Choreonoid as a Software Framework for Implementing Graphical Robotics Applications

*Shin'ichiro Nakaoka (AIST)

1. Introduction

Choreonoid[1] is a graphical software tool which was originally developed for the purpose of choreographing motions of humanoid robots[1]. Fig.1 shows a screenshot of it. Choreonoid allows a user to create dynamically stable whole body motions of biped humanoid robots with operations like key frame editing of CG character animations [2]. In addition to that, the base part of Choreonoid is designed to be a general framework for integrating various algorithms or functions into a powerful graphical user interface. This feature can make Choreonoid more useful robotics software tool. This presentation introduces this aspect of Choreonoid. We call it "Choreonoid framework" when we distinguish it from the whole Choreonoid package including the choreography interface.

2. Design Goals

Key points of our design goals of the Choreonoid framework are as follows:

1. To cover a high-level, rich GUI framework
2. To allow a developed application to make full use of PC's functions and performances not only for internal computations but also for their visualizations and interactions with user operations
3. To provide a lot of flexibility in implementing and extending functions on the framework
4. To be available for many people

There are many libraries and frameworks which are popularly used in many robotics programs, but few of them sufficiently cover the GUI part. An important role of our framework is to complement this area of robotics software tools.

The second point would not necessarily be important for every applications, but it is necessary for our original purpose of implementing choreography interface. It has to render complex 3D robot models and do a heavy computation of the dynamic balance adjustment while directly interacting with user operations. The quality of such an interface highly depends on the second point, and there will be other robotics applications which require it.

The third point is also important to robotics applications which consist of a lot of modules cooperating with each other. High-level robotics tasks often require such systems and the framework should be able to handle them.

For the fourth point, depending on special systems or expensive systems should be avoided, and the portability of the whole system should be increased.

[1]The name "Choreonoid" is a combination of "choreograph" and "humanoid," and expresses the original motivation to develop this tool.

3. Basic Design of the Framework

To achieve the goals described above, we designed the Choreonoid framework as follows.

3·1 Languages and Processes

To achieve the second point in the design goals, the framework employs C++ as a main programming language, and all modules built on the framework are basically executed in a single runtime process of the operating system. This kind of implementation is not special at all, but separating a system into back-end processes and front-end processes and applying the different languages to them is also popular in robotics systems. For example, robot simulator OpenHRP[3] consists of back-end CORBA servers implemented with C++ or Java and a front-end GUI implemented with JAVA. In contrast to such a system, our framework intends to be a more monolithic, straightforward system with minimum overheads so that it can be more reasonable for implementing applications like our choreography interface.

In fact scripting languages are useful for implementing a part where the maximum hardware performance is not required. In order to achieve this, the framework embeds the Python interpreter and provides Python wrappings for major objects in the framework. This function is still under development, but we think it will be an important function of the framework.

3·2 MVC-like Signal-based Processing

One of the important factors to achieve the third point in the design goals is an object processing style like so-called "Model-View-Controller(MVC)".

The key technique to achieve this processing style is "Signal", which flexibly connects various actions of objects. Objects in the framework can have arbitrary set of signal members and each signal can have multiple connections to any actions (functions) of any objects. A particular state change or event emits its corresponding signal and the emission of the signal causes its connected actions. This technique can achieve "Observer Pattern" in a flexible manner.

In this style, objects dealing with internal model can be independent from their visualizations or user input handling, and the objects can concentrate on their own work. Similarly, objects dealing with the

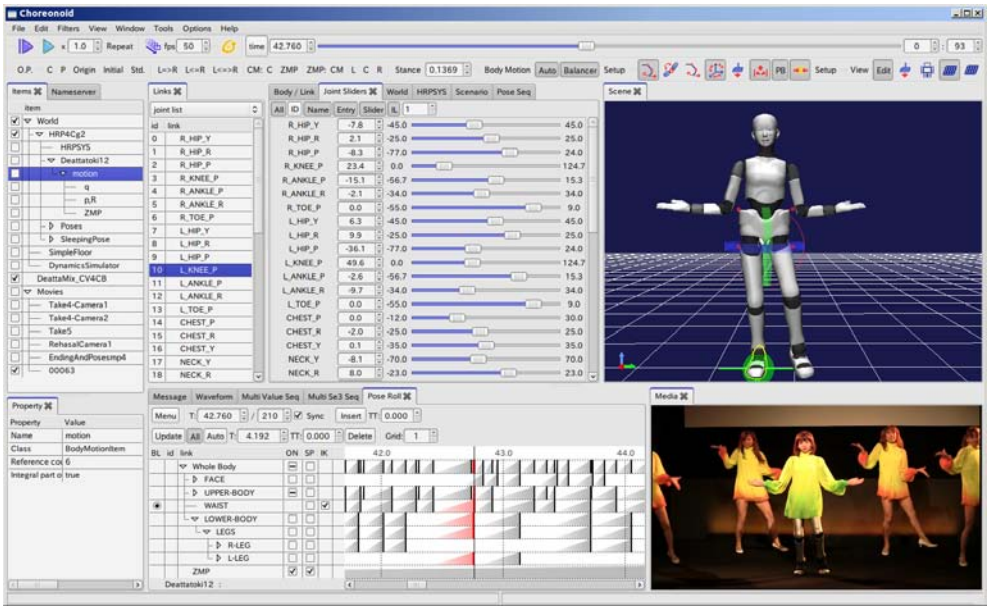

Fig.1 A screenshot of Choreonoid

visualization can concentrate on their own work. All they have to do is updating the visualization when related signals are emitted. Objects dealing with user inputs just control model objects and they do not have to consider other objects which handles the visualization or user inputs. This mechanism allows object to cooperate with other related objects while keeping the independence in implementation. This plays an important role in achieving the third point in the design goals.

3·3 Tree Structure Managing Various Model Objects

In the Choreonoid framework, model objects explicitly handled by a user is called "Items", and items are managed in a tree structure which is explicitly shown on the GUI component called "ItemTreeView". In the item tree, various types of items are simultaneously managed and item positions including parent-child relationships can be freely set by a user. The relationships are not only a visual factor but also a factor to determine relationships in actual processing of the items. This design can make the management of items simple and flexible. Additional functions which handle new combinations of item types can be easily implemented and integrated on the framework. This tree-based management is also an important factor for the third point of design goals.

3·4 View System

The Choreonoid framework provides the view system, which integrates a lot of GUI window panels into a single main window. Each window panel is called a "View", and views are placed on the main window with tiled separations and tabbed overlaps. The placement can be freely modified to be appropriate for operations of the moment. This kind of window management is similar to that of integrated development environments (IDE) such as Visual Studio and Eclipse. With this system, a lot of GUI components can be easily managed by a user and a developer can concentrate on implementing each GUI component.

3·5 Plugin System

In this framework, a set of additional functions are implemented as a plugin module. A plugin can use not only functions provided by the framework but also functions provided by other plugins. In addition, a function of a plugin can be implemented so that it can be later extended by other plugins. These are achieved by handling plugins just as usual shared libraries (DLLs) for other plugins. The drawback of this manner is that version up of a plugin can easily cause a situation where other plugins depending on it must be recompiled, because public ABIs easily change in C++ unless developers are very careful about them. Choreonoid however employs this manner in order to obtain the highest flexibility of the plugin system.

3·6 Libraries Used in the Framework

The framework is built on the following libraries:

Qt A cross-platform GUI framework library. It supports popular operating systems including Windows, Linux and Mac OS X. The GUI system of the Choreonoid framework is based on Qt and higher-level functions are added on it. Qt also provides non-GUI functions such as thread management, and some of them are also used in the Choreonoid framework to increase the portability.

OpenSceneGraph A library for rendering 3D scenes with high-level scene graph processing. It

is built on the standard OpenGL API and works on any operating systems providing OpenGL. This library is used in the 3D scene rendering system of the Choreonoid framework.

Eigen A vector / matrix library. This library has a lot of advantages over other vector / matrix libraries. Most of vector /matrix computations in the Choreonoid framework are done using this library.

Boost C++ libraries Collection of many useful C++ libraries which compliment the standard C++ library. Bind, Function, Signals and Smart Ptr of Boost libraries play an important role in the framework design, and many other Boost libraries are also used in the framework.

All these libraries are open-source ones and each license allows users to control the license of their own programs using (linking) the library. All these libraries are also portable. This helps the framework to work on both Linux and Windows, and we are planning to support Mac OS X, too. The libraries greatly contribute to achieve our goals of the framework design.

4. Application Examples

4·1 Actual Robot Playback System

The HRP humanoid robots (HRP-2, HRP-3, HRP-4C, etc.) have a modularized control system called "Hrpsys"[4], which can be remotely controlled with CORBA or RTM[5] communications. To control actual robots from Choreonoid, we developed "HrpsysPlugin", which is a Choreonoid plugin which processes remote commands of Hrpsys. Using this plugin, motion data managed on Choreonoid can be easily executed on an actual robot just by operating the motion playback on Choreonoid [6]. In this way, extending existing functions to cooperate with other systems can be achieved by writing a plugin.

4·2 Humanoid Choreography System

As described in section 1, we originally began to develop Choreonoid to realize the interface for choreographing whole body motions of biped humanoid robots. Now the key part of the choreography system is implemented as a plugin called "ChoreographyPlugin". Thus the choreography system can be considered as one of applications built on the Choreonoid framework.

The main functions added by ChoreographyPlugin are key pose interpolation with dynamic balance adjustment and a view for editing key pose sequences (PoseRollView). The framework in itself provides the functions for editing poses of a robot model with a rich set of graphical user interfaces. PoseRollView cooperates with those functions so that a user can efficiently create a sequence of various key poses. Every time key poses are added, modified or removed, the interpolation system immediately adjusts the key poses and their interpolated motions so that the resulting motion can be kinematically and dynamically stable[2]. Cooperating with HrpsysPlugin, created motions can be directly executed on the actual robot with the synchronization with the motion of the virtual robot on Choreonoid.

Fig.2 Dance demonstration performed by biped humanoid robot HRP-4C [7]. The motion of the robot was created using Choreonoid.

This system will enable humanoid robots to be used as a kind of "digital content technology" like CG characters. Toward this goal, we are making an attempt to produce attractive humanoid contents in cooperation with professional creators [7]. Fig.2 shows one of those contents created with Choreonoid. In this demonstration, biped humanoid robot HRP-4C[8] sings and dances with human back dancers. The whole body motion data of the robot was created using Choreonoid with ChoreographyPlugin, and the created demonstration was highly evaluated in the world. This means the Choreonoid framework has an potential to build practical, full-fledged robotics applications.

5. Future Work

First of all, we will release Choreonoid to the public with an open-source license. Then we will continue to improve Choreonoid to be widely used software framework for implementing graphical robotics applications.

6. Acknowledgments

A part of this work is funded by NEDO Intelligent RT Software Project.

References

[1] "AIST press release: Development of integrated software to generate humanoid robot motion easily" (2010). http://www.aist.go.jp/aist_e/latest_research/2010/20101105/20101105.html.

[2] S. Nakaoka, S. Kajita and K. Yokoi: "Intuitive and flexible user interface for creating whole body motions of biped humanoid robots", Proceedings of the 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, Taipei, Taiwan, pp. 1675–1682 (2010).

[3] F. Kanehiro, H. Hirukawa and S. Kajita: “OpenHRP: Open architecture humanoid robotics platform”, International Journal of Robotics Research, **23**, 2, pp. 155–165 (2004).
[4] F. Kanehiro, K. Fujiwara, K. Harada, K. Kaneko, S. Kajita, K. Yokoi and H. Hirukawa: “Motion control system of HRP-2 -plugins and their execution control (in Japanese)”, Proceedings of 21th Annual Conference of the Rbotics Society of Japan, Japan (2003).
[5] N. Ando, S. Kurihara, G. Biggs, T. Sakamoto and H. Nakamoto: “Software deployment infrastructure for component based RT-systems”, Journal of Robotics and Mechatronics, **23**, 3, pp. 350–359 (2011).
[6] S. Nakaoka, F. Kanehiro, K. Miura, M. Morisawa, K. Fujiwara, K. Kaneko, S. Kajita and H. Hirukawa: “Creating facial motions of Cybernetic Human HRP-4C”, Proceedings of the 10th IEEE-RAS International Conference on Humanoid Robots, Paris, France, pp. 561–567 (2009).
[7] S. Nakaoka, K. Miura, M. Morisawa, F. Kanehiro, K. Kaneko, S. Kajita and K. Yokoi: “Toward the use of humanoid robots as assemblies of content technologies - realization of a biped humanoid robot allowing content creators to produce various expressions -”, Synthesiology, **4**, 2, pp. 80–91 (2011).
[8] K. Kaneko, F. Kanehiro, M. Morisawa, K. Miura, S. Nakaoka and S. Kajita: “Cybernetic Human HRP-4C”, IEEE-RAS International Conference on Humanoid Robots, Paris, France (2009).