CLARA: A Contemporary Approach to Physics Data Processing

V Gyurjyan[1], D Abbott[1], J Gilfoyle[2], D Heddle[3], G Heyes[1], S Paul [3], C Timmer[1], D Weygand[1], E Wolin[1]
Thomas Jefferson national Accelerator Facility
12000 Jefferson Ave. Newport News, VA 23606

E-mail: gurjyan@jlab.org

Abstract. In traditional physics data processing (PDP) systems, data location is static and is accessed by analysis applications. In comparison, CLARA (CLAs12 Reconstruction and Analysis framework) is an environment where data processing algorithms filter continuously flowing data. In CLARA's domain of loosely coupled services, data is not stored, but rather flows from one service to another, mutating constantly along the way. Agents, performing event processing, can then subscribe to particular data/events at any stage of the data transformation, and make intricate decisions (e.g. particle ID) by correlating events from multiple, parallel data streams and/or services. This paper presents a PDP application development framework based on service oriented and event driven architectures. This system allows users to design (Java, C++, and Python languages are supported) and deploy data processing services, as well as dynamically compose PDP applications using available services. The PDP service bus provides a layer on top of a distributed pub-sub middleware implementation, which allows complex service composition and integration without writing code. Examples of service creation and deployment, along with Clas12 track reconstruction application design will be presented.

1. Introduction

Modern high energy and nuclear physics experiments require more and more computing power to keep up with continuously increasing experimental data volumes. The relatively short amount of time spent on experimental data acquisition requires a comparatively long time on data analysis. To complete quality physics data production, intellectual input from far-flung groups must be brought together. The physics data production and analysis process plays a central role in collaborations, and if done well, can raise both the quality and quantity of scientific output.

Physics data production in collaborative environments requires a specific computing model. The traditional computing model used in physics data analysis is based on self-contained, monolithic software applications running in batch mode. The inefficiency of using this model in a distributed environment, in terms of application deployment, maintenance, response to bugs, update propagation,

[1] Jefferson Lab
[2] University of Richmond
[3] Christopher Newport University

etc., is obvious. Other important limitations of the traditional model are scalability, heterogeneity and fault-tolerance.

In large, experimental physics collaborations it is impossible to enforce policies on computer hardware, requiring every group to buy specific hardware with a specific operating system. Collaborating groups would like to be able to use whatever computing resources they have at their home institutions, which evolve as new hardware is added. It becomes a real issue to deploy software applications in a heterogeneous environment. Remote collaborators must invest considerable manpower to update and rebuild software applications over and over again merely to retain compatibility with others.

There are two strategies that most experimental physics groups are adopting to overcome these problems: 1) a full centralization of the computational environment or, 2) implementation of a multitier grid computing infrastructure. Unfortunately there is a high price tag in maintaining and operating a full-scale computational grid. In any case, users generally look for location independent read/write access to data, as well as flexibility of the design, operation, maintenance and extension of PDP applications.

A significant challenge is that the PDP process usually has a very long life time, and therefore it is important to be able to upgrade its technologies. This entails that the PDP software applications be organized in a way that permits the discarding of aged components and the inclusion of new ones easily, without having to redesign and recode entire software packages at each change. It is a given that PDP applications will evolve over time. Inefficient and unsatisfying software modules will be dropped, and new ones will be added. Our experience shows that software evolution and diversification is important, resulting in more efficient and robust applications.

New generations of young physicists doing analyses of data may or may not have programming skills required for extending/modifying applications that were written while utilizing older technologies. For example, Java is the main educational programming language in many universities, but most of the data production software applications are written in C++ and some even in FORTRAN. We will now show that the CLARA framework is capable of providing an environment to develop agile, scalable, easily deployable, and maintainable PDP applications.

2. CLARA Framework

The CLARA framework aims to enhance the efficiency, agility, and productivity of PDP processes by making SOA [1] services the primary means through which data analysis logic is implemented. PDP applications, developed using the CLARA framework, consist of services, each running in a context that is agnostic to the global application logic. Services are loosely coupled and can participate in a multiple algorithmic compositions. Legacy processes or applications can be presented as services and integrated into a PDP application. Simple services can be linked together and presented as a single, complex, composite service. This framework provides federation of services, so that service-based PDP applications can be united while maintaining their individual autonomy and self-governance. It is important to mention that CLARA makes a clear separation between the service programmer and the PDP application designer. The physicist can be productive by designing and composing PDP applications using available and efficiently written services in the inventory without knowing service programming details. Services usually are long-lived and are maintained and operated by its owners on distributed CLARA containers. This approach provides an application designer the ability to modify them by incorporating different services in order to find optimal operational conditions, thus demonstrating the overall agility of the CLARA framework.

3. CLARA Services

This framework was designed based on a specific set of principles. The fundamental unit of CLARA-based PDP application logic is the service. Services exist as physically independent software programs with a common interface defined by the framework, as seen in table 1.

User applications, compliant to the required interface can be presented as CLARA services by using the service engine integration unite (SIU), illustrated in the figure 1. The SIU allows simple multithreading of user services.

Table 1. Service interface definition.

	Type	Definition
Name	String	Service name
Description	String	Short service description
Author	String	Service author
Version	String	Version of the service
Input data type	Object, primitives	Acceptable input data type
Output data type	Object, primitives	Generated output data type

Each service has its own set of capabilities. Those capabilities suitable for invoking by applications can be discovered via registration information available from the CLARA platform registry services. One design recommendation is to keep a small and simple service code base, which will help future service programmers to easily extend, modify, maintain and port services. Services must be abstract and agnostic to any PDP logic. Data centricity is an important requirement. Services must be discoverable and able to take part in complex service compositions.

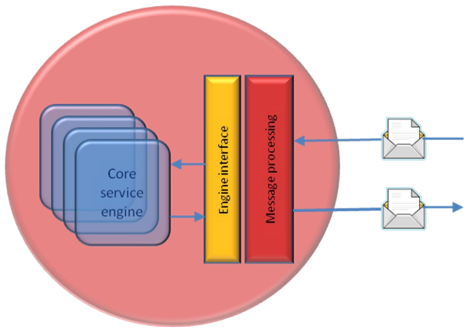

Figure 1. User service engine integration unit.

Two coupling modes exist between services and service consumers. Contract-to-functional coupling is used between CLARA services, making them bound to a contract according to which they must receive and send data. Each service itself can be a consumer. The second mode is one in which CLARA services are coupled to consumers (other services, software agents, etc.) by consumer-to-contract coupling, which is defined as an agreement of a service to trigger particular service engine execution after receiving an input data. Using only these two, data-in-data-out coupling contracts, services are able to abstract and encapsulate service programming details (programming languages, technologies, algorithmic solutions, etc.). Service function information is obtained through meta-data available as part of the contract. Service quality information can be obtained from the CLARA platform registration services.

3.1. CLARA service types and service inventory

Physics data analysis logic is implemented as a services or a service compositions, designed in accordance with CLARA service design principles. CLARA specifies four types of services: entity, utility, task and orchestrated task. Entity services are highly reusable and generic. They are atomic enough to take part in different service compositions. Users find many self-contained and legacy software systems very useful. These systems can be presented as utility services. Task and orchestrated task services are both composite services, with the only difference being that task services are self-governed, while orchestrated services are aggregated services controlled by the agents from the orchestration layer of the framework (see figure 2).

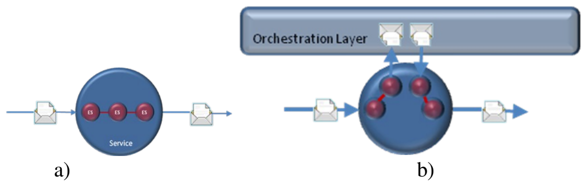

Figure 2. Service composition types: a) task service, and b) orchestrated task service.

A service composition is comprised of services that have been assembled to provide the functionality required to accomplish a specific task. CLARA distinguishes between two types of service compositions: primitive and complex. Primitive compositions use message exchange across two or more services. Complex compositions, however, require an orchestrator. Because the framework's requirement for services is to be agnostic to any PDP logic, one service may be invoked by multiple PDP applications, each of which can involve that same service in a different composition. A collection of entity services can form the basis of a CLARA service repository that can be independently administered within its own physical deployment environment. So, the CLARA framework helps to build services, service compositions, and service inventories. The service oriented approach of CLARA changes the overall complexion of a PDP application. Because the majority of services delivered are reusable resources agnostic to analysis, they do not belong to any one application. By dissolving boundaries between applications, the physics data production is increasingly represented by a growing body of services that exist within a continuously expanding service inventory.

4. CLARA Design Architecture

The CLARA architecture consists of four layers (see figure 3). The first layer is the PDP service bus. This layer provides an abstraction of the cMsg publish subscribe messaging system [2]. Every service or component from the event processing layer communicates via this bus which acts as a messaging tunnel between them. Such an approach has the advantage of reducing the number of point-to-point connections between services required to allow services to communicate in the distributed CLARA platform. By standardizing communication between services, adapting a PDP system to changes in one of its components becomes easier and simplifies the data transfer security implementation (for example by deploying a specialized access control service). The service layer houses inventory of the entity and composite/complex services used to build PDP applications. Administrative and registration services are also part of this layer. The administrative service is responsible for SIU service creation, deployment, recovery, cloning, and removal. The registration service stores information about every registered service in the service layer, including address, description and operational details. The orchestration of service-based physics data analyses applications is accomplished by the help of an application controller agent resident in the orchestration layer of the CLARA architecture. The application controller will continuously monitor the load of a particular service or composite service chain, request administrative services to clone a service, or restrict access to the service or chain. Agents from the physics complex event processing (PCEP) layer are designed to subscribe and analyze event data in real-time in order to generate immediate insight and enable instant response to changing conditions in the active, orchestrated service based PDP application. A PCEP agent looks at events in the context of other events rather than in isolation, and generates new (high level) events. Examples of PCEP agents include high level triggers, particle identification processes, and etc.

5. CLARA topology

A CLARA cloud contains multiple distributed platforms. Each CLARA platform itself contains

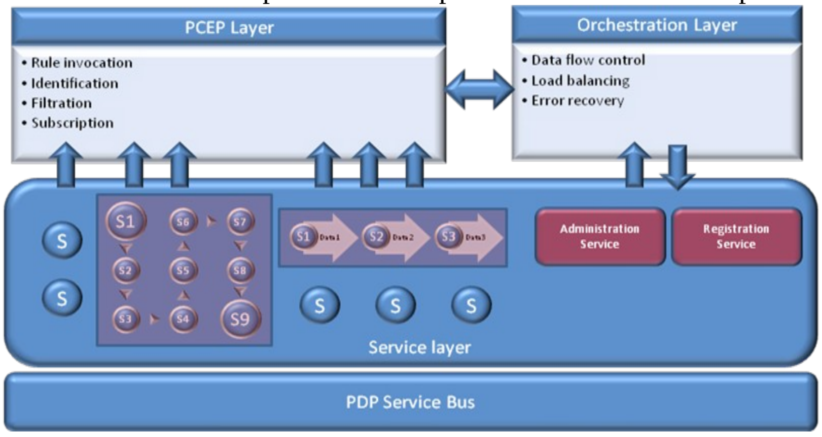

Figure 3. CLARA design architecture.

multiple CLARA containers. A container in reality is a Java Virtual Machine (JVM), providing a complete run time environment for CLARA SIUs executions, and allows several SIUs to concurrently execute in the same container. Each platform can be split among several hosts, and contains one front-end container and multiple distributed containers. The front-end container is the container where the platform main pub-sub server, administrative and registry services are running. Every non front-end container also contains a local pub-sun server, administration services for administering CLARA SIUs, and has monitoring services to monitor local CPU performance. CLARA C++ services run as standalone processes in their own run time environment.

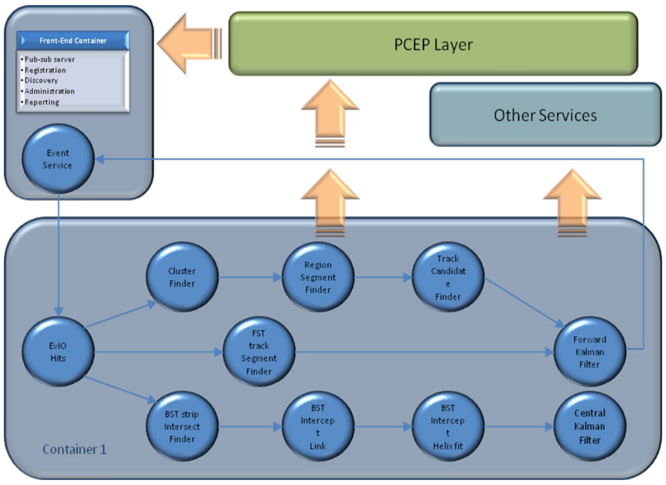

Figure 4. CLARA based SOT application diagram.

6. Implementation and performance

One example of an application that utilizes the CLARA framework is SOT (Service Oriented Tracking). SOT is tracking software for the Clas12 detector consisting of 11 services connected together in the CLARA framework (see figure 4). The input of SOT is an event encoded in EVIO format, which contains information about the hits in each of the Clas12 detectors.
The first of the services (Evio2Hits) takes this EVIO-encoded information and converts it into lists of hit objects. Hit objects are sent along to three different chains of task services within SOT - one chain for each of three detectors in Clas12 for which SOT does tracking. In the chain of services for the drift chamber tracking, for example, one service groups the hits into clusters, another links these clusters into region segments, and a third links the region-segments into track candidates.
The data transferred between services are in the form of lists of objects representing clusters, region segments, track candidates, etc. At the end of each of these three task services is a service called the Data Summary Service (DSS) whose purpose is to take these lists of internally used objects (such as tracks and clusters) and append them to the EVIO event that was initially sent to the Evio2Hits service. The output of DSS is an EVIO event containing all of the same information as the input event plus the reconstruction data.

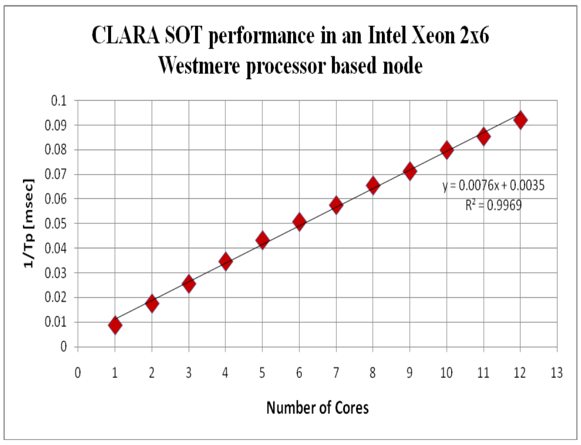

Figure 5. Composite service multithreading.

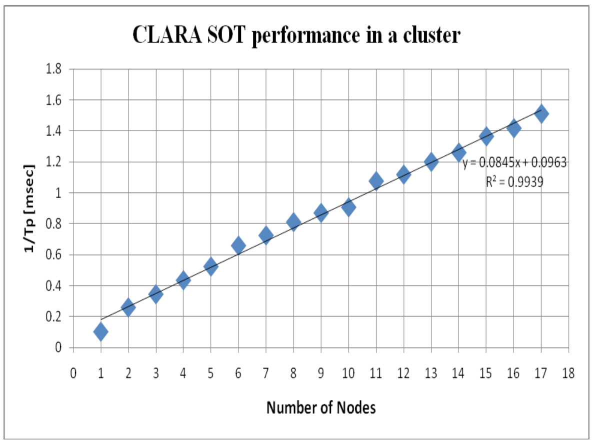

Figure 6. Distributed processing.

6.1. Performance

SOT was deployed in the CLARA platform running on an 18 node Xeon 2x6 Westmere CPU cluster. The row data set was selected to contain events having at least one charged track in the Clas12 detector. The measurements were conducted reconstructing one event at a time through 12 events at a time in the single CLARA container, utilizing all 12 cores of a node. The result of this measurement is illustrated in the Figure 5, showing a linear dependence between the average processing time and the number of cores. Figure 6 shows the performance of the SOT application distributed over 17 CLARA containers. Here we also obtained a linear scaling of the tracking performance.

7. Conclusion

An SOA-based physics data production application development framework was written in pure Java. This framework provides a clear separation between the PDP application designer and service programmer. It is a service development environment and standardized data exchange for increased interoperability of services. It also provides multithreading and multiprocessing totally transparent to users. The only requirement of the user is that the deployed service engines must be thread enabled and thread safe. The ease of service-based application development and deployment promotes application diversification and overall agility. We designed and deployed a CLARA service-based

Clas12 track reconstruction application, showing ~650µsec per event relative processing time on the CLARA platform running 18 node Xeon 2x6 Westmere cluster.

8. References

[1] Thomas Erl 2007 *SOA: Principles of Service Design* (Prentice Hall, ISBN: 0-13-234482-3)

[2] C Timmer, et al. cMsg – A G Purpose, Publish-Subscribe, Interprocess Communication Implementation and Framework., *Proceedings of the International Conference on Computing in High Energy and Nuclear Physics, Victoria BC, Canada 2007.*