Framework for Haptic Interaction with Virtual Avatars

Paul Evrard, François Keith, Jean-Rémy Chardonnet, Abderrahmane Kheddar

Framework for Haptic Interaction with Virtual Avatars

Paul Evrard François Keith Jean-Rémy Chardonnet Abderrahmane Kheddar

Abstract— **In this paper we present an integrative framework centered on haptic interaction with virtual avatars. This framework is devised for general prototyping and collaborative scenario studies with haptic feedback. First we present the software architecture of the framework and give details on some of its components. Then we show how this framework can be used to derive in a short time a virtual reality simulation. In this simulation, a user directly interacts with a virtual avatar to collaboratively manipulate a virtual object, with haptic feedback and using fast dynamics computation and constraint-based methods with friction.**

I. Introduction

Recently, several simulators for virtual avatars have been developed for various purposes and applications. In the computer graphics community, avatars dynamic simulators are proposed with dynamic controllers for off-line or interactive physics behavior for a plausibly realistic animation in gaming or motion generation of digital actors, e.g. see [1][2][3], but to our best knowledge interactivity with haptic feedback is not of a major concern or is not integrated in a generic framework. In the robotics field, dynamic simulators have also been developed for the purpose of planning or sensory control simulation. For example, Son *et al.* [4] proposed a general framework for robotic dynamic simulation with interactive capabilities including force feedback. This simulator accounts for constraint-based contact modeling with friction; it uses a hybrid method to switch between different contact status. Khatib's team proposed an impressive framework, called SAI, for interactive dynamic simulation [5] using the operational space formulation with prioritized tasks [6] and was probably among the firsts to use haptic feedback for interaction with a virtual avatar. At AIST, an open platform named OpenHRP has been developed to be dedicated to the general humanoid studies [7]. The next release (version 3) is forecast to be distributed freely with open source code. However, OpenHRP is not interactive and does not include force feedback. There are more advanced versions that are commercially available or forecast to be so (e.g. R-Station or Microsoft Robotic Studio). In virtual prototyping, Duriez *et al.* [8] showed interactive simulation for deformable objects using constraint-based methods and solving contact with friction using iterative algorithms that include haptic feedback. Using the same basis as Ruspini and Khatib's work [5], Chardonnet *et al.* [9] proposed a fast dynamic simulator for humanoids without discreete Coulomb's friction cones and able of handling complex shapes.

The authors are with AIST/CNRS Joint Japanese-French Robotics Laboratory, Tsukuba, Japan {evrard.paul,francois.keith, jr-chardonnet,abderrahmane.kheddar}@aist.go.jp

This work extends our previous framework [10] to handle fast constrained-based dynamic computation with force feedback. Comparing to previous work, we demonstrate that our simulator is able to handle complex shapes in real-time and integrate force feedback without any specific treatment. Our framework intends to include not only task-driven simulations, but also cognitive aspects linked to haptic interaction such as haptic communication and advanced interaction with digital actors that can be either virtual or real (robots). This paper focuses on the proposed integrative software architecture. The dynamics and computer haptics models have been discussed in [10]. This framework is devised to integrate developments in digital actors control with a focus on haptic tasks and communication.

II. Software architecture

A. Presentation of AMELIF

Our software developments were realized under an integrative framework called AMELIF. This framework proposes a structure and an API for the representation of virtual scenes including articulated bodies, and interfaces for driving simulations and manipulating the elements of the scene. AMELIF has been created in order to allow fast and easy prototyping of virtual reality algorithms (related to collision detection, dynamics, interaction with virtual avatars...), and most particularly algorithms related to virtual avatars. The modularity of our framework allows the customization and replacement of most components without modifying the core application, thus guarantying consistency between the developments and interoperability between customized components. AMELIF is a cross platform framework and has been successfully tested under the Windows and Ubuntu Linux operating systems. It is based on the wxWidgets C++ library [11], but the wxWidgets API is hidden to the users of the framework so that they do not need to know anything about this library. The components developped under this framework can be either open-source or released as dynamic libraries.

Building modules as separate libraries allows customization and prototyping while manipulating as little code as possible: to work on high-level interaction with virtual avatars, only the binaries of the required modules are needed, which relieves from the pain of setting up all the modules and building them in a monolithic way. It also allows users who do not care for some modules not to have to know they exist: a user who is not interested in dynamics will never have to manipulate interfaces related to dynamics as he will simply not install the dynamic simulation module.

AMELIF includes a core application to display and run the simulations, and a core library that provides interfaces for communication with the core application and a set of utilities and interfaces that can be used by all the components as a common basis. One of the communication interfaces allows writing simulations that can be run from the core application. This interface has three main methods: one that is responsible for the initialization of the scene, one that drives one simulation step, and one for the cleanup. The state of the virtual scene is accessible from these methods. Each implementation of this interface has to be compiled as a dynamic library that will be loaded by the application. The application will automatically call the methods for initialization, simulation and cleanup; the simulation will be executed in its own thread.

B. Architecture and components

Modules developped under AMELIF are intended to work together with minimal dependencies (e.g. a component should not depend upon its client components). Components integrated to this framework eventually grow and get more complex with time, as the framework has been devised for prototyping. Components should be robust to changes of the components from which they take their input, while having no specific knowledge about their own client components. Finally, algorithms should all be written in such a way that they need not be edited when new components are developped.

Four modules have already been developped and integrated to AMELIF: the scene library, the collision detection module, the devices library, the dynamic simulator. Figure 1 shows the dependencies between the interfaces of each module. The dialog boxes and main programs created by users of AMELIF communicate with the application through the core library. They can capture and handle events generated by the mouse or the keyboard and send instructions to the application to modify its behavior. All the modules potentially use the interfaces and tools provided by the core library (although this is not necessary, this is the case for all the modules we integrated up to now).

1) Scene library: The scene library provides components for the creation, display and manipulation of virtual scenes. A virtual scene in AMELIF contains two kinds of bodies: simple bodies and linked bodies. An articulated body is a set of linked bodies connected by joints. ”Decorations” can be attached to bodies (e.g. frames attached to bodies), which position and orientation can be defined relative to their parent body in the scene or in the world frame. Bodies are associated with basic information e.g. geometry, mass, position, external forces, inertia.

The scene library is central as it stores the current state of the scene and all its geometric information. Modules can indirectly interact together through the scene library by modifying the state of the scene: for example controllers will act on the joint torques of articulated bodies, while the dynamic simulator will read these torques and use them

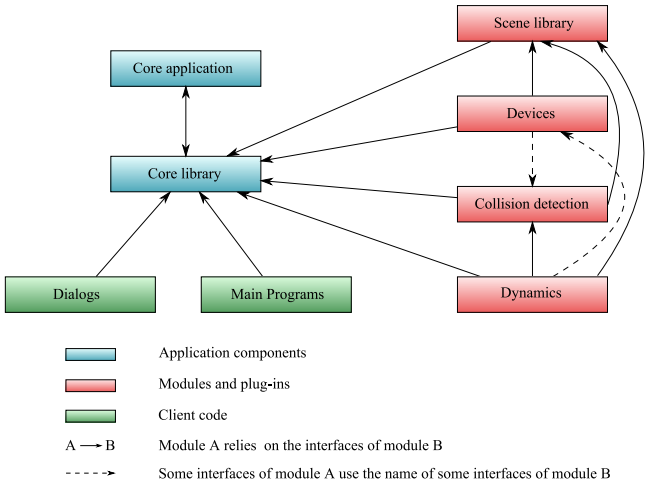

Fig. 1. Dependencies among interfaces of AMELIF

together with the forces applied on bodies to update their accelerations and the joint accelerations. Almost all modules developped under AMELIF will rely on the scene library as it provides the common representation of the scene and its entities and the interfaces to manipulate them. For this reason, modularity and flexibility were major constraints during the development of this module. Each component of the scene library can be customized independently, and without having to modify existing code of the module. One can, for example, use its own representation of multi-bodies, rather than the default one, without modifying the other elements of the scene or existing parsers. This mechanism allows the extension and customization of the scene library at various levels without editing any existing code. Moreover, as long as the customized components comply with the interfaces of the module (using a Bridge pattern [12] if necessary), they can be used by other researchers with very little effort.

2) Collision detection: The collision detection module provides interfaces to trigger the detection of collisions between sets of bodies of the scene. A user has the possibility to create different sets of bodies and to detect collisions among these sets independently. This can be useful when the virtual scene includes elements that are totally separated. Among the groups, it is possible to associate flags to pairs of bodies. These flags will determine which data is required from the collision detection low-level library about the specified pair of bodies (proximity distance, interpenetration...). Finally, it is possible to exclude pairs from a set of bodies, or to add pairs and detect collisions among the specified pairs only.

3) Device management: The devices library manages external devices such as haptic devices. It provides a high-level interface to get information about various devices that can be used within AMELIF. The haptic device library possibly uses the collision detection and

the dynamic simulator to handle interactions between the haptic probe and the virtual environment. This dependency can be reversed if the haptic device is provided with its own collision detection library and the ability to compute the feedback force; in this case, the dynamic simulator would rely on the haptic device to take into account the interaction force between the haptic tool-tip and the environment for the computation of direct dynamics.

4) Dynamic simulator: The dynamic simulator provides interfaces to drive a dynamic simulation and to run usual algorithms related to dynamics (direct dynamics, computation of the operational space inertia matrix...). The dynamics simulation is responsible for computing accelerations from the forces applied on bodies and for resolving algorithms to handle contacts, impacts, deformations, etc. It is also responsible for integrating the computed accelerations and updating the state of the scene. This module depends on the collision detection for bodies and articulated bodies not to pass through each other.

The collision and dynamics modules can communicate with other modules in two ways: they can either be used as usual components that are queried on demand; or they can be used as publishers that will send events to their listeners, according to the well-known Observer (or Publisher/Subscriber) design pattern[12]. This way of communicating with other modules has a strong advantage upon the usual query-on-demand communication as it allows a clear and automatic separation between algorithms and code that handles output data types when polymorphism can not easily be used (a typical case in communication among modules). Client code can also be made aware of new output information automatically, whereas with query-on-demand, the user needs to read the documentation of the queried module to keep aware of new outputs. This mechanism is used by the collision detection module to send information about detected collisions to all the subscribers modules, which can either handle the collision as soon as it is received or store the information for future use. Likewise, the dynamic simulator publishes interactions, which are collisions augmented with a contact force information. A publisher module has the responsibility for defining the interface of its subscribers. Thus, the publisher module has control over the data that a client module can receive and can or must handle. A publisher module is aware of the existence of its clients and thus can have control over them while being independent of the specific type of its client.

C. Default implementation

1) Collision detection and dynamic simulation: Each module integrated to AMELIF defines a set of interfaces with a default implementation for each interface. Additional implementations can be developped if needed, and these implementations can be substituted to the default ones if they follow the contracts established by the interfaces. The default implementation of the collision detection relies on PQP[1] and supports the computation of proximity distances and intersections between triangles. Our dynamic simulator is implemented as a listener of the collision detection module. Once the collision detection module is initialized, the dynamic simulator will register as one of its subscribers. Each time a collision is detected, the dynamic simulator will preprocess the collision and store the resulting information for the next step of the direct dynamics. Currently, the dynamic simulator listens to all collisions types published by the collision detection but only handles punctual collisions.

The implementation of the direct dynamic model based on Featherstone's Articulated Body Method (see in [10], [9]), strongly relies on the joints of a multi-body. Whereas the same information will always be needed from the links of an articulated body (mass, inertia, position...), information about the joints is subject to changes depending on the joint types. While with one degree-of-freedom joints, several quantities that are involved in the Featherstone's algorithm are real values, the same quantities for spherical joints are matrices. The joint positions and velocities will likewise be real values for single degree-of-freedom joints and vectors or quaternions for multiple degrees-of-freedom joints. Explicitly relying on the joint type in the Featherstone algorithm would make the maintenance of the direct dynamic model component difficult, as the algorithm would need to be rewritten for each new joint type that would be taken into account. Therefore, we wrote the algorithm in term of a polymorphic joint hierarchy. Each type of joint handles its specific computations itself; the code outside the polymorphic joints is generic and will thus be robust to the addition of new joint types or to any change in the representation of joints.

The dynamic simulator can mix collisions handled with the penalty method and collisions handled with constraint-based methods. The general algorithm for one step is the following:

- First, apply the gravitational force to each body.
- Then check for interactions between the haptic device (if any) and the scene; add the interaction force to the current force applied to the body in contact with the haptic probe.
- Then handle penalty interactions: compute the penalty force and add it to the current forces applied on interacting bodies. Then compute the free accelerations of all the bodies and joints.
- Handle constraint-based interactions, compute the constrained accelerations and add them to the free accelerations.
- Finally integrate the system.

2) Haptic interfacing: Our framework is centered on haptic interaction with virtual environments and with virtual avatars. An example of such a haptic interaction is the realization of a collaborative manipulation task with haptic feedback. For this purpose, we interfaced the PHANToM© Premium 1.5 High Force™ device commercialized by SensAble

[1] www.cs.unc.edu/~geom/SSV

Technologies[2]. This device has six degrees of freedom with a six degree-of-freedom force/torque feedback.

For the time being, we consider two different ways of interacting: touching or dragging objects (including avatars). In both cases, interaction will add an external force $\mathbf{f}_e$ to the dynamics of the objects:

$$\ddot{\mathbf{q}} = \mathbf{A}^{-1}\left(\mathbf{\Gamma}(\mathbf{q}) - \mathbf{b}\left(\mathbf{q}, \dot{\mathbf{q}}\right) - \mathbf{g}(\mathbf{q})\right) + \mathbf{A}^{-1}\mathbf{J}_c^T\mathbf{f}_c + \mathbf{A}^{-1}\mathbf{J}_e^T\mathbf{f}_e \tag{1}$$

When an object is touched, the feedback force is obtained via the Sensable OpenHaptics™ Toolkit. This library provides a collision detection algorithm that computes the feedback force due to contact between the virtual tip of the PHANToM© handle and the virtual environment, as well as the coordinates of the contact point. The interaction force at the contact point is computed by weighting the feedback force by an arbitrary chosen coefficient. This force is added to the dynamics equation during the computation of the free acceleration of the bodies.

Dragging objects is useful for tasks like pick-and-place, pushing or collaborative tasks (with avatars). As the user moves the device, the object behaves accordingly. To achieve this, the most common way used in interactive simulations is to model a virtual coupling between the device and the dragged object using a virtual spring-damper force:

$$\mathbf{f}_e = k_p(\mathbf{x}_{\text{device}} - \mathbf{x}_{\text{contact}}) + k_v(\dot{\mathbf{x}}_{\text{device}} - \dot{\mathbf{x}}_{\text{contact}}) \tag{2}$$

where $\mathbf{x}_{\text{device}}$ and $\mathbf{x}_{\text{contact}}$ are the positions/orientations of the haptic device and of the object, respectively. k_v is chosen so that $k_v = \sqrt{2mk_p}$, with m the mass of the object. Again, this force is added to the dynamics equation during the computation of the free acceleration.

III. Demonstrator

To illustrate the advantages of our software architecture and the capabilities of our simulator, we developped a small demonstrator in which a user interacts at different levels with a virtual avatar. We also introduce some modules that are currently under development and being integrated to AMELIF: the Haptic Perception module and the Visual Perception module. The goal of this section is to emphasize the possibilities offered by our framework for prototyping algorithms centered on interaction with virtual avatars.

A. Scenario

In this demonstration, a virtual HRP-2 humanoid robot stands in front of a table upon which lies an object. The robot first goes to a predefined initial posture, and then stands idle. The user touches the robot with a haptic probe; the robot is triggered by this action and stares at the body that was just touched. The robot then sees the haptic probe near the touched body (if the haptic probe is in its field of view). It thus starts to stare at the probe. The user moves the probe toward the object on the table and just waits aside. After a given amount of time during which the probe is close to the object, the robot starts focusing on the object and graps it. Once the object is grasped, the robot's arm is compliant to forces applied on the object, which allows the user to move the object to any configuration together with the robot. Let us examine how such a behavior can be implemented under AMELIF and how, starting from this implementation, virtual prototyping for Virtual Reality-related algorithms centered on human-avatar interaction can easily be realized.

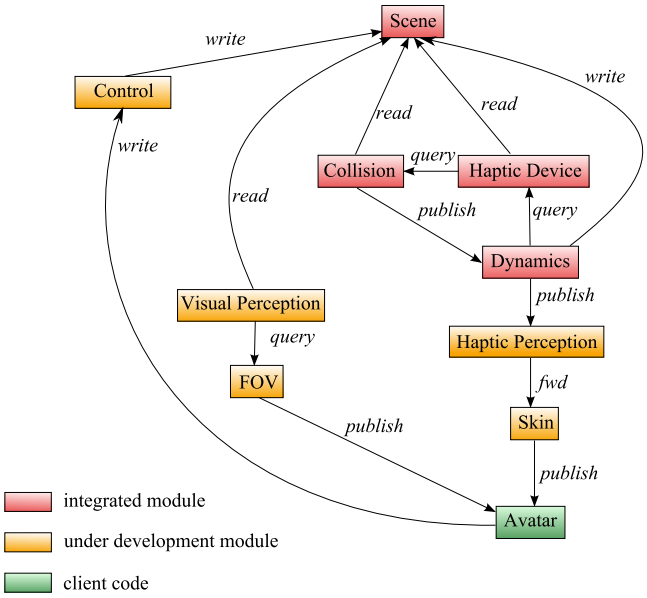

Fig. 2. Structure of the demonstrator

B. The virtual HRP-2 avatar

The highest level component of the demonstrator is the HRP-2 avatar. This component will implement a finite state machine that will drive the avatar into different states. In each state, a different set of behaviors are activated. From the definition of the scenario, five states can be drawn: the "going to initial position" state, the "idle" state, the "focus on cursor" state, the "grasping phase" state and the "task" state. As the scenario is very simple, these states are just sequential. Once we leave a state, we can never come back to this state. This make the implementation of the Finite State Machine very easy. The "going to initial position" state is straightforward and just rely on controllers that will servo the joints to follow specified trajectories defined off-line. The "idle" state is just a state where the robot keeps its current position and can just rely on a simple PD controller.

C. Dynamic simulation

The very first thing to do is to simulate the virtual environment. AMELIF provides default implementations for the visual and haptic rendering of a scene, low-level haptic interaction with free-bodies and constraint-based dynamic simulation with frictions. All we have to do is to load the modules, parse XML files in which the environment is described using the parser provided with the framework, and run the components in the main program.

[2] www.sensable.com

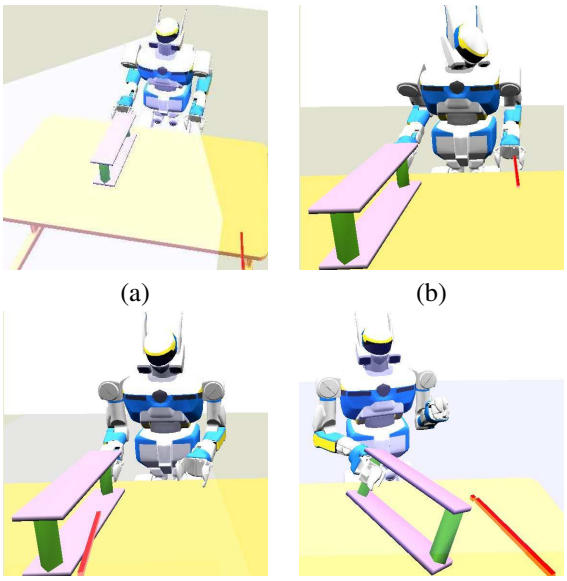

Fig. 3. The different steps of the simulation: (a) Go to initial configuration (b) Trigger the robot by patting its hand (c) Show it the object (d) Move the object together

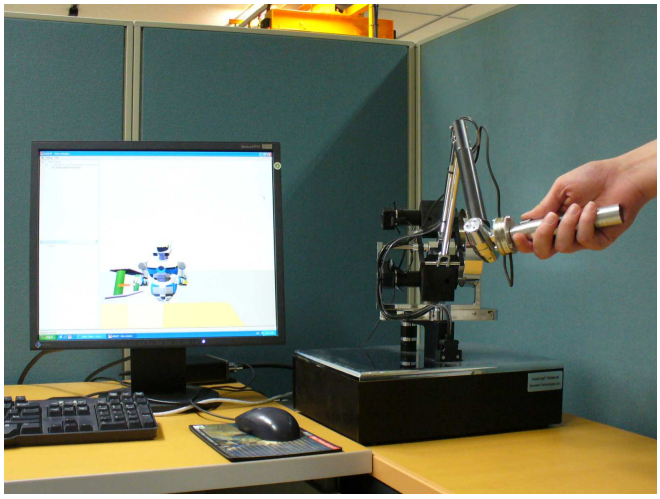

Fig. 4. Interacting with the HRP-2 avatar using a haptic device

D. Focus on the haptic probe

The ”idle” state is left when the robot is touched. Therefore, we need to let the HRP-2 avatar know when something touches it. Typically, this can be implemented with the Skin component, which is part of the Haptic Perception module. This module is currently under development and proposes interfaces and components related to haptic perceptions (various force sensors, skins...). The Haptic Perception module is implemented as a subscriber of the dynamic simulator. Indeed, the simulator outputs collisions and their associated interaction forces and bodies. When a virtual avatar is registered in the Haptic Perception module as having a skin, the Haptic Perception module will trigger it each time this avatar is involved in a collision, and will send it the associated interaction force. This communication between the Haptic Perception module and the avatar is realized through a Skin component, which allows users to model various phenomena related to the perception of forces by the skin.

The ”focus on cursor” state can be implemented in a similar way. Instead of the Haptic Perception module, we use a Visual Perception module. This module allows an avatar to be triggered on different conditions. The simplest way of triggering the avatar is to associate a Field of View (FoV) component to the avatar. A FoV component raises messages when objects of the scene that have been registered as visible objects in the Visual Perception Module enter and leave the field of view. We have the possibility to easily add uncertainty about the position of the objects in the field of view by adding an occluding component between the FoV and the avatar.

In our demonstration, we did not add any uncertainty and considered the avatar was able to perfectly determine the position of objects in his field of view. Using a very simple controller, we servo the joints of the robot's neck so as to stare at the cursor when the cursor is inside the field of view. When both the haptic probe and the object on the table are in the field of view, the avatar looks at the distance between them and if it stays under a given threshold during a given amount of time, the ”focus on cursor” state is left and the ”grasping phase” state is entered.

In our demonstration, the grasping phase is implemented using pre-defined joint trajectory. However, using the jacobian computation algorithm provided in the scene library, an inverse kinematics algorithm could be used to automatically reach the object at any location.

E. Task

In the ”task” state, the virtual HRP-2 was position-controlled using a PD joint controller. To make the robot compliant to the interaction force with the haptic device, the desired joint positions q_d are defined by the following equation:

$$M\ddot{\mathbf{q}}_d + B\dot{\mathbf{q}}_d = \mathbf{J}_e^T \mathbf{f}_e \tag{3}$$

where M and B are diagonal matrices of positive coefficients corresponding to a virtual inertia and a virtual damping, $\mathbf{f}_e$ is the interaction force with the haptic device, and $\mathbf{J}_e$ is the Jacobian relating joint velocities to Cartesian velocities and angular velocities for the kinematic chain that starts from the root body of the avatar (here, the waist of the robot) and ends at the body touched by the haptic device. To get the haptic device force, we either need to subscribe to the dynamic simulator and listen for interactions between the haptic device and the environment or to directly query the haptic device manager. For our demonstration, we chose the first solution.

F. Results

Figure 2 shows the final architecture of the demonstrator, including components that are part of the framework and components that are specific to the demonstrated scenario. The relationship between the different components and modules is represented by arrows. The only components that we

need to write to implement this scenario are the controllers used by the avatar as well as the higher-level behavior encapsulated in the Avatar component. We ran the demonstration on a computer equipped with a PHANToM© Premium 1.5 High Force™. The robot indeed reacted according to the scenario. Figure 3 shows screenshots of the different states of the scenario. The translucid, light blue pyramid represents the field of view of the robot. The haptic probe is represented by the orange bar. The setup of the demonstration is shown on figure 4. Although most parts of the demonstrator use naive and simple implementations, it lays the basis for more complicated simulations where sensor noise for the haptic and visual perception can easily be integrated. A user who focuses on the haptic interaction during the task phase and does not want to start from scratch can use such a demonstrator and focus on the task behavior without caring about the other components, while other users can at the same time work on aspects related to vision. The integration of their own components to the demonstrator is easily realized if the users comply to the proposed interfaces. The overall amount of code for this simple, skeletic demonstrator is rather small and the development cost for the demonstrator was no more than a couple of person-hours (not including the haptic and visual perception modules which are provided to the user). The demonstrator is robust to changes in the modules on which it depends; the implementation of the dynamics simulator and collision detection modules used by this simulator would have no influence on the demonstrator's code. A user can relplace one of these modules by any other implementation by changing only two lines of the demonstrator's code. Any haptic device can be integrated to AMELIF; replacing the haptic device used in the demonstrator by another integrated haptic device would then require to change only two lines of code.

IV. Conclusion and future work

In this paper, we presented a framework called AMELIF, centered on interaction with virtual avatars. This framework provides integrative aspects by its flexibility and extensibility and allows fast and easy prototyping for algorithms related to dynamics, collision detection, haptic and visual rendering and low- and high-level interaction with virtual avatars. Various utility tools and components are also available. Several modules were developped under this framework which together form a solid basis for simulating virtual reality environments. Other modules, related to perception, are currently being integrated. In the near future, various control architectures will be integrated, including Operational Space Control, stack of tasks, optimal control, and others.

The integrated simulator has been used to demonstrate haptic interaction with virtual avatars. The virtual avatar was enhanced with low-level behaviors and a simple higher level behavior that switches low-level controllers according to events that occur during the simulation. The goal of this demonstrator was to show how this framework can be used to derive Virtual Reality scenarios involving different kinds of interactions with virtual avatars using different modalities, for example to prototype algorithms. In the future, we will use this framework to study and simulate higher level behaviors based on semantics that will be associated to patterns of interaction force between avatars and their environment. These behaviors will be based on an optimization of task sequences, allowing smoother and faster motions and thus more realistic haptic interaction with virtual avatars.

Acknowledgments

This work is partially supported by grants from the ImmerSence EU CEC project, Contract No. 27141 `www.immersence.info/` (FET-Presence) under FP6.

References

[1] J. Hodgins and W. Wooten, "Animating human athletes," *Robotics Research*, pp. 356–367, 1998.

[2] A. Shapiro, D. Chu, B. Allen, and P. Faloutsos, "The dynamic controller toolkit," in *The 2nd Annual ACM SIGGRAPH Sandbox Symposium on Videogames*, San Diego, CA, August 2007.

[3] S. Hasegawa, I. Toshiaki, N. Hashimoto, M. Salvati, H. Mitake, Y. Koike, and M. Sato, "Human-scale haptic interaction with a reactive virtual human in a real-time physics simulator," *Comput. Entertain.*, vol. 4, no. 3, p. 9, 2006.

[4] W. Son, K. Kim, and N. M. Amato, "A generalized framework for interactive dynamic simulation for multirigid bodies," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 34, no. 2, pp. 912–924, April 2004.

[5] D. C. Ruspini and O. Khatib, "Collision/contact models for dynamics simulation and haptic interaction," in *International Symposium of Robotics Research*, Snowbird, Utah, October 9-12 1999, pp. 185–195.

[6] L. Sentis, "Synthesis and control of whole-body behaviors in humanoid systems," Ph.D. dissertation, Stanford University, July 2007.

[7] H. Hirukawa, F.Kanehiro, and S.Kajita, "Openhrp: Open architecture humanoid robotics platform," in *Int. Symp. Robotics Research*, 2001.

[8] C. Duriez, F. Dubois, A. Kheddar, and C. Andriot, "Realistic haptic rendering of interacting deformable objects in virtual environments," *IEEE Transactions on Vizualization and Computer Graphics*, vol. 12, no. 1, pp. 36–47, January-February 2006.

[9] J.-R. Chardonnet, S. Miossec, A. Kheddar, H. Arisumi, H. Hirukawa, F. Pierrot, and K. Yokoi, "Dynamic simulator for humanoids using constraint-based method with static friction," in *IEEE International Conference on Robotics and Biomimetics*, Kunming, China, December 17-20 2006, pp. 1366–1371.

[10] F. Keith, P. Evrard, J.-R. Chardonnet, S. Miossec, and A. Kheddar, "Haptic interaction with virtual avatars," in *Proceedings of the Euro-Haptics (Madrid June 9-13)*, 2008, pp. 630–639.

[11] J. Smart and K. Hock, *Cross-Platform GUI Programming with wxWidgets*, 2005. [Online]. Available: http://www.wxwidgets.org

[12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.