Scopira: An Open Source C++ Framework for Biomedical Data Analysis Applications – A Research Project Report

Aleksander B. Demko
Institute for Biodiagnostics
National Research Council Canada
435 Ellice Avenue
Winnipeg MB R3B 1Y6
Aleksander.Demko@nrc-cnrc.gc.ca

Rodrigo A. Vivanco
Institute for Biodiagnostics
National Research Council Canada
435 Ellice Avenue
Winnipeg MB R3B 1Y6
Rodrigo.Vivanco@nrc-cnrc.gc.ca

Nick J. Pizzi
Institute for Biodiagnostics
National Research Council Canada
435 Ellice Avenue
Winnipeg MB R3B 1Y6
Nicolino.Pizzi@nrc-cnrc.gc.ca

ABSTRACT

In MRI research labs, algorithms are typically implemented in MATLAB or IDL. If performance is an issue they are ported to C and integrated with interpreted systems, not fully utilizing object-oriented software development. This paper presents Scopira, an open source C++ framework suitable for MRI data analysis and visualization.

Classifications: D.3.2 C++, D.2.13 Reusable libraries

General Terms: Design, Languages

1. INTRODUCTION

The driving force for this project was to develop a comprehensive, object-oriented programming architecture using C++ for the development of applications geared towards exploratory data analysis of magnetic resonance images (MRI). This approach strikes a balance between slow, interpreted languages such as IDL and MATLAB and fast, compiled languages such as C. As application frameworks, the lack of object-oriented development and the difficulty in integrating C/C++ within these systems does not make them ideal platforms for complex software development.

The emphasis with Scopira [1] has been on high performance, open source development and the ability to easily integrate other C/C++ libraries used in the medical imaging field by providing a common OOP API for applications. As some analysis algorithms lend themselves to parallelization, and with the advent of Beowulf clusters, Scopira has been augmented to take advantage of MPI [2] for distributed and parallel computing. A dedicated, Scopira-specific parallel execution is currently under development. Unlike other parallel programming interfaces such as MPI and PVM, Scopira's facilities provide an object-centric approach with support for common parallel programming patterns and approaches.

2. N-DIMENSIONAL DATA ARRAYS

Scopira provides an n-dimensional template class ***narray***, and its helper templates ***nslice*** and ***nindex***. The ***narray*** class is applicable to any numerical data type of any dimension (vectors, matrices, cubes, etc). The resulting class is then tuned for that particular type, and has the same performance characteristics as a native C-language array of the same type. Scopira also provides bounds checking (via ***asserts()***) when in debug builds, which are removed on release builds resulting in performance that would be no worse than the C implementation equivalent .

The ***nslice*** template class is a virtual n-dimensional array that is simply a *reference* to an ***narray***. The class only contains dimension specification information and is easily copyable and passable as function parameters. Element access translates directly to element accesses in the host ***narray***. An ***nslice*** must always be of the same numerical type as its host ***narray***, but can have any dimensionality less than or equal to the host. This flexibility is very powerful; one could have a one-dimensional vector slice from a matrix, cube or five-dimensional array, for example.

The ***narray*** class provides hooks for alternate memory allocation systems. One such system is the *DirectIO* mapping system. Using the memory mapping facilities of the operating system (typically via the ***mmap*** function on POSIX systems), a disk file may be *mapped* into memory. When this memory space is accessed, the pages of the files are loaded into memory transparently. Writes to the memory region will result in writes to the file. Furthermore, as the ***narray*** class is 64-bit clean, on 64-bit architectures very large files may be used as datasets and the operating system will page portions of the file into memory as needed.

3. TOOLS SUBSYSTEM

Scopira consists of modular subsystems that can be used as needed by developers. The tools subsystem provides generic facilities useful in many programming domains, not just numerical and scientific computing. A reference counting scheme provides the basis for memory management. Scopira implements a template class that emulates standard pointer semantics while providing implicit reference counting on any target object. Other useful utilities such as threading/locking, random number generation, XML file processing and dynamic library loading are provided.

Scopira provides a flexible, polymorphic and layered input/output system. Flow objects may be linked dynamically to form I/O streams. Scopira includes *end flow* objects, which terminate or initiate a data flow for standard files, network sockets and memory buffers. *Transform flow* objects simply translate data from one form to another, such as binary-to-hex, buffer

OOPSLA '05, October 16–20, 2004, San Diego, California, USA.

consolidating and ASCII encoding. Future transformers could include CRC calculators, compressors and cryptographic ciphers.

Serialization flow objects provide an interface for objects to encode their data into a persistent stream. Through this interface large complex objects can quickly and easily encode themselves to disk or over a network. Upon reconstruction, the serialization system re-instantiates objects from type information stored in the stream. Shared objects – objects that have multiple references – are serialized just once and properly linked to multiple references.

4. GRAPHICAL USER INTERFACE

This subsystem provides a basic graphical API wrapped around GTK+ [3] and consists of widget and window classes that become the foundation for all GUI widgets in Scopira. More specialized and complex widgets, particular useful to numerical computing and visualization are also provided. This includes widgets useful for the display of matrices, 2D images, bar plots and line plots. Developers can use the basic GUI components provided to create more complex viewers for a particular application domain.

A complementary subsystem provides the base OpenGL-enabled widget class that utilizes the GTKGLExt library [4]. The GTKGLExt library enables GTK+ based applications to utilize OpenGL for 2D and 3D visualization.

A *model* is defined as an object that contains data and is able to be *monitored* by zero or more *views*. A *view* is an object that is able to bind to and listen to a model. Typically, views are graphically in nature, but in Scopira non-graphical views are also possible. A *project* is a specialized model that may contain a collection of models and organize them in a hierarchical fashion. Full graphical Scopira applications are typically project-oriented, allowing the user to easily work with many data models in a collective manner. A basic project-based application framework is provided for developers to quickly build GUI applications using *models* and *views.*

Scopira also provides a *Lab* facility to rapidly prototype and implement algorithms that need casual graphical output. Users code their algorithm as per usual, and a background thread handles the updating of the graphical subsystem and event loop.

5. TYPE REGISTRATION SUBSYSTEM

This subsystem provides a general singleton registration object for registering serializable data types and through which applications and external plug-ins register their data models and views. At runtime, Scopira pairs the compatible models and views for presentation to the user. A collection of utility classes for the easy registration of typical objects types such as data models and views are provided. This registration mechanism succeeds regardless of how the code was loaded; be it as part of the application, as a linked code library or as an external plug-in.

Third parties can easily extend a Scopira application that utilizes models and views extensively. Third party developers need only register new views on the existing data models in an application, then load their plug-in along side the application to immediately add new functionality to the application. The open source C++ image processing and registration library ITK [5] has been successfully integrated into Scopira applications at run time using the registration subsystem.

6. MPI EXTENSION

This subsystem provides a set of ***narray*** aware template functions and input/output classes that allow developers to easily interface with the MPI programmer's API. Using C++ trait classes for type information and the size data already stored in ***narray***, these functions drastically reduce the amount of parameters needed from the programmer thereby reducing common mistakes when using MPI.

7. AGENTS EXTENSION

A Scopira-based parallel execution framework is currently being developed. This extension has several notable goals particularly useful to Scopira applications. The API is completely object-oriented. This includes using the flow system for messaging, task movement and check-pointing (supporting both primitive and basic data types a swell as serializable objects) as well as the registration system for task instantiation.

An *agent object* is launched whenever a Scopira application utilizes the agent API. This object manages the additional worker threads and network connections (to other agents) while the application runs. This is completely transparent to the programmer.

The mechanics and implementation of the agents and their load balancing system are built into the agents extension library, and thereby, Scopira applications. Users do not need to install additional software, nor do they need to explicitly configure or setup a parallel environment. This is paramount in making cluster and distributed computing accessible to the non-technical user, as is makes it a transparent feature in their graphical applications.

Scopira will contain three different agent types. The *local* agent is non-network aware, single machine (but multi-threaded) agent that may be used when distributed computing is not desired or unavailable. The *cluster* agent is used with dedicated, full connected and persistent *Beowulf*-like clusters are available. Load balancing and resource allocation decisions are done at the global level. Finally, a *decentralized* agent may be used to build larger, complex agent networks. This agent makes resource allocation decisions based on local information only (allowing for network scalability) and may be utilized over unstable network links to possibly unreliable remote agents. This is the most dynamic agent of the set, requiring many peer-to-peer like approaches to resource allocation and deployment.

8. APPLICATIONS/CONCLUSIONS

Scopira has been used to implement two MRI data analysis applications. EvIdent [6] uses data driven cluster analysis to identify brain activation regions with functional MRI. The other application, currently being developed, calculates cerebral blood flow and regions of edema to identify areas of ischemia in brain scans of experimental animal brain stroke models using MRI. The MPI subsystem was used to implement a parallel feature sub-selection pattern recognition algorithm on a Beowulf cluster.

9. ACKNOWLEDGEMET

The Natural Sciences and Engineering Council (NSERC) is gratefully acknowledged for its support of this investigation.

10. REFERENCES

[1] Scopira Website; http://scopira.org/

[2] MPI Website: http://www.mpi-forum.org/

[3] GTK+ Website: http://www.gtk.org/

[4] GTKGLExt Website: http://gtkglext.sourceforge.net/

[5] ITK Website: http://www.itk.org/

[6] EvIdent Website: http://scopira.org/evident/