Component Based Dataflow Processing Framework

V. Gyurjyan[1], A. Bartle[3], C. Lukashin[2], S. Mancilla[4], R. Oyarzun[4], A. Vakhnin[5]

[1] TJNAF, Newport News, VA

[2] NASA Langley Research Center, Hampton, VA

[3] Mechdyne, Co., Virginia Beach, VA

[4] Universidad Técnica Federico Santa María, Chile

[5] Science Systems and Applications Inc., Hampton, VA

Abstract—In this paper we present SOA based CLAs12 event Reconstruction and Analyses (CLARA) framework used to develop Earth Science multi-sensor data fusion, processing, and analytics applications (NAIADS: NASA JLAB collaboration). CLARA design focus is on two main traits: a) real-time data stream processing, and b) service oriented architecture (SOA) in a flow based programming (FBP) paradigm. Data driven and data centric architecture of CLARA presents an environment for developing agile, elastic, multilingual data processing applications. The CLARA framework presents solutions, capable of processing large volumes of data interactively and substantially faster than batch systems.

Keywords—service oriented; publish-subscribe, data-driven, data-flow, earth science, data fusion

I. Introduction

In this technologically complex world we live in today—where data are interconnected, information changes swiftly, and expertise is widely dispersed—we need a different approach to the software development. The phenomenon known as Moore's Law (doubling of computer processing speed every 18 months) is just one manifestation of the greater trend that all technological changes occur at an exponential rate. As a result we produce data a lot faster than we can process them. It is obvious that existing software applications are having a hard time keeping up with this unprecedented technological growth. Software applications are forced to evolve functionally in an ever growing technological environment, yet they keep old paradigms and structures, resulting in a structural deterioration and eventual termination. Software organizations and companies spend enormous amount of resources to maintain software applications with a design based on the old Von Neumann paradigm, which is not developed to take advantage of new technological trends, such as multi-core, multi-processor hardware systems. The majority of developed science data processing applications (SDP) are single, sequential processes that start at a point in time, and advance one step at a time until they are finished. In the current era of multi-core hardware architectures and distributed relational data sources this approach has noticeable limitations. Contemporary research [1,2] show that software programming paradigms and architectures, such as service oriented architecture (SOA) and flow based programming (FBP) [3] can withstand future technological pressures and ease the long term application maintenance. In this paper we present our take on developing data driven applications, using specialized data processing services in a FBP environment.

II. Component Based Approach

The CLARA framework uses SOA [4] to enhance the efficiency, agility, and productivity of data processing applications. CLARA is an approach to develop data processing applications based on the concept of multiple asynchronous processes (software building blocks), called services, communicating by means of data streams. Thus, an application is viewed as a system of data streams being transformed by services. Services are the primary means through which data processing logic is implemented. CLARA application services are running in a context that is agnostic to the global data processing application logic. Services are loosely coupled and can participate in multiple algorithmic compositions. Legacy processes or applications can also be presented as services and integrated into a data processing application. Services can be linked together and presented as one, complex, composite service. This framework provides a federation of services, so that service-based data processing 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 data processing application designer. An application designer can be productive by designing and composing (no programming involved) data processing applications using available, efficient and professionally written software services (building blocks, "black box" approach) without knowing service programming technical details. Services usually are long-lived and are maintained and operated by their owners on distributed CLARA service containers. This approach provides an application designer the ability to modify data processing applications by incorporating different services in order to find optimal data processing conditions, thus demonstrating the overall agility of the CLARA framework. CLARA is a multilingual framework that supports services written in Java, Python and C++ languages.

978-1-4799-9926-2/15/$31.00 ©2015 IEEE

A. Service Interractions

The CLARA framework provides developers with the means for interacting with services based on the publish-subscribe message exchanges. Message passing is the most popular communication model for distributed computing. It is the key for building SOA-based frameworks. This model is attractive due to the fact that messaging does not emulate the syntax of programming language function calls (unlike CORBA and RPC for example). Instead, structured data messages are passed between distributed components (i.e. services). In this distributed communication model, success largely depends on the design of the message structure: a communication envelope that describes not only transferred data, but also communication and service operational details. In order for a service communication to be truly useful, every service has to share/use the same vocabulary for expressing the communication details (i.e. common message-interface).

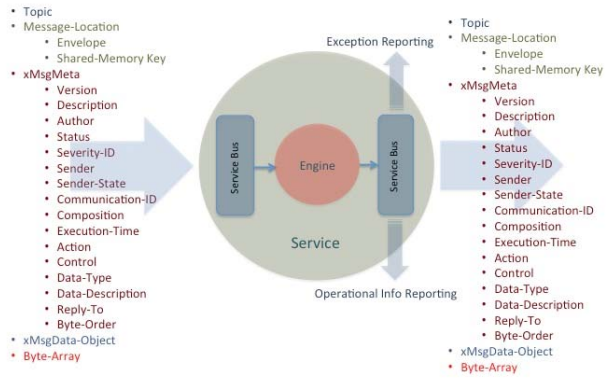

Figure1. CLARA data envelope. Envelope can have two or four part configuration based on the data transferred modes between local or remote services. Local: 1)Topic, 2)Shared-Memory-Key, and Remote: 1)Topic, 2)"envelope", 3)xMsgMetadata, 4)serialized object. Note that Topic is the address of a service.

CLARA transient data envelope is the main message interface between services. The mutual understanding and acceptance of this object couples the framework services. When we say CLARA services are loosely coupled we mean that this transient data object is the one and only physical coupling between CLARA services. The CLARA data envelope is illustrated in the Figure 1. The metadata segment of the transient data structure defines the data type, version, and the author of the transient data object. Even though we consider only event level parallelization for CLARA based data processing applications, we do not discard the possibility of having multi-tier services (services that are shared by multiple service based applications) in service compositions for building certain data processing applications. The *communication-ID* is designed to synchronize request/response pairs, and guarantee application specific data privacy. Even though the main objective of the CLARA service is to transform the input data at its output, it can also generate and deliver notifications to other services that register their interest in one or more data-events. This feature we found to be useful for exception handling and process monitoring. Service developer is unable to predict future customers (i.e. services that will be linked to it). Only a final data processing application designer knows the event/data flow outline. Rather than contacting services directly, the implicit invocation mechanism only signals that output-data is ready (a data is arrived at the service input) and it does not say what needs to be done to that data (how to react to that data). This clearly improves its maintainability, and it simplifies data processing application engineering processes.

III. Design Architecture

The CLARA architecture consists of three layers, illustrated in Figure 2. The first layer is the *service-bus* that provides an abstraction of the xMsg publish-subscribe messaging system [5]. xMsg is a messaging system that scales perfectly to tens of thousands of processes, running over many boxes with many cores as wanted. xMsg is using a ZeroMQ [6] socket library and implements messaging patterns like topic pub-sub, workload distribution, and request-response. Every service or component from the *orchestration-layer* communicates via the *service-bus*, which acts as a messaging tunnel between services. 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 cloud.

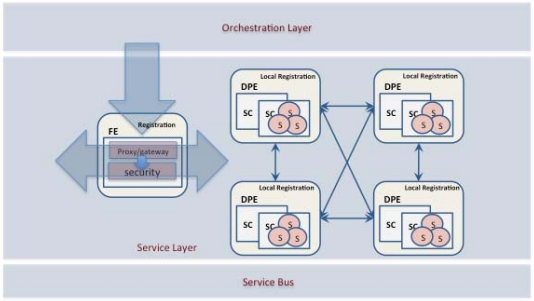

Figure 2. CLARA 3-architecture: Front-End DPE (FE). Data Processing Environment (DPE), Service Container (SC), Service(S).

The *service-layer* houses the inventory of services used to build data processing applications. An administrative registration service stores information about every registered service in the service layer, including address, description and operational details. The orchestration of data analyses applications is accomplished by the help of an application controller (application orchestrator), which resides in the *orchestration-layer* of the CLARA architecture. Front-End (FE) is a specialized data processing environment (DPE) that

houses a master registration service on top of the global registration database. The growing need to fuse, correlate and process widely distributed data in the CLARA environment will undoubtedly demand data security measures. The data and communication security is a critical requirement in the CLARA design. CLARA security does not only concern the security in the DPEs at each end of the data transfer chain (for e.g. running DPEs behind the firewalls). When transmitting data the communication channel should not be vulnerable to attack. Effort is thus invested to prevent data hijacking, followed by data decryption and re-insertion of the corrupted data. As a solution, CLARA provides normative data-proxy and security services that can be deployed in the FE. To insure secure data network a) authorized access, b) user authentication, c) data encryption and d) transiet data envelope integrity, are considered by security services.

A. Basic Components

The CLARA platform is composed of multiple distributed data processing environments (DPE). DPE provides a run-time environment for *services* and *service-containers* and allows several services to concurrently execute on the same environment. The DPE conceptual diagram is shown in the Figure 3.

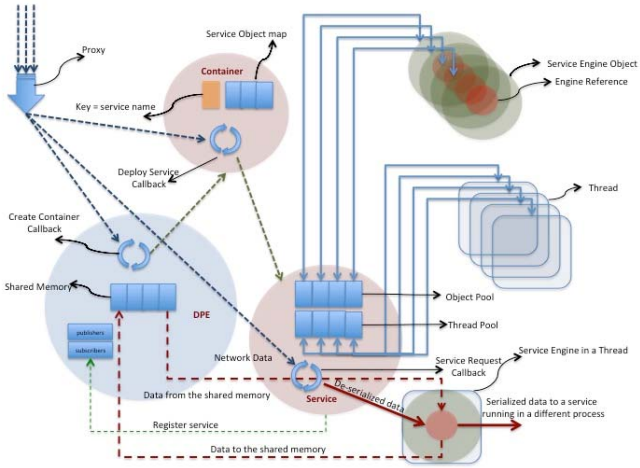

Figure 3. DPE diagram

The DPE hosts the registration service and the shared memory that is used by *service-containers* to communicate transient data between services within the same DPE. The use of shared memory prevents unnecessary copying of the data during service communications. The DPE runs one or many *service-containers* that are designed for logical groupings of services. *Service-containers* and *services* may also be distributed across multiple computing nodes for the purposes of scaling horizontally to handle increased data volume. CLARA application *orchestrator* can start *service-containers* in a specified DPE, and monitor and track functionality of *services* by subscribing to specific events from a *service-container*, reporting the number of requests to a specific service, as well as notifying when a successful execution of a particular service (or its failure) has occurred. Each *service-container* deploys one or many *services*, presenting their computing algorithms encapsulated in *service-engines*. CLARA *service* implements SOA SaaS as a way of delivering on-demand, ready-made data processing solutions (*service-engines*) as CLARA services. The CLARA data processing application user uses a *service*, but does not control the operating system, hardware or network infrastructure on which they are running. The quality of the data-processing application (including syntactic, semantic qualities, and performance) depends highly on the quality of constituent service, delegating a request to its engine. It is, therefore, critical to test and validate a *service-engine* before deploying it as a CLARA service. *Service-engines* must be validated with respect to workflow, thread-safety, integrity, reliability, scalability, availability, accuracy, testability and portability. The DPE also contains a proxy for service connections, eliminating direct connections between services. The use of the proxy solves service dynamic discovery problems. The DPE proxy acts like a simple stateless message switch (much like an HTTP proxy). Each *service-container* running in a DPE contains the map of deployed *service* objects. Every *service* object subscribes to service requests and processes the request by getting the *service-engine* object from the object pool, and executing the user provided engine in a separate thread.

B. Service Registration and Discovery

The core of the CLARA registration and discovery mechanism is the normative registration service that the CLARA services and service containers are registering with. The registration service, which is started within the DPE, functions as a naming and directory facility to uncover services. Services and service-containers in the CLARA registry are described using unique names, types and descriptions of their functionalities. The CLARA naming convention defines the service container name as: *DPE-host-IP:service-container-name,* where the service-container-name is an arbitrary string specified by the user. Likewise, the service name is constructed as: *DPE-host-IP:service-container-name:service-engine-name*. The service-engine-name is the class that implements CLARA *Engine* interface. The service is advertised by its service description in the registry. Querying the name and a service functional description defines the service discovery process. Registration database is duplicated in the front-end DPE (FE), controlled by the master registration service. Each registration service periodically pushes its entire database to FE. So now the orchestrator has an option to query the local DPE for local service discovery or accessing FE for registration information of services and/or containers of entire CLARA platform.

IV. Application Composition and Data Dynamic Routing

A service composition is comprised of services that have been assembled to provide the functionality required to accomplish a specific data processing task. CLARA distinguishes between two types of service compositions: primitive and complex. Primitive compositions are based on a

single simple routing statement and use message exchange across two or more services. Complex compositions however contain multiple routing statements, including conditional routing statements. Because the framework's requirement for services is to be agnostic to any data processing logic, one service may be invoked by multiple data processing applications, each of which can engage that same service in a different composition. The service-oriented approach of CLARA changes the overall complexion of a data processing 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 data processing is increasingly represented by a growing body of services that exist within a continuously expanding service inventory. CLARA introduces dynamic routing of the transient data envelopes. Data routing information is stored within the *composition* field of the transient metadata. The *composition* field of the metadata carries the information about the application service based design, telling the receiving service where the input data is coming from and where the result of the engine execution will be directed.

TABLE I. CLARA APPLICATION COMPOSITION OPERATORS

Operator	Description	Example
+	data link operator defining the data route	s1 + s2
,	data multiplexing or *logical OR* operator	s1 + s2,s3 s1,s2 + s3
&	logical AND operator	s1,s2 +&s3
;	data branching operator indicating the end of a statement	s1+s2; s2 +s3;
{ }	scope operator defines a set of routing instructions	unbound { s1+s2; s2+s3+s4; }
==	service state equal operator	if(s1==state1)
!=	service state is not-equal operator	if(s1 != state2)

if,elseif,else	conditional statement operators	If(s1 == state1) {
		} elseif(s1 != state2) {
		} else { }

unbound	defines a scope not bound to a condition	unbound { }
&&	logical AND operator	if ((s2 === "xyz") && (s1 == "abc")) { s2 + s3; }
‖	logical OR operator	if ((s2 === "xyz") ‖ (s1 == "abc")) { s2 + s3; }

As a result CLARA application design can be altered during the application execution time. Table 1 describes CLARA application design operators.

V. NAIADS IMPLEMENTATIONS AND PERFORMANCE

The NAIAD is collaborative project between JLAB and NASA to develop a prototype for the next generation Earth Science multi-sensor data fusion, processing, and analytics framework [7]. Maximizing information content by combining multi-sensor data and enabling advanced science algorithms is critical to reducing uncertainty in weather and climate research. The data fusion of Earth Science observations is a challenge - it is complex and the data volume is large and distributed among many data centers. NAIADS is addressing this challenge by developing and running critical services, such as into-memory Data Staging, multi-sensor coincident data-Events Builder, data-Event streaming with IO optimization, and user-oriented on-line services for data processing control and analytics in the CLARA environment. The NAIADS architecture design and workflow is shown in Figure 4.

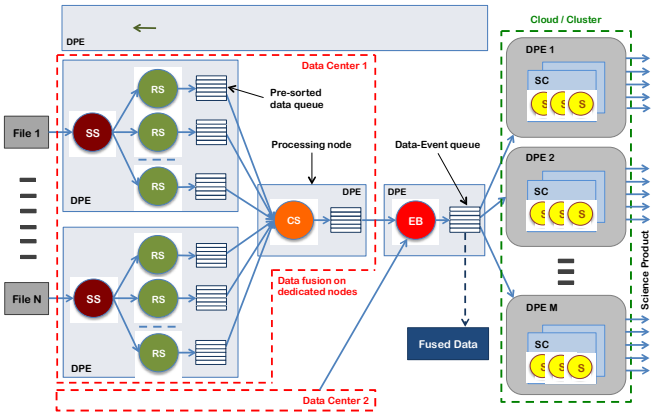

Figure 4. The NAIADS architecture and workflow: data staging (SS), reading (RS), concentrating (DS) and data-Event building (EB) is performed on dedicated nodes (within red dashed line), data-Event-based streaming and processing on Cloud/Cluster with minimized IO indicated within green dashed line. The option of creating Fused Data Product, shown in blue, allows stopping and resuming process, maintaining optimized IO.

Extensive tests of the NAIADS fase-1 were performed on the Amazon cloud (AWS) with as many as 16 S2 cloud nodes. Approximately one month of SCIAMACHY data was processed during these tests. NAIADS AWS Cloud compute environment was configured to have 16 C3/4.8xlarge, compute-optimized instances.

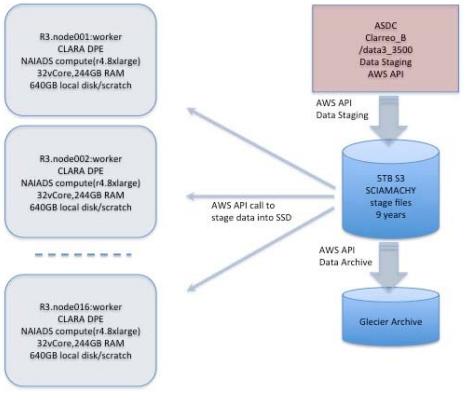

Figure 5. AWS Cloud configuration for NAIADS, fase-1: 16 compute optimized worker nodes with the data transfer and storage diagram.

The SSD storage on every instance, and fully redundant AWS Simple Storage System (S3), as well as Glacier archive were

used to store MODIS Level-2 and SCIAMACHY Level-1data sets and source code. AWS cloud configuration for NAIADS fase-1 is shown in the Figure 5.

The Figure 6 shows the NAIADS fase-1 data processing performance in a multicore and multi-node system of AWS S2. The test included SCIAMACHY Level-1 data merging with IGBP surface index, data quality control and spectral resampling. Every data record for a singe SCIAMACHY measurement represents a data event.

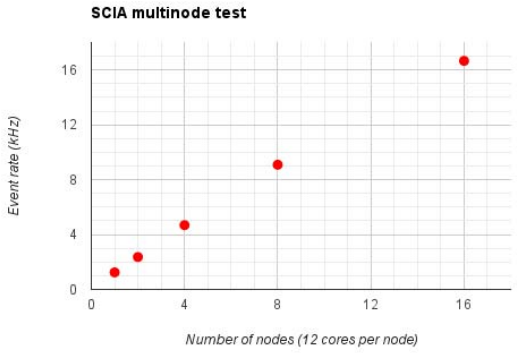

Figure 6. The NAIADS test case (using CLARA Java binding) processing performance scaled over 16 computing nodes (32-core).

CONCLUSION

An SOA-based data processing framework has been developed, presenting service interface implementations in Java, C++ and Python languages. The framework was successfully used to develop the Clas12 (JLAB) experimental data processing [8], as well as processed data correlation and mining applications (ODU). Nuclear physics data processing applications written within the CLARA framework demonstrated increased performance and simplified maintenance. The data processing performance increase is largely due to data stream processing, solving the problem that input data must be processed without being physically stored. The CLARA application is made of loosely coupled software building blocks. CLARA application designer's main focus is the data flow between application building blocks. As a result the CLARA application is more robust and agile, since application building blocks can be improved individually and can be replaced. This type of application is more fault-tolerant, since faulty blocks can be replaced without bringing down the entire application. CLARA application is also elastic, since at run-time, new data processing blocks can be added to enhance application functionality or existing blocks can be removed to fit available computing resources. The current focus of CLARA development is on efficient Earth Science Data fusion within the NAIADS project

ACKNOWLEDGMENTS

The NAIADS project, and Earth Science oriented application of the CLARA framework is funded by the NASA ROSES 2014 program - Advanced Information Systems Technology.

REFERENCES

[1] J. Schekkerman, "How to Survive in the Jungle of Enterprise Architecture Frameworks: Creating Or Choosing an Enterprise Architecture Framework", Trafford Publishing, 2004 - Business & Economics.

[2] B. M. Michelson, "Event-Driven Architecture Overview", Patricia Seybold Group and Elemental Links, Inc. 2006

[3] P. Morrison, "Flow-Based Programming: A New Approach to Application Development", CreateSpace Independent Publishing Platform; 2 edition, May 14, 2010.

[4] T. Erl, "SOA: Principles of Service Design", Prentice Hall, 2007, ISBN: 0-13-234482-3.

[5] V. Gyurjyan "xMsg: General Purpose Publish-Subscribe Messaging System", unpublished.

[6] P. Hintjens, "ZeroMW: Messaging for Many Applications", "O'Reilly Media, Inc., March, 2013

[7] C. Lukashin et al., "Earth Science Big Data Challenge: Data Fusion NAIADS Concept and Development" presentation at the NASA Earth Science Technology Forum, Pasadena, CA, June 2015.

[8] V. Gyurjyan *et al,* "CLARA: A Contemporary Approach to Physics Data Processing*", Journal of Physics: Conf. Ser.* Volume 331, 2011