General Collaboration Structures for Interactive Data Language Applications

Minjun Wang
EECS Department, Syracuse University, U.S.A
Community Grid Lab, Indiana University, U.S.A
501 N Morton, Suite 222, Bloomington IN 47404
minwang@indiana.edu

Geoffrey Fox
Community Grid Laboratory, Computer Science Department, School of Informatics and Physics Department, Indiana University, U.S.A
gcf@indiana.edu

Marlon Pierce
Community Grid Laboratory, Indiana University, U.S.A
501 N Morton, Suite 224, Bloomington IN 47404
mpierce@cs.indiana.edu

Abstract

Interactive Data Language (IDL) is an array-oriented data analysis and visualization application, which is widely used in research, commerce, and education.

It is meaningful to make end user IDL applications collaborative on events between computers over networks, using a common message broker as the underlying communication system, and deploy them in Peer-to-Peer Grid architecture.

In order to make a specific user application collaborative, we normally have to program on it, here and there throughout the whole package, mostly in places related to event handlers.

While this approach is workable, it seems not a general solution for every application.

We then propose two potential general solutions for solving the problem – the Dynamic Structure and the Embedded Structure. We specially focus on the Embedded Structure, analyzing the reasons, possible ways and points to conquer the problem, and the benefit out of it. We propose the "embedded collaboration object" concept for it.

1. Introduction

Interactive Data Language (IDL) is an array-oriented data analysis and visualization application, which is widely used in research, commerce, and education [1, 2]. Its application areas include engineering, medical physics, astronomical and space science, earth science, etc.

It offers rapid interactive data analysis and visualization, a programming environment, and support for end user applications.

IDL is available for Windows, UNIX, Linux, Macintosh and VMS platforms and Operating Systems. This high availability facilitates data analysis and visualization in multi-platform environment, and ensures high code portability among platforms and systems.

People from different categories around the world have developed and been using diverse IDL end user applications in their respective areas.

It is contributing and meaningful to make IDL end user applications (especially interface-intensive ones) collaborative between computers of same or different platforms, using a common message broker as the underlying communication system, and deploy them in Peer-to-Peer Grid [3, 4, and 5] architecture.

Normally such collaborative applications consist of a type of Master (or Master Client) and a type of Participant (or Participating Client) using small text event messages for the communication between them. During a session, the Master captures events in its process, deals with them, and sends the event messages to the participant for rendering the displays in the participant's process, so that both of them can share the screen displays simultaneously. There can be multiple participants working with the Master concurrently and independently [5, 6, and 7].

The places to develop the codes for collaboration are usually within the end user applications; i.e., we have to change, modify and add new codes to programs, here and there throughout the whole application, mostly where are related to the event handlers.

We have been working this way on a real IDL application package – ReviewPlus [8] from General Atomics (USA) [9], which is a general-purpose data visualization tool – and have developed most of it to be collaborative on a Polling structure [7, 10].

This solution to collaboration is just constrained to specific applications; we have to start the developing process all over again for each and every application. This means a general solution for all end user IDL applications is highly appreciated.

In this paper, we propose two potential general solutions: *Dynamic Structure* and *Embedded Structure*.

The Dynamic Structure is supposed to dynamically generate collaborative IDL codes from any standalone, event-related IDL applications; it takes as input standalone IDL application(s), and outputs either a Master client or a Participating client, or both, depending on the options the user chose before its execution of the process.

The Embedded Structure encapsulates all the collaboration factors and embeds in the end user IDL applications in the form of embedded objects; we need to modify some IDL system library routines and add new ones to include collaboration codes, deploy them with and embed the orchestrating ones in the end user IDL applications. This is more feasible, we will discuss the possibility and benefit out of it.

2. Previous Work

We have been working on a real life IDL application package – ReviewPlus from General Atomics (USA), which is a general-purpose data visualization tool – and have developed most of it to be collaborative on a Polling structure.

The result collaborative software consists of a type of Master (or Master Client) and a type of Participant (or Participating Client) using small text event messages in the communication between them.

We use a common message broker – NaradaBrokering Message Service [11, 12] – for the message communication.

During a session, the Master captures events in its process, deal with them, and send the event messages to the participant for rendering the displays in the participant's process, so that both of them can share the screen displays simultaneously. There can be multiple participants working with the Master concurrently and independently.

In the Polling structure, both the Master and Participant connect to NaradaBrokering Message Service. On the master side, it captures events in the event handlers, processes and sends the event messages to NaradaBrokering for broadcasting to participants. On the participant side, whenever the broker has an event message to broadcast, it updates the public variables in one of its interface, i.e., it modifies the event flag by increasing one, and puts the message in a synchronized message queue. The main loop of the participant is constantly polling on the event flag; as long as there are still messages in the queue, it modifies the event flag by decreasing one, removes a message from the head of the queue, and then renders the display according to the instructions of the message.

We use a Grid-based Collaboration Model in the design and development, as shown in Figure 1.

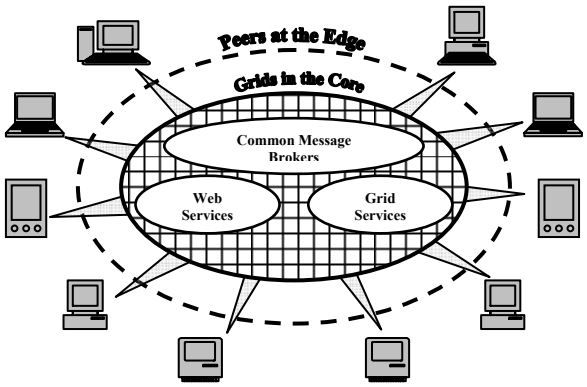

Figure 1. A grid-based collaboration model

In this model, there are two categories of computing – Grid computing and Peer-to-Peer computing.

Grid computing [3, 13, 14, and 15] is the basis; it largely comprises stable, formal, and efficient high-functionality services like Web Services, Grid Services, Common Message Brokers, etc., which are deployed as Grids on structured, well-organized and powerful supercomputers. They are in the core of the model.

Peer-to-Peer computing [16] is the interface to this world; it offers user-friendly, convenient, intuitive and easy accessible applications and services such as the popular commodity software used daily and everywhere. They are installed on a variety of personal devices, such as desktops, laptops, PDAs, smart phones, etc. They are at the edge of the model.

The infrastructure of Networks and the Internet ties up and correlates the two computing categories. It enables Peer-to-Peer Grids computing to be a trend, which harnesses the advantages of the two categories so that they complement each other, which also brings new opportunities and challenges to computing in all.

We realize the Peer-to-Peer Grids computing idea in this process. We deploy the Narada Message Broker as a Grid and use it for message communication between the Master and Participants of the

applications; and we deploy the Master and Participants as Peers at the edge and make them collaborate on events (Shared Event Model).

3. The Problem

The normal way to make a specific IDL application collaborative is to work on the programs of that application, delete, change, and add codes. On the master version of it, writing codes to catch events and send messages to the participant version of it for rendering. Events are caught in the event handlers of the Master and on the Participant the event structures are passed as parameters to the event handlers in the calls to them. However, in the overall programming, the IDL system routines and libraries are kept untouched; the abstraction is kept in the system.

For instance, we have been working this way on the specific IDL application package – ReviewPlus, and have made most part of it collaborative. All we have done is developing on this package and programming it here and there through the whole package, mainly in event handlers and places related to event handlers to achieve collaboration.

This solution proves to be workable and efficient, but for every different user application we have to repeat the whole developing process again and use the same or similar technologies and skills on that specific application. It is like reinventing the wheel or building another house using the same blueprint. It just costs unnecessary time and effort, and therefore it seems not a general solution for all end user IDL applications to be collaborative.

To solve this problem, we propose for it two potential general solutions – the Dynamic Structure and the Embedded Structure.

4. Dynamic Structure

While the Polling structure is feasible and practical for real life IDL applications to be collaborative, we keep asking a question:

Is there a general structure that can dynamically generate collaborative IDL codes from any standalone, event-related IDL applications?

Or, is it possible to develop a general application that generates collaborative IDL applications, taking a standalone IDL application or applications as its input, yet still integrating and using common message brokers like NaradaBrokering?

We would call this potential general structure Dynamic Structure, and the potential general application will use this structure in its implementation. This general application takes as input standalone IDL application(s), does lexical, syntax and semantic analyses, proceeds collaborative IDL code generation, and potentially code optimization. It would output either a Master client or a Participating client, or both, depending on the options the user chose before its execution of the process. This general application would be a specific compiler because it would go through the steps of the life-cycle of compilation theory [17].

This is illustrated in Figure 2.

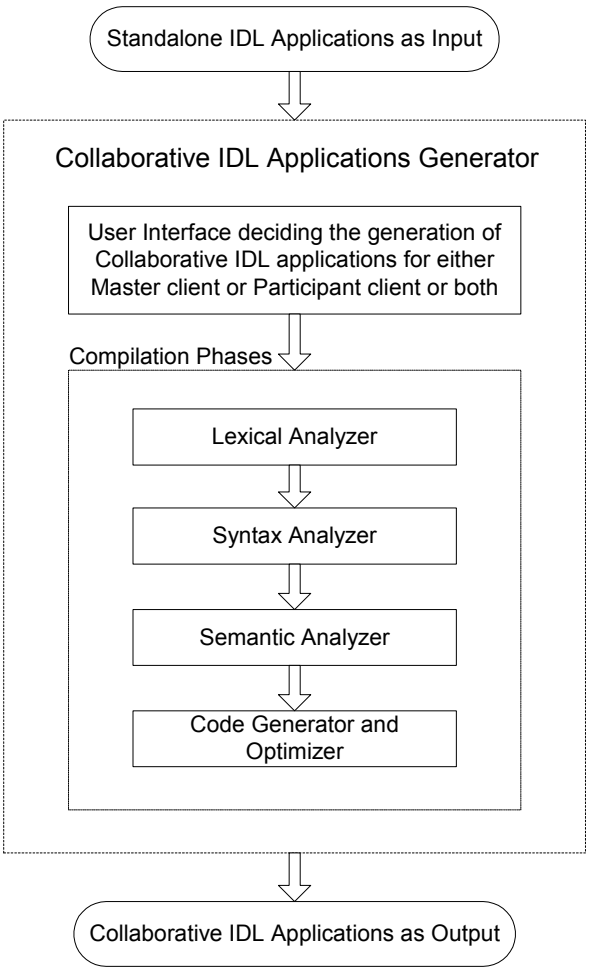

Figure 2. Dynamic structure for generating collaborative IDL applications

The Dynamic Structure is based on the Polling Structure; it is more innovative and a potential general solution for collaboration in IDL applications. If succeeded in implementation, it would mean great time, effort and cost savings, and efficiency and ability improvements.

This structure can be implemented on UNIX platform, using C/C++, lex and yacc in the

programming. Lex and yacc are tools in the aids of lexical and semantic analyses.

The difficulties and problems of this might come from that it might be too ambitious. Apart from the fact that it has to deal with all the cases and situations in the input applications, it also has to take into account the platform, system and environment. As we have experienced in the implementing of the collaborative ReviewPlus on the Polling structures, it is hard to wrestle with the system and environment, and the problems suddenly showing up were often unforeseen.

5. Embedded Structure

While it is possible for the dynamic structure to be a general solution for generating collaborative IDL applications from standalone ones, the complexity in doing that would be high and the effort would be huge, since compilers and operating systems always cost many people many years' smart-and-hard working and cooperation.

Is there an alternate solution that can achieve the generality of collaboration in IDL applications yet is much simpler and easier than the dynamic structure to save time, effort and cost?

After investigating the functionality of the IDL system and observing the routines from the system library, we realize that there could be such a chance. We address this discovery in the following sub-sections.

5.1. Potential Breakthroughs

To find a general solution for user desktop IDL applications to be collaborative globally, why not explore the opposite territory? Why don't we keep the user IDL applications "untouched" (or almost "untouched") and accommodate the IDL system routines and libraries to satisfy the end user IDL applications' collaboration needs?

More specifically, why can't we modify and develop the IDL system library routines such as "XMANAGER.pro", whose codes are accessible, and add new ones to mimic some system functions, whose codes are private? If this way works, we can just put our time and effort here once and for all, and expect the developed package would suit every end user application's need for global collaboration over the Internet.

How could this be possible?

To answer this question, let us begin with the IDL system library routine "XMANAGER.pro".

XMANAGER is written in IDL and its code is open in the library of the RSI [2] IDL commodity software; it provides the main event loop and management of widgets created in widget programs, and registers the widget programs with it. It then takes control of the event processing until all the widgets have been destroyed in a session.

XMANAGER is not called much and often; usually the statement appears once at the end of a widget program, like:

```
Xmanager, 'ReviewPlusSetup', self.wTLB, /no_block
```

It also orchestrates other system routines, including the function *WIDGET_EVENT,* which returns events for widget hierarchies.

It is possible for us to deal with all the event handling and processing in this routine and all other related system routines to achieve collaboration. In other words, we can develop our codes regarding collaboration issues within these system routines only, thus save the necessity of programming in the end user IDL applications and keep them untouched.

Specifically, we can modify and develop these IDL system routines, let the XMANAGER orchestrate the performance, and make a version for the Master client ("XMANAGER_MASTER.pro") and a version for the Participant client ("XMANAGER_PARTICIPANT.pro"). The Master part captures, processes, and dispatches events in message to a common broker, while the Participant part receives message from the broker, processes and renders the events.

Then we can deploy the two versions to the end user's IDL applications supposed to be Master and Participant respectively, and replace "XMANAGER.pro" with "XMANAGER_MASTER.pro" on Master and "XMANAGER_PARTICIPANT.pro" on Participant.

This implies the benefit described in the next sub-section.

5.2. The Benefit

Had we succeeded, all we have to do in any end user IDL applications is to deploy our routines with the applications, and just replace the string "Xmanager" with "Xmanager_Master" in the user programs on the Master client side, and with "Xmanager_Participant" on the Participant.

Usually in a simple widget program, only one "Xmanager" statement is called at the end. Suppose there is a huge IDL application in which there are a hundred calls for "Xmanager", all we have to do (or let some utility software do it) is to replace 100 strings for both Master and Participant. This effort is almost nothing.

5.3. Embedded Collaboration Object

Usually, such a general solution for achieving collaborative IDL applications from standalone ones should be a standalone software application like the dynamic structure we have mentioned in the previous section; but in this case, it is "embedded", the opposite of a standalone one. Just like the name "embedded operating system" is a trend nowadays, we can analogously name it as "embedded collaboration object."

6. Conclusion

In this paper we described our previous work about Grid-base collaboration on a specific end user IDL application package (ReveiwPlus) and the way to make it collaborative on events via message broker. We pointed out the limitation or problem with the implementation by working on specific IDL applications. We then proposed two potential general solutions for solving the problem – the Dynamic Structure and the Embedded Structure. We specially focused on the Embedded Structure, analyzing the reasons, possible ways to conquer the problem, and the benefit out of it. We proposed the "embedded collaboration object" concept for it. At this early stage of the research, much more interesting work needs to be done.

References

[1] Liam E. Gumley, *Practical IDL Programming*, Morgan Kaufmann Publishers, San Francisco, CA 94104-3205, USA, 2002.

[2] Research Systems Inc. http://www.rsinc.com/

[3] Fran Berman, Geoffrey Fox, and Tony Hey, *Grid Computing: Making the Global Infrastructure a Reality*, John Wiley & Sons Ltd, Chichester, West Sussex PO19 8SQ, England, 2003.
See http://www.grid2002.org

[4] Geoffrey Fox, Hasan Bulut, Kangseok Kim, Sung-Hoon Ko, Sangmi Lee, Sangyoon Oh, Shrideep Pallickara, Xiaohong Qiu, Ahmet Uyar, Minjun Wang, and Wenjun Wu, "Collaborative Web Services and Peer-to-Peer Grids", or in sxw, presented at 2003 Collaborative Technologies Symposium (CTS'03).

[5] Minjun Wang, Geoffrey Fox, and Shrideep Pallickara, "Demonstrations of Collaborative Web Services and Peer-to-Peer Grids", *Journal Of Digital Information Management*, June 2004, Volume 2, Issue 2, pp. 93-96.

[6] Minjun Wang and Geoffrey Fox, "Design of a Collaborative System" for Open Office, Proceedings of IASTED KSCE 2004 Conference, US Virgin Islands, November 2004.

[7] Minjun Wang, Geoffrey Fox and Marlon Pierce, "Grid-based Collaboration in Interactive Data Language Applications", Proceedings of IEEE International Conference on Information Technology, Las Vegas, Nevada, April 4-6, 2005.
http://grids.ucs.indiana.edu/ptliupages/publications/GridCollabIDL_ITCC2005.pdf

[8] ReviewPlus Data Visualization Software User Manual
http://web.gat.com/comp/analysis/uwpc/reviewplus/manual/

[9] General Atomics and Affiliated Companies http://www.ga.com/

[10] Minjun Wang, Geoffrey Fox and Marlon Pierce Instantiations of Shared Event Model in Grid-based Collaboration to be published.

[11] Geoffrey Fox, Shrideep Pallickara, and Xi Rao, "A Scalable Event Infrastructure for Peer to Peer Grids", proceedings of 2002 Java Grande/ISCOPE Conference, Seattle, November 2002, ACM Press, ISBN 1-58113-599-8, pp. 66-75.
http://grids.ucs.indiana.edu/ptliupages/publications/ScaleableEventArchForP2P.doc

[12] Shrideep Pallickara and Geoffrey Fox, "Efficient Matching Of Events in Distributed Middleware Systems", *Journal Of Digital Information Management*, June 2004, Volume 2, Issue 2, pp. 79-87.

[13] Ian Foster and Carl Kesselman, *The GRID: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, Inc., San Francisco, CA 94104-3205, USA, 1999.

[14] The Globus Alliance
http://www.globus.org

[15] Geoffrey Fox, "Grids of Grids of Simple Services", for CISE Magazine, July/August 2004.

[16] Andy Oram, *PEER-TO-PEER: Harnessing the Power of Disruptive Technologies*, O'Reilly & Associates, Inc., Sebastopol, CA 95472, USA, 2001.

[17] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman, *Compilers: Principles, Techniques, and Tools*, Addison-Wesley Publishing Company, USA, 1988.