Fermi National Accelerator Laboratory

FERMILAB-Conf-91/85

Object Oriented Design and Programming for Experiment Online Applications— Experiences with a Prototype Application

Gene Oleynik
Fermi National Accelerator Laboratory
P.O. Box 500
Batavia, Illinois 60510

March 1991

* Presented at Computing and High Energy Physics Conference, KEK, Tsukuba, Japan, March 10-15, 1991.

Object Oriented Design and Programming for Experiment Online Applications - Experiences with a Prototype Application

Online Support Department
Fermilab
P.O. Box 500
Batavia, Il 60510
Tel: 708-840-3921

Sponsored by DOE contract No. DE-AC02-76CH03000

The increase in the variety of computer platforms incorporated into online data acquisition systems at Fermilab compels consideration of how to best design and implement applications to be maintainable, reusable and portable. To this end we have evaluated the applicability of Object Oriented Design techniques, and Object Oriented Programming languages for online applications. We report on this evaluation.

We are designing a specific application which provides a framework for experimenters to access and display their raw data on UNIX workstations that form part of their distributed online data acquisition systems. We have chosen to implement this using the C++ OOP langauge. We report on our experiences in object oriented design and lessons learned which we will apply to future software development.

1 INTRODUCTION: WHAT ARE GOALS OF SOFTWARE DESIGN?

We are responsible for providing online and data acquisition software systems for experiments. In pursuit of this, we have always been committed to the formal practice of and adherence to software engineering methodologies in design and implementation. Additionally, in the current environment of rapidly evolving marketplace of high performance processors, experimenters will insist on using the best and newest platform and embedded systems available. We are committed to enhancing our ability to provide platform independent, portable software systems.

Clearly, the goals of software design methodologies are simply higher productivity and better quality software. For the design and implementation of our PAN-DA [1] software system we applied traditional techniques of understanding the requirements of the problem domain through structured analysis (SA) and structured design (SD) [2]. We can note that PAN-DA now comprises over 50 independent software packages.

The borderline between design and analysis is often fuzzy. However, how a software system is split up into programs and subroutines, and the exact mechanics of how information is passed between them are clearly design issues, not analysis issues.

Thus we have always followed the analysis phase by a formal, well defined, design phase, in some cases using formal structured design tools. These techniques and the traditional programming languages (e.g., Fortran, C) they support do not address issues of reusability and extensibility. Once identified, each piece of software designed with structured techniques is built up from scratch and is typically not able to take advantage of other packages, resulting in repetition of effort.

2 AN EXAMPLE OF "REUSABILITY" IN STRUCTURE DESIGN

Figure 1 is a structure chart of part of the PAN-DA system which provides a simple connection oriented protocol for control of front end embedded processors [3]. It specifies over 40 procedures and is complex since it specifies several concurrent threads of execution. It is worth noting that this "model" of the software implementation was revised 45 times and completed in six weeks. A working implementation was in use only two weeks afterwards.

Figure 1. PAN-DA Component Structure Chart

The topology of this diagram reflects good structured design technique. The upper half is tree-like, where modules are functionaly decomposed into component pieces. The broad middle is the heart of the design and is very specific to the problem domain. The topology of the lower part is a network and reflects that the components of the lower half are used by many mid-level components. These modules are reused within the problem domain.

SD methodology dictates that design of these lower leafs is done as a result of stepping down the problem domain from the top to the bottom. The definition of the lower modules is thus constrained to be specific to the particular problem domain being addressed. It becomes unlikely that many of these procedures can be used outside of the immediate project. The methodology just does not encourage this kind of reusability.

3 WHY WE ARE LOOKING TO USE OBJECT ORIENTED PROGRAMMING

An analysis of virtually all the subroutine and "functional" libraries of both VAXonline [4] and PAN-DA reveal certain commonalities. Sets of related procedures have internal data structures associated with them that are "hidden" from the user by making only a handle to the structure visible to the user interface. We have been associating a set of operations with a well defined set of attributes. But we have

had no help from the programming language and environment. Object Oriented Programming languages clearly hold out promise of relieving us of severe burdens of programming, debugging and maintenance of this type of software.

Our initial design specifications result in software with a cohesive internal structure and with loose coupling and dependencies between modules. However, we have found through bitter experience that when new, often unanticipated features are added, the cohesiveness begins to be lost and the coupling grows tighter. The result is software that is a victim of its own success and is ultimately more difficult to support.

A recent example of this sort in our experience is the tape logger program for VAXonline [5] It started out as a well designed and engineered piece of software for logging events to 9 track drives. Support for logging to Exabytes was then added in, soon followed by support for writing multiple fixed length files per drive to enable staging the tapes in 9 track chunks to the Fermilab Amdahls. The result of these changes - changes that were performed with good engineering practice - was a diffusion of the original crispness of the design boundaries. Changes to the software now have unforseen side effects, so that the software is unstable with respect to changes.

The languages themselves promote tight coupling and diffuse cohesion (e.g., FORTRAN with its common blocks, and C with its global variables). Part of the problem is ingrained in procedural languages since data must be explicitly passed between procedures, or stored in global structures. This provides procedures with inappropriate access to such data.

4 STARTING DOWN THE OBJECT ORIENTED ROAD

It was thus natural for us to be enthusiastic about the promise of Object Oriented Design (OOD), Object Oriented Programming languages (OOP), and methodologies. We followed the progress of Paul Kunz [6], from SLAC, and his tutorials, read the literature [7], and scanned the network news groups [8] for developments in our problem domain.

The Object Oriented paradigm addresses reusability from the top-down by building from high levels of abstraction to more concrete levels. The identification and design of the Classes as independent entities which tightly couple specific data (attributes) and actions (methods), with well defined and minimal global interface, promote reusability outside the scope of the problem domain.

OOP languages provide formal support for encapsulation and data hiding and promote crisp boundaries for abstractions. Inheritance from higher levels of abstraction (through addition of sub-classes), and the abstraction of data typing and support for overloading (multiple definitions of a single operator) provide a powerful mechanism for extensibility. This is achieved without disturbing the internal

program structure and without impacting the interface to the existing users of the program functions.

We undertook a first evaluation of available OOPs. Several constraints governed our choices. They range from the necessity of having to link to existing FORTRAN and C libraries (such as Hbook, and our event distribution packages) to an awareness that we must support a wide range of experiment clients at a minimal dollar cost to them. Our initial evaluation of Eiffel [9] showed that it had problems interfacing to FORTRAN and generated some poor compiled code. We have not yet received the newest version to evaluate. We are required to provide cross platform support for Unix applications that we develop since our clients have a variety of Unix workstations (SGi, Sun, Decstations and IBM/Rios). This ruled out the use of Objective C. Given the industry support for C++, we are using it as the language of implemention of our first Object Oriented application project. However, we intend to allocate resources to enable reimplementation of the project in Eiffel when the new version of the compiler arrives.

We have also investigated the availability of online tools for the OOD process. We have found, so far, that the industry is not quite up to speed. Our product search has found almost no cost effective OOD products that are deliverable now. Cadre [10] and IDE [11] promise delivery of tools in the next six months, we may try a standalone Mac based design product [12]. We will continue to expend effort in this area. We have found that the availability of online tools is essential to ensure successful committment to software engineering methodologies.

5 GOALS AND DEFINITION OF THE PILOT PROJECT, "COMMISSIONER"

We chose as our protoype pilot project in the Object Oriented paradigm an application to allow online monitoring of event data under Unix. We chose this for various reasons. We were dissatisfied with the modularity and extensibility of our previous implementation of such an application for VAXonline. A recent project for a large collider experiment (DO) showed the physicists' desire for an easy to use tool to manipulate pieces of events for simple statistics and online monitoring which could be linked with existing experiment online code [13]. Such an application integrated with our existing data acquisition systems, does not yet exist. The application requires definition of a source of event data, extracting and handling of user specified pieces of the events, defining and filling histograms for display, and output of events, histograms and plots.

It requires an interactive user interface to drive the application and also the ability to easily incorporate previously developed experiment analysis code. As a prototype, we are not aiming for something as complete or fully functional as Reason or PAW [14]; but we are developing the beginnings of a framework for the handling of online event streams in an extendible and experiment configurable fashion.

Our choice of pilot project was also guided by some conservatism. Object Oriented Programming Languages have the danger of generating code that may have more run time overhead than when the program design is done for traditional programming languages. We did not risk a project with severe real time execution constraints as our first venture into this new technique.

6 EXPERIENCES WITH OOD

We can only agree with all the learned journals that warn that unless one spends the resources up-front to come up to speed on the new problem domain view, the going is tough. We have insisted that anyone who participates in the design has to have read Bertrand Meyer,Booch and C++[7]. We sent individuals to classes. In spite of this, we have spent many hours coming to consensus on terminology and viewpoint, and understanding how to design and construct the classes and their interrelationships. It reminds us, indeed, of when we first embarked on SASD. Cerebral comprehension is not sufficient and people have to get a gut feeling for the concepts.

The literature and OOD course notes stress that one must attack any design from several different perspectives. Classes are initially identified from the top-down (e.g., event, histogram, plot). They are then refined and understood from the bottom up (e.g., events from camac, events from tcp/ip, single dimension histograms, additive plots, etc.). The design is further modified and enhanced through thinking from the OOD point of view (e.g., treating "lists of histograms" as a single entity to be handled in a precisely analogous way to a single histogram).

It is worth noting that we are using X Windows and Motif as the basis for the user interface for our future online application developments. This, it turns out, meshes extremely well with migration towards the Object Oriented methodology. Experience we are gaining designing and coding applications using these interfaces is helping us grow in our understanding in the overall OOD and OOP arena.

7 COMMISSIONER CLASS ARCHITECTURE

We include here the top level Commissioner Class Architecture. It illustrates the class relationships needed for the project. Figure 2 shows functional classes in Commissioner. Commissioner accepts buffers of data (events) from a source, analyzes the data and fills histograms. The experimenter can display events and plot histograms during the analysis. Concurrency is achieved by separating the control interface from the event analysis process, or Factory.

Figure 2. Commissioner Functional Classes

The ControlInterface uses ScreenTools and an Editor to create a Factory. Each Factory contains commands to get events from a source, analyze and histogram. The ControlInterface uses the Syntax Checker before the Factory is started. A Configuration DataBase exists for each Factory and contains parameters required by the Source, files of experimenters code to be included, and timeouts. After it is started, the Factory communicates with the ControlInterface. The experimenter can request Events via the EventHoldingArea.

Figure 3 is a sketch of the inheritance diagram for event sources in Comissioner. It illustrates how class abstraction can be used to promote extensibility. By selecting out the commonality of what a source is, a well defined framework is provided for adding new sources. The configuration "mixin" class is used to provide a common method for sources to define configurable parameters that can be set from a menu (or optionally from a file). The data classes, together with polymorphism (essentially treating all types of data classes as a generic data class), allow the configuration parameters to be treated in a data type independent fashion.

Figure 3. Source Class Inheritance Diagram

We are using HBOOK and HPLOT packages in our first implementation. Our C++ interface around these FORTRAN packages enables replacement of them with other packages at a later date.

8 CONCLUSIONS

We are well on the way to a practical OOP implementation of an application for online use, with a goal of having something for experimenters to use in the next 4 months. Already in the design process we can look ahead to how to re-use the software we are coding in extended applications. For example, we are committed to providing a "small-DA" for test stand applications in the next year. The object oriented design and implementation of Commisssioner is resulting in an easily extensible model. New front-end event acquisition sources can be added under the source classes, and new data logging functionality as extensions to sinks. Providing for multiple threads of events, partitions, and their analysis are treated as part of the same design.

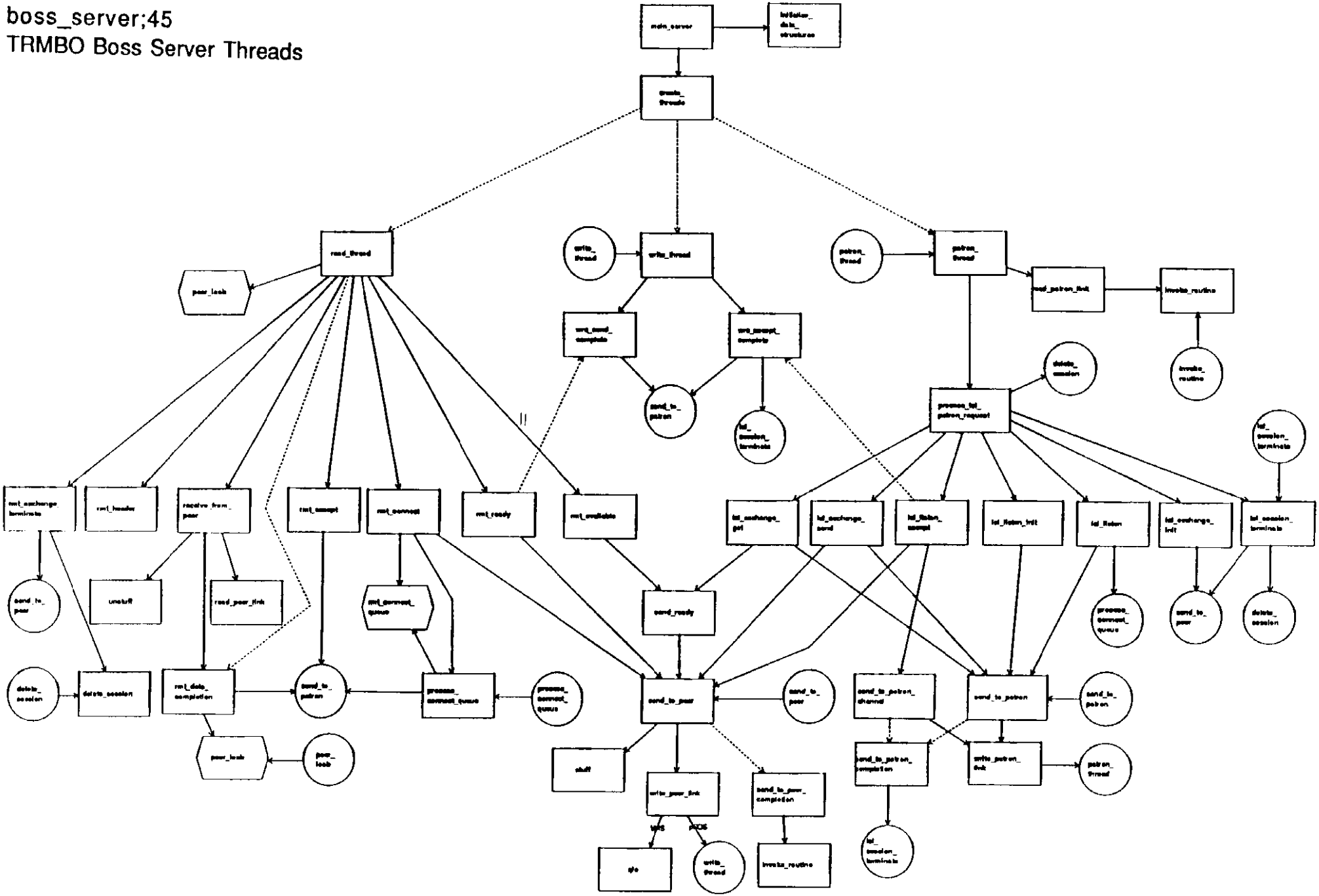
We are up to speed and testing the true applicability of this new paradigm in the online and data acquisition environment.

9 REFERENCES

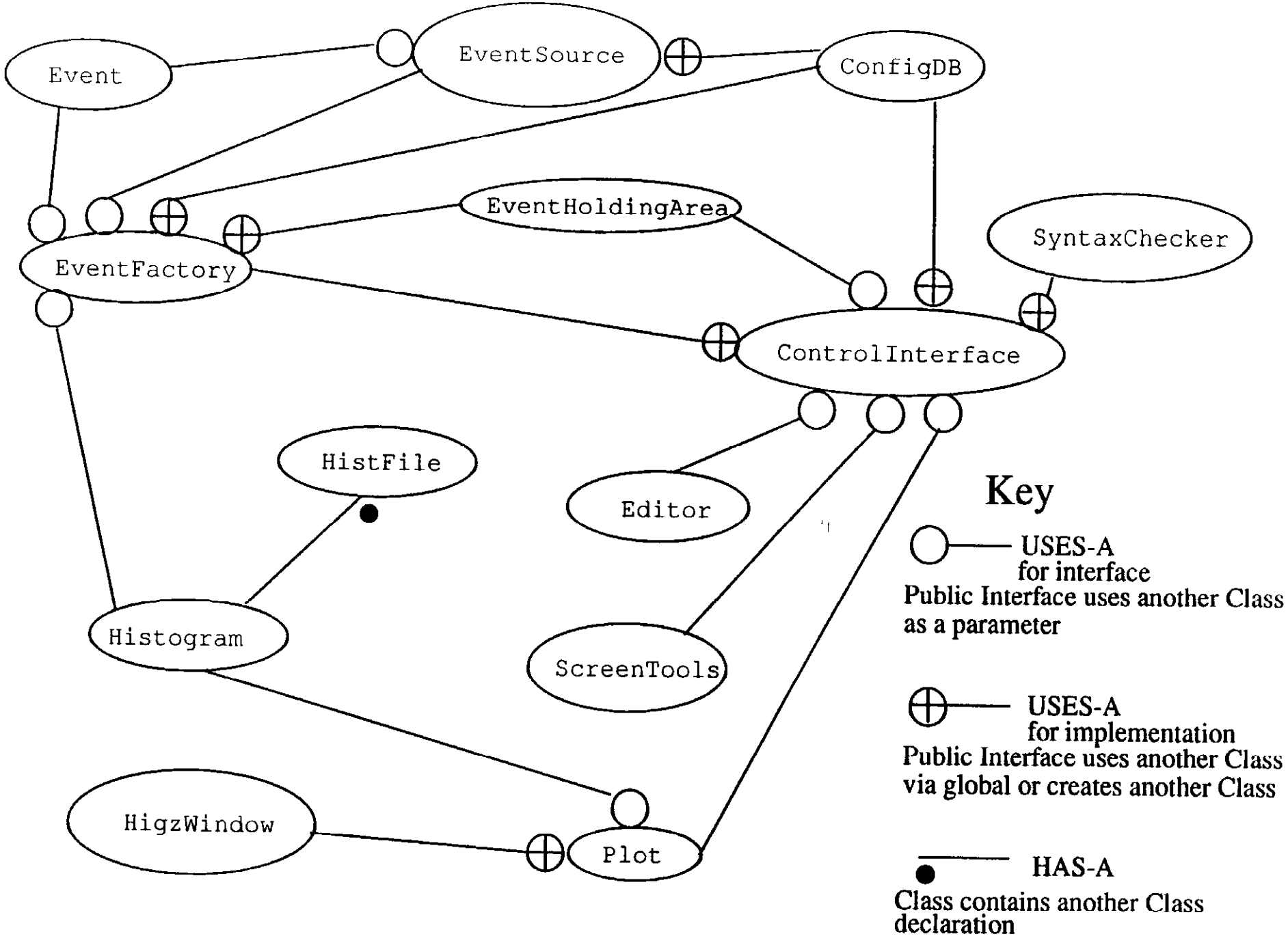
[1] PAN-DA, An integrated Distributed Data Acquisition System - D. Berg et al, P185.
[2] Structured Analysis and System Specification, Tom Demarco.

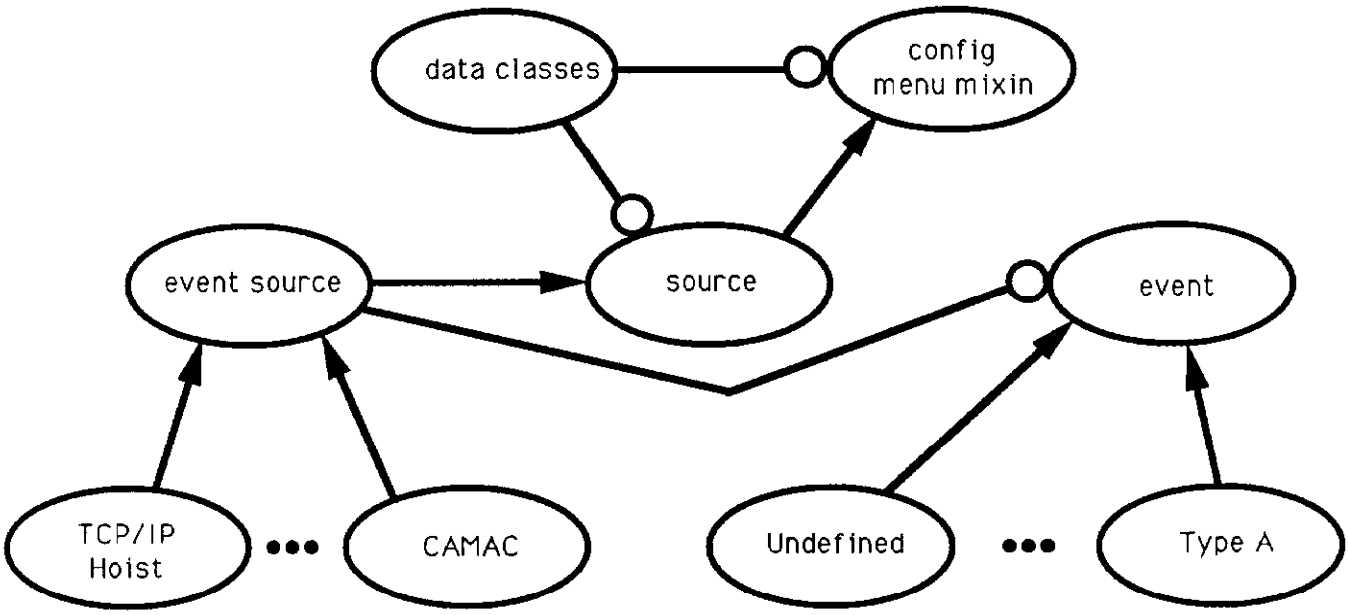
[3] TRMBD, B. Mackinnon et al, PN 400, Internal Fermilab Document
[4] Vaxonline, V. White, et al, "The VAXONLINE Software System at Fermilab" IEEE Transactions on Nuclear Science, Vol.NS-34, No.4, August 1987
[5] OUTPUT Component of VAXONLINE Software, Don Petravick, et al, Internal Fermilab document
[6] The Reason Project, R. Kunz et al, AIP Conference Proceedings 209, P290, CHEP 90, AIP Conference Proceedings 209, CHEP 1990.
[7] B. Meyer - Object-Oriented Software Construction ,Booch - Object Oriented Design with Applications, Ellis and Stroustrup - The annotated C++ manual, B. Eckel, Using C++,S. Shlaer, S. Mellor - Object Orientd Systems Analysis.
[8] general, C++ and Eiffel news groups
[9] TM, Interactive Software Engineering Inc, Goleta, Ca 93117
[10] Teamwork TM, Cadre Technologies Inc, RI 02903.
[11] Software Through Pictures, IDE, San Francisco.
[12] MacAnalyst, Macdesigner, Excel Software, IA 50158
[13] Aldis Users Guide, W. Bliss et al, PN437, Internal Fermilab document.
[14] PAW - Physics Analysis Workstation, R. Brun et al. Cern

boss_server;45
TRMBO Boss Server Threads



Commissioner Class Architecture





Source class

public:
- create
- configure mixin
- init
- get
- destroy

protected:
- cleanup
- state
- config table

Event class

- buffer
- event types
- etc.

configuration menu mixin class

static configured menu(file)
methods for parsing, handling files,
configuration table description

Configure
Source/Event
keyword description value

- source in file
- read from file
- use current