. This is accomplished by extending its message protocol, using messages that can support any arbitrary data structure. The server object is then responsible to accommodate the integration of external software modules, only by setting up compatible communication protocols.

As mentioned, the package has already been employed in a large number of national and international projects, including the EC-funded FP6 projects GNOSYS [8] and INDIGO [39], [6], the FP7 project First-MM [5], as well as several national projects. As a result of these projects, it was extended to include several add-ons. In the following section we outline these use-cases and discuss how ORCA was fit into their development lifecycle.

5.1 INDIGO

The goal of INDIGO was to develop technology that facilitates the advancement of human-robot interaction. This was achieved both by enabling robots to perceive natural human behavior, as well as by making them act in ways that are familiar to humans. To achieve these goals, INDIGO attempted

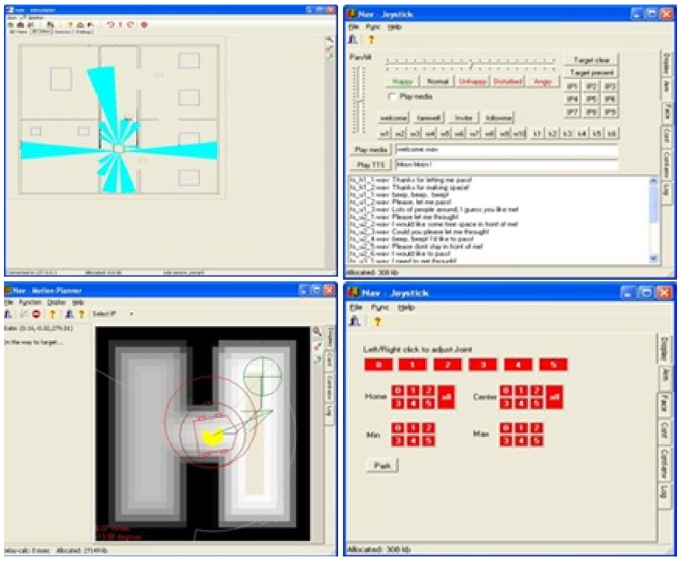


Fig. 10. Various add-on modules have been developed to augment ORCA's functionality, including: Simultaneous localization and mapping (top left), navigation and planning (bottom left), language modules (top right) and HCI interfaces(bottom right).

to exploit and advance various enabling technologies. It employed a mechanical head capable of mimicking human facial expressions, and supporting naturalistic spoken conversation. Advanced natural dialogue capabilities facilitated human-robot interaction, involving input and output from various modalities, such as spoken natural language, gestures, emotions, and facial expressions.

Role of ORCA: To facilitate this tight development lifecycle, ORCA provided a simulation platform that was able to utilize the output from software modules with ease. During the course of the project, several add-on components have been developed, which used ORCA to communicate with each other. The first class of add-ons regards human-robot communication technologies (Fig. 10). These include components for natural language interaction and virtual emotions, as well as HRI interfaces for human-robot interaction. The second class of add-ons that have been developed pertains to locomotion modules. These modules used ORCA's simulator to replicate the mobile robot (along with sensor and odometry readings), and ORCA's server communication system, in order to handle intercommunication issues.

5.2 GNOSYS

The aim of GNOSYS was to develop and validate a conceptual system for Cognitive Agents, with special emphasis on the design of an abstraction architecture used to represent the knowledge that the agent possesses of the environment and itself. To accomplish this, the GNOSYS's agent was able to

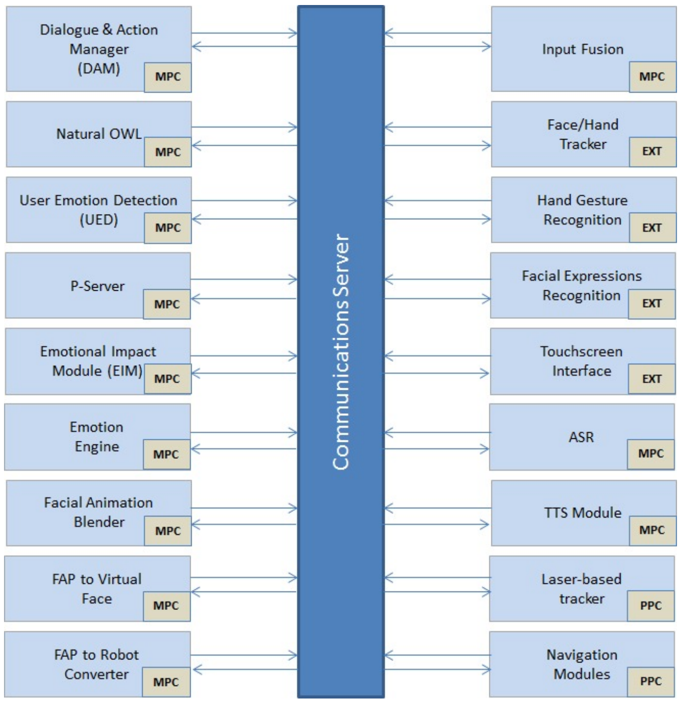
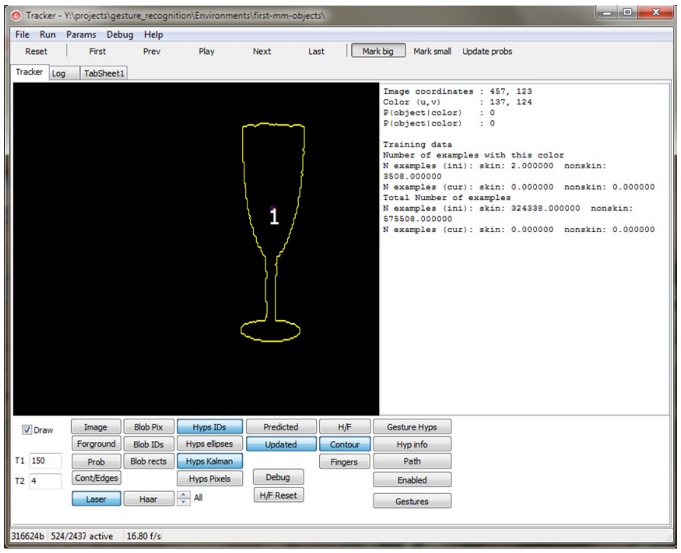


Fig. 11. ORCA's communication protocol extended during the course of the GNOSYS project



```
Image coordinates : 457, 123
Color (u,v)       : 137, 124
P(object|color)   : 0
P(object|color)   : 0

Training data
Number of examples with this color
N examples (ini): skin: 2.000000  nonskin:
3508.000000
N examples (cur): skin: 0.000000  nonskin: 0.000000
Total Number of examples
N examples (ini): skin: 324338.000000  nonskin:
575508.000000
N examples (cur): skin: 0.000000  nonskin: 0.000000
```

Fig. 12. Loading and rendering of object contour models using the ORCA simulator.

integrate knowledge about objects, relations, goals, context information, and solution strategies, and derive useful properties about a situation. Moreover, two additional mechanisms would operate on top of this system: (i) an abstraction mechanism, responsible for creating and organizing a hierarchy of concepts and (ii) a reasoning process which operated on the concept system and was used to make inferences for virtual actions and select the one that will realize the greatest reward. The architecture was highly distributed, and included components of attention control, prioritising responses, detecting novelty and creating new goals. Both sensory and motor attention were also employed, while a goal-oriented computational model was used to fuse together user tasks with tasks originating from the agent.

Role of ORCA: ORCA was used to simulate various components developed in the project, with significant efforts being directed towards ensuring that this integration is achieved at both the conceptual and technical levels. In addition, it was extended to include add-ons pertaining to the simulation of virtual faces and spoken dialogue systems (Fig. 11).

The aforementioned extension in the ORCA software contributed both in increased robustness, communication speed and new processing and visualization capabilities. Using ORCA's platform, all the software modules of GNOSYS were redesigned and deployed in the new platform, using the unified model above. More importantly, all GNOSYS modules, were equipped with GUI-based implementation. The overall GNOSYS demonstrator included totally 25 software modules, from which the 11 constitute the core implementation.

5.3 First-MM

The goal of First-MM was to build the basis for a new generation of autonomous mobile robots that can flexibly be instructed to perform complex mobile manipulation. The project developed a novel robot programming environment that allows even non-expert users to specify complex manipulation tasks in real-world environments.

To accomplish these goals, the project extended research works in robot programming, navigation, manipulation, perception, learning by instruction, and statistical relational learning. All these were integrated together to develop an advanced technology for mobile manipulation robots, that can flexibly be instructed even by non-expert users to perform challenging manipulation tasks in real-world environments. Integration of the individual components was accomplished through ORCA's communication protocol, in which various components of mobile manipulation and tracking were fused together. The integrated system served as a prototype that was able to flexibly solve mobile manipulation tasks like object transportation activities in complex and dynamic environments.

Role of ORCA: ORCA was extended with a novel software architecture, to simulate the environments and objects dealt in the First-MM project. Support was added for photorealistic rendering based on OpenGL, and material libraries for representing object information. Moreover, to aid the design of robotic structures, standard models of robot kinematics were added to the simulator, in addition to a variety of robot sensors, as well as several add-ons, regarding pose estimation and object tracking (Fig. 12). Integration, during the initial stages, was carried out completely on the simulation platform.

6 Conclusion and future work

Simulation is a cost-efficient and robust solution for robotics research. ORCA uses a central server object, which is responsible for communicating information about a simulation towards any peer that requests it. As a result, it can handle large scale projects, by enabling development activities to occur at arbitrary locations. In the current paper we demonstrated how this setup can be used to reduce the processing requirements of scalable environments. To accomplish this, we have made use of ORCA's server, in order to transmit already processed physics properties about an object in other simulation instances throughout the network. One can make use of this important property to simulate large environments: Robots can be setup to process only the physics properties of an entity that are communicated by the server, thus avoiding computationally intensive processing.

One of the main focuses of the simulator is the seamless definition and management of environments. Towards this end, ORCA provides a powerful GUI, which allows designing and manipulation of an environment directly from the software. The above effectively render ORCA an ideal robotic simulator for demanding research and development activities. In the immediate future we plan to incorporate additional robotic platforms, such as the NAO humanoid robot. In addition we are working towards releasing the source code of the software, and redistributable clients for various platforms.

Acknowledgments

This work has been partially supported by the European Commission under contract number FP7-248258 (First-MM project).

References

[1] L. Žlajpah, "Simulation in robotics," *Mathematics and Computers in Simulation*, vol. 79, no. 4, pp. 879–897, 2008. 1

[2] O. Khatib, O. Brock, K.-S. Chang, F. Conti, D. Ruspini, and L. Sentis, "Robotics and interactive simulation," *Communications of the ACM*, vol. 45, no. 3, pp. 46–51, 2002. 1

[3] M. Reckhaus, N. Hochgeschwender, J. Paulus, A. Shakhimardanov, and G. K. Kraetzschmar, "An overview about simulation and emulation in robotics," *Proceedings of SIMPAR*, pp. 365–374, 2010. 1

[4] E. Hourdakis, G. Chliveros, and P. Trahanias, "Orca: A physics based, robotics simulator able to distribute processing across several peers," in *Proceedings of the International Symposium on Robotics, 2013 Seoul*, 2013. 1

[5] FIRST-MM, "Flexible Skill Acquisition and Intuitive Robot Tasking for Mobile Manipulation in the Real World," http://www.first-mm.eu/, [Online; accessed: 08-Oct-2013]. 1, 5

[6] INDIGO, "Interaction with Personality and Dialogue Enabled Robot," http://www.ics.forth.gr/indigo/, [Online; accessed: 08-Oct-2013]. 1, 5

[7] XENIOS, "Human robot interaction based on visual, audio and natural language interpretation for use in robotic systems," http://www.ics.forth.gr/xenios/, [Online; accessed: 08-Oct-2013]. 1

[8] GNOSYS, "An Abstraction Architecture for Cognitive Agents," http://www.ics.forth.gr/gnosys/main/main_frames.html/, [Online; accessed: 08-Oct-2013]. 1, 5

[9] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3. IEEE, 2004, pp. 2149–2154. 2

[10] G. Echeverria, N. Lassabe, A. Degroote, and S. Lemaignan, "Modular open robots simulation engine: Morse," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 46–51. 2

[11] O. Michel, "Webotstm: Professional mobile robot simulation," *arXiv preprint cs/0412052*, 2004. 2

[12] M. Freese, S. Singh, F. Ozaki, and N. Matsuhira, "Virtual robot experimentation platform v-rep: a versatile 3d robot simulator," in *Simulation, Modeling, and Programming for Autonomous Robots*. Springer, 2010, pp. 51–62. 2

[13] T. Laue, K. Spiess, and T. Röfer, "Simrobot-a general physical robot simulator and its application in robocup," in *RoboCup 2005: Robot Soccer World Cup IX*. Springer, 2006, pp. 173–183. 2

[14] R. Smith *et al.*, "Open dynamics engine," 2005. 2, 3.2

[15] R. Hess, *The essential Blender: guide to 3D creation with the open source suite Blender*. No Starch Press, 2007. 2

[16] E. Coumans *et al.*, "Bullet physics library," *Open source: bulletphysics.org*, 2006. 2, 3.2

[17] J. Jarez and A. Suero, "Newton game dynamics," 2008. 2, 3.2

[18] B. Mirtich and J. Canny, "Impulse-based simulation of rigid bodies," in *Proceedings of the 1995 symposium on Interactive 3D graphics*. ACM, 1995, pp. 181–ff. 2

[19] A. Seugling and M. Rolin, "Evaluation of physics engines and implementation of a physics module in a 3d-authoring tool," *Umea University*, 2006. 2

[20] A. Boeing and T. Bräunl, "Evaluation of real-time physics simulation systems," in *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*. ACM, 2007, pp. 281–288. 2

[21] C. HECKER and J. LANDER, "Product review of physics engines, part one: The stress tests [en lınea]," Tech. Rep. 2

[22] C. Hecker and J. Lander, "Product review of physics engines, part two: The rest of the story," Technical report, Tech. Rep., 2000. 2

[23] S. Balakirsky, S. Carpin, G. Dimitoglou, and B. Balaguer, "From simulation to real robots with predictable results: methods and examples," in *Performance Evaluation and Benchmarking of Intelligent Systems*. Springer, 2009, pp. 113–137. 2

[24] E. Woo, B. A. MacDonald, and F. Trépanier, "Distributed mobile robot application infrastructure," in *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, vol. 2. IEEE, 2003, pp. 1475–1480. 2

[25] O. Kubitz, M. O. Berger, and R. Stenzel, "Client-server-based mobile robot control," *Mechatronics, IEEE/ASME Transactions on*, vol. 3, no. 2, pp. 82–90, 1998. 2

[26] C. Kapoor and D. Tesar, "A reusable operational software architecture for advanced robotics," Ph.D. dissertation, Citeseer, 1996. 2

[27] S. A. Schneider, V. W. Chen, and G. Pardo-Castellote, "The controlshell component-based real-time programming system," in *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on*, vol. 3. IEEE, 1995, pp. 2381–2388. 2

[28] M. Montemerlo, N. Roy, and S. Thrun, "Perspectives on standardization in mobile robot programming: The carnegie mellon navigation (carmen) toolkit," in *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, vol. 3. IEEE, 2003, pp. 2436–2441. 2

[29] A. Orebäck and H. I. Christensen, "Evaluation of architectures for mobile robotics," *Autonomous robots*, vol. 14, no. 1, pp. 33–49, 2003. 2, 5

[30] ROS, "Robot Operating System ," http://www.ros.org/, [Online; accessed: 08-Oct-2013]. 2

[31] G. Metta, P. Fitzpatrick, and L. Natale, "Yarp: yet another robot platform," *International Journal on Advanced Robotics Systems*, vol. 3, no. 1, pp. 43–48, 2006. 2

[32] H. Utz, S. Sablatnog, S. Enderle, and G. Kraetzschmar, "Miro-middleware for mobile robot applications," *Robotics and Automation, IEEE Transactions on*, vol. 18, no. 4, pp. 493–497, 2002. 2

[33] H. Bruyninckx, "Open robot control software: the orocos project," in *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, vol. 3. IEEE, 2001, pp. 2523–2528. 2

[34] N. Ando, T. Suehiro, and T. Kotoku, "A software platform for component based rt-system development: Openrtm-aist," in *Simulation, Modeling, and Programming for Autonomous Robots*. Springer, 2008, pp. 87–98. 2

[35] R. T. Vaughan, B. P. Gerkey, and A. Howard, "On device abstractions for portable, reusable robot code," in *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, vol. 3. IEEE, 2003, pp. 2421–2427. 2

[36] D. Brugali and M. E. Fayad, "Distributed computing in robotics and automation," *Robotics and Automation, IEEE Transactions on*, vol. 18, no. 4, pp. 409–420, 2002. 3, 3.1, 3.1

[37] S. Wasson, "Ageia's physx physics processing unit," The tech report, PC hardware explored, Tech. Rep., 2008. 3.2

[38] K. Erleben, J. Sporring, and H. Dohlmann, "Opentissue-an open source toolkit for physics-based animation," in *MICCAI Open-Source Workshop*, 2005. 3.2

[39] S. Konstantopoulos, I. Androutsopoulos, H. Baltzakis, V. Karkaletsis, C. Matheson, A. Tegos, and P. Trahanias, "Indigo: Interaction with personality and dialogue enabled robots," *Artificial Intelligence*, pp. 5–6, 2009. 5

Emmanouil Hourdakis is a post-doctoral researcher at the Computational Vision and Robotics Laboratory of the Foundation for Research and Technology - HELLAS (FORTH). He has extensive research experience in computational modeling and control methods for robotics, while he has collaborated in several national and international robotics projects. His recent work includes bio-inspired locomotion, physics-based simulation, computational modelling, cognitive neuroscience and machine learning. He is an active reviewer in several conferences and Journals.

Georgios Chliveros is a research engineer with postgraduate and doctoral studies in Pattern Recognition. Prior to joining CVRL at FORTH-ICS, he received research tenures with HP Labs (UK), the Materials and Engineering Research Institute (Sheffield, UK), the Sheffield Centre for Robotics (UK), and held an industrial exchange fellowship with the DCIS lab of Thales RnD (NL). He has worked on EU and UK, NL and GR national projects on robotic systems development, with emphasis on localisation and tracking problems.

Panos Trahanias is a Professor with the Department of Computer Science, University of Crete and the Foundation for Research and Technology - Hellas (FORTH). Currently, he is the Head of the Computational Vision and Robotics Laboratory at FORTH. Prior to joining University of Crete and FORTH (1993) he had research appointments with the Institute of Informatics and Telecommunications, National Center for Scientific Research Demokritos, Athens, Greece (1989-1991), the Department of Electrical and Computer Engineering, University of Toronto, Canada (1991-1993), and has been a consultant to SPAR Aerospace Ltd., Toronto (1992-1993). His research interests are in human-robot interaction, robot navigation, remote-access robotic systems, brain-based modeling and augmented reality applications. He has acted as coordinator of EU-funded projects and numerous nationally funded projects, and has participated in many RTD projects. He has published over 120 papers in technical journals and conference proceedings, has contributed in two books, and has been General Chair of EUROGRAPHICS 2008 and General Co-Chair of ECCV 2010.