Distributed Software Framework

For Biosphere 2 Land Evolution Observatory (LEO) Autonomic Cyber-Physical System (ACPS)

Shafiul Islam
The Department of Electrical and Computer Engineering
The University of Arizona
Tucson, U.S.A.
jacky@email.arizona.edu

Abstract— **This paper presents the architecture, design, and implementation of a real-time Distributed Software Framework for Biosphere 2 Land Evolution Observatory Autonomic Cyber-Physical System, which uses an optimum technology mix discovered through intensive research, design, and development over a period of two years (2010- 2012) using a novel adaptable process framework named as Jacky's Universal Process. It has a Service Oriented Architecture with Publish/Subscribe interaction pattern and Object Oriented Design. It applies self-healing feature of Autonomic Computing and uses Cloud Computing and OpenSplice Data Distribution Service. The distributed system software (software + service) of this framework is a complete production quality software product that requires near zero maintenance since only sensor drivers for new sensor types need to be developed and by appropriately mixing and matching the services all required system level capabilities can be provided. This framework deployed on B2 server, is capable of handling 45x the expected load having a total of about 148,500 sensors. It is highly reliable, robust, fault tolerant, scalable (both vertically and horizontally), extensible, secured, and easy to use. It successfully resolves all technological risks, provides concept consistency, and supersedes the functional and non-functional requirements.**

Keywords-distributed software framework; data distribution service; service oriented architecture; autonomic computing, jacky's universal process.

I. INTRODUCTION

The Biosphere 2 Land Evolution Observatory (LEO) is an interdisciplinary project aimed to quantify various earth and atmospheric processes to understand the complex non-linear interaction among these processes by coupling controllable physical systems with numerical models of the interacting processes using a cyber-physical system, which is a specialized cyberinfrastructure (CI) for LEO and referred to as Autonomic Cyber-Physical System (ACPS). ACPS requires a highly reliable, robust, fault-tolerant, scalable, extensible, and easy to maintain real-time Distributed Software Framework (DSF) with a life span of about 10 years that can be deployed on any heterogeneous distributed system and resolve integration risks. The primary users of this framework are researchers and engineers interested in development of scientific domain specific applications and computational models, which in turn are meant to be used by scientists and students for research and education (e.g., CI for Atmospheric Sciences, Earth Sciences, and Engineering Research) as mentioned in NSF's CI vision for 21st century discovery [2].

In order to meet the challenging requirements and resolve technological risks, autonomic computing, cloud computing, service oriented architecture (that uses publish/subscribe interaction pattern), and object-oriented design were used. The optimum technology mix was discovered through intensive research, design, and development over a period of two years (2010-2012). In this paper, the final production quality architecture, design, and implementation of ACPS DSF for Biosphere 2 LEO are presented. A secondary outcome of this research and development effort, also introduced in this paper, is the inception of a novel adaptable process framework for software engineering of self-managing distributed systems, which is named as Jacky's Universal Process (JUP).

The rest of the paper is organized as follows: Section II provides a theoretical foundation through literature review; Section II explains the methodology used; Section IV lists the requirements, and discusses the architecture and design; Section V discusses testing, and results; Section VI discusses experimentation and results; and finally, Section VII describes conclusion and future work.

II. LITERATURE REVIEW

At present, no similar distributed software framework for cyber physical systems that use autonomic computing exist. Hence the fundamental concepts are briefly presented here to provide the theoretical foundation.

A. Distributed Systems

A collection of independent systems that appear as a single coherent system is called a distributed system [3], which has key goals of achieving reliability, availability, adaptability, expandability, scalability, robustness, and fault-tolerance (through redundancy) while providing distribution transparency [4]. ACPS DSF, being a distributed software framework, naturally provides the non-functional needed by Biosphere 2 LEO.

 ISBN: 978-1-61208-230-1

B. Autonomic Computing

The overall goal of Autonomic computing, modeled upon autonomous nervous system, is that computing systems will self-manage taking only high-level objectives from administrators (human beings) [5]. The four aspects of self-management are: 1) Self-configuration; 2) Self-optimization; 3) Self-healing; and 4) Self-protection. In ACPS DSF, self-healing aspect is implemented to provide fault tolerance for critical system-level services, and monitoring and notification for sensors of the physical system.

C. Cloud Computing

Data center hardware and software is known as a cloud [6]. Using a composability methodology, cloud computing systems can be classified into any of the five layers [7]: 1) Cloud Application Layer (SaaS); 2) Cloud Software Environment Layer (PaaS); 3) Cloud Infrastructure Layer (IaaS); 4) Software Kernel; 5) Hardware and Firmware. For example, Amazon EC2 is IaaS, Google AppEngine SaaS, and Microsoft Azure is PaaS [6]. ACPS DSF system-level services need to be run locally on Biosphere 2 servers to avoid latency issues as experienced during testing on Amazon EC2. However, ACPS DSF application-level software can easily be deployed to a cloud.

D. OMG Data Distribution Service

Object Management Group (OMG) Data Distribution Service (DDS) is an open specification for publish-subscribe (PS) data distribution systems [8] that attempts to provide formal definition for defining Quality of Service (QoS) to configure service and help connect information producers (publishers) with information consumers (subscribers). Many real-time applications, including ACPS DSF, have the need to have pure data-centric architectural pattern and take advantage of DDS. OpenSplice [9] is the most advanced, complete and widely used (commercial and open source) implementation of OMG DDS specification. This is a tried and tested commercial-of-the-shelf product and was chosen for ACPS DSF as the OMG DDS implementation of choice.

III. METHODOLOGY

This project was primarily a complex large-scale interdisciplinary engineering project with intensive technology research. The greatest risks in the project were the technical risks and the greatest challenge was to maintain conceptual integrity among interdisciplinary Biosphere 2 staff members. The Spiral model was applied when the project was completely risk driven. As critical risks were resolved by incorporating new technologies like OpenSplice DDS and as development moved from middleware / distributed system software towards distributed application software, the approach moved more towards Lean (Agile) [11] principles. All of the critical risks have been resolved by developing Proof of Concepts (PoCs) for Data Turbine, OpenSplice DDS, real-time Visualizations using Matlab compiled codes, and coming up with ways to deliver data from DDS and plots over the web using Java Server Pages (JSP). Code quality has been ensured by incorporating recommendations of S. McConnell [18] whenever applicable.

By going through the activities in the engineering notebook and through self-reflection, the hybrid (model that was being used naturally) has been extracted. John S. Miranda, a manager at Intel, proposed that any organization that tries to implement such hybrid approach should have a set of questions / criteria to decide the best mix. This research confirms that requirements stability, software generalizability, software life expectancy and dependency are some key criteria.

Although, at present, there is no quantitative data to confirm the effectiveness of such a hybrid approach (the focus of this project was in research, design, and development of ACPS DSF), but the fact that a complicated large-scale software project like ACPS DSF was very successful may set some foundation for future research. In this paper a novel adaptable process framework for Self-managing Distributed Systems that adds a third dimension to the Rational Unified Process (RUP) [1] [12] is proposed. This new process framework is named as **Jacky's Universal Process (JUP)** of Software Engineering for Self-managing Distributed Systems. As shown in Figure 1, while the pre-existing axes from RUP provide sequential increments and iterative workflows, in JUP the third dimension provides parallel augmentations. Augmentations are different from increments in that they are very loosely coupled services that can be connected or disconnected anytime as required. Each

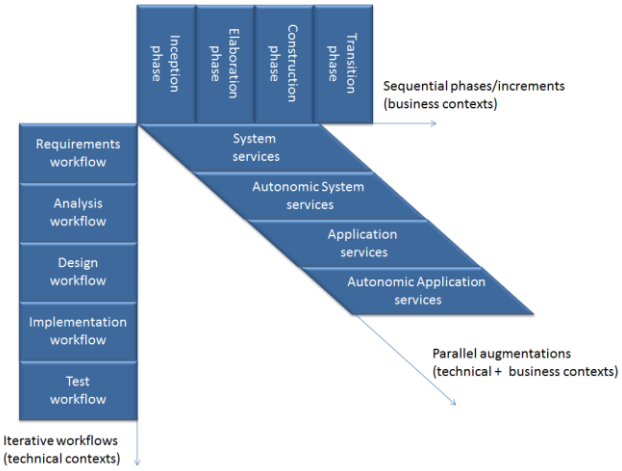

Figure 1. Jacky's Universal Process (JUP)

parallel augmentation can apply Spiral, Lean (Agile), or Hybrid e.g. Spiral + Lean (Agile), or any other model as appropriate. The four universal categories of services in this third augmentation axis are:

- **System Services:** Any distributed system level services.
- **Autonomic System Services:** Any or all of Self-* features of Autonomic Computing at the system level (global).
- **Application Services:** Any distributed application level services.

- **Autonomic Application Services:** Any or all of Self-* features of Autonomic Computing at the application level (local).

The four universal categories of services can be seen in the ACPS DSF Software Architecture (Figure 3). For example, Universal Critical Services are System Services, Autonomic Managers are Autonomic System Services, Visualizations are Application Services, and Control Panel is an Autonomic Application Service. At present use of JUP in different kinds of distributed system that have some capabilities of Autonomic Computing is advocated. Further research in JUP will provide some quantitative measures of its effectiveness. For now, use ACPS DSF as the case study to learn JUP by example.

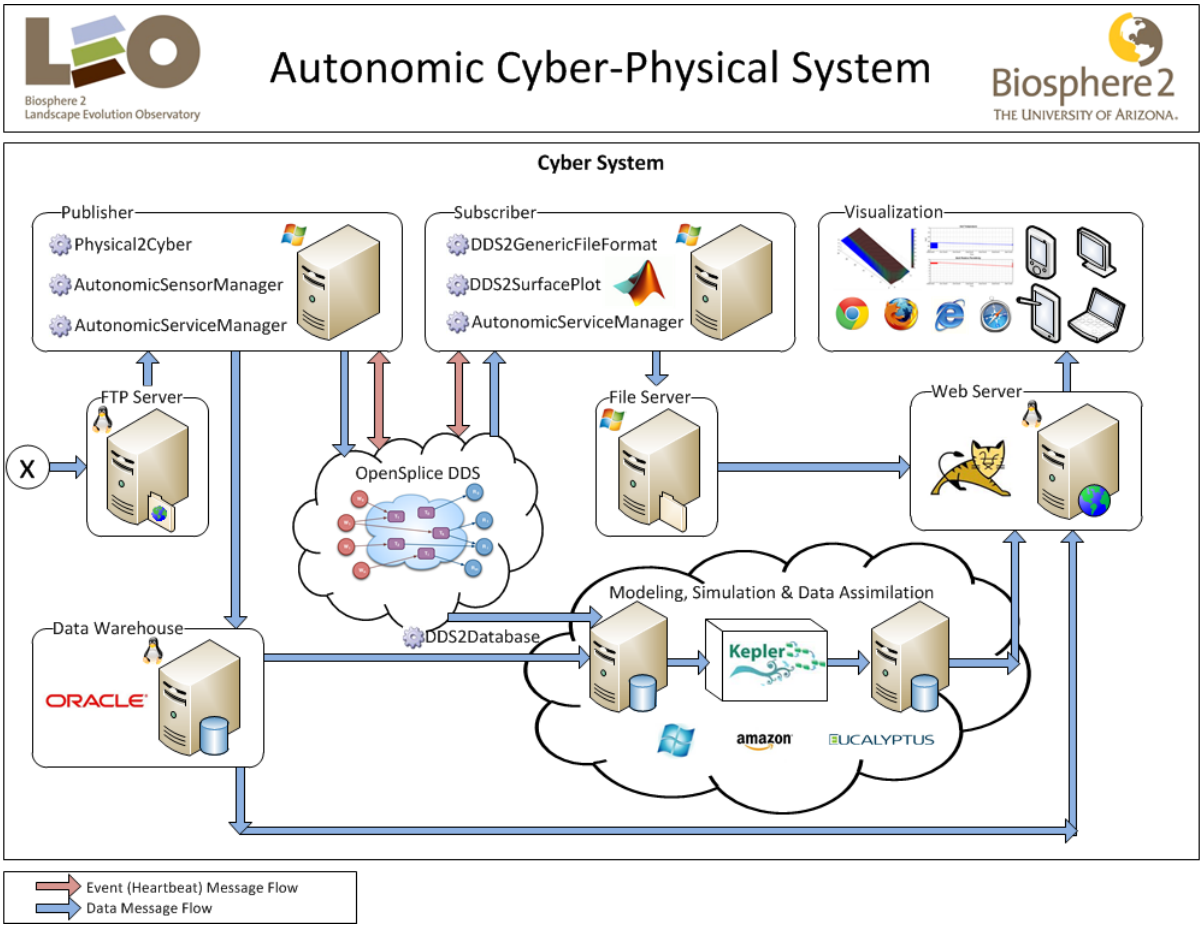

Figure 2. System Architecture (Cyber System)

IV. REQUIREMENTS, ARCHITECTURE, AND DESIGN

A. *Requirements*

As any other project, the requirements of this project were very vague initially. The vision, overall requirements, and detailed requirements have been collected through interaction with Biosphere 2 Scientists and Biosphere 2 Staff members. At any point, technology risks and non-functional requirements were the key drivers for all subsequent system architecture, software architecture, software design and implementation. The overall requirement was to research, design, and develop a distributed software framework that would facilitate the establishment of LEO cyberinfrastructure by providing a standard reliable, robust, and fault-tolerant means of data acquisition, data distribution, data visualization, data assimilation, modeling, and simulation. No hard and fast metric for feature requirements were defined, but the overall requirements can be listed as follows:

- The software framework should collect data from the physical system and store it in database.
- The software framework should make data available to real-time monitoring and visualization.
- The software framework should have facility for off-line modeling and simulation.
- The software framework should use technologies best for LEO's cyberinfrastructure.
- The visualizations should be available to students/faculty members over the web.
- The software framework should be:
  - Scalable
  - Reliable
  - Robust
  - Fault tolerant

 ISBN: 978-1-61208-230-1

- Easy to maintain (preferably no maintenance)
- Easy to extend

B. System Architecture

The system architecture, provided in Figure 2, is the final standard cyber system architecture consisting of heterogeneous systems, which can be modified in the future if desired and/or required. In the future, there will be at least one feed-back loop going from the cyber system to the physical system.

Basically, this architecture shows that the entry into the cyber system happens through the file system—a 'Source' folder where the physical system drops measurement data files. From there, Physical2Cyber uploads data to both database and DDS. From the DDS, DDS2GenericFileFormat samples the most recent sensor values in to a special generic file format for current and future web/cloud applications to read. Also, there is a DDS2SurfacePlot, a composite service, which uses CommandExecuter to generate surface plots. DDS2Database is service that can be used to sample DDS contents directly into database. AutonomicSensorManager monitors sensors and writes notifications of sensor failures according to defined policy and also stores knowledge of failures. AutonomicServiceManager monitors the heartbeats of the critical services according to the defined policy. In case of failures of any services, it takes the appropriate actions as defined in policy while storing knowledge of any failures. The web/cloud application reads data from the 'Sink' folder (in particular, from files created and updated byDDS2GenericFileFromat and notification files created by AutonomicSensorManager), and database to show text data, visuals, text and provide audio warnings when required [13].

All of the servers shown in this system architecture are important (ftp server being the most important one) and the flow of data and events through them can easily be analyzed

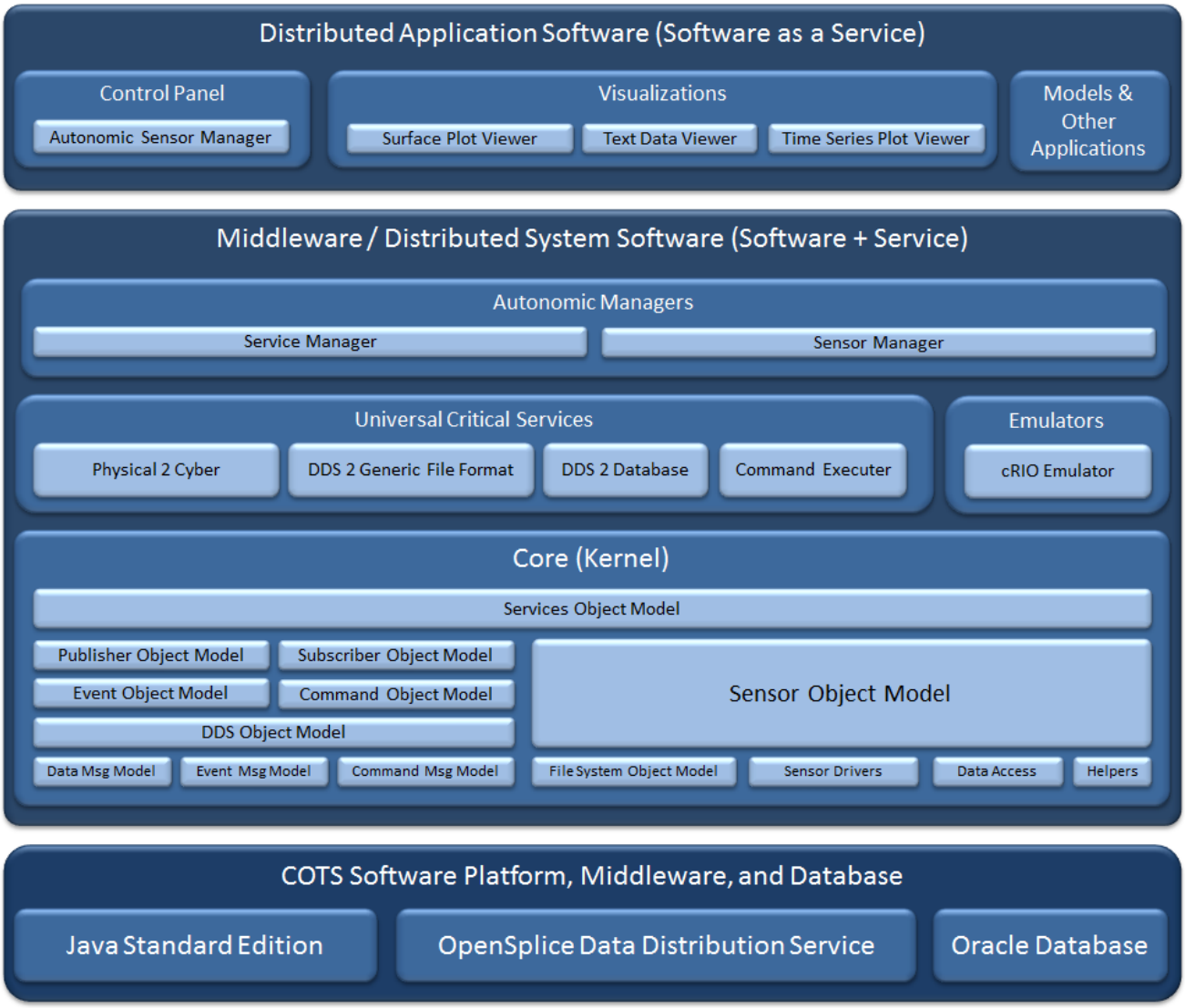

Figure 3. Software Architecture

and best understood using the system architecture, which also shows the best technology mix. The physical system is modeled as a system that generates data. As a matter of fact, details of the physical system, which was outside the scope of my responsibility, is not provided to emphasize the cyber system.

C. Software Architecture

The ACPS Distributed Software Framework (DSF) architecture is shown in Figure 3. It basically consists of three layers in an open architecture i.e. any top level layer can call any of the bottom layer(s). This architecture is a Service Oriented Architecture (SOA) that combines layered and data space architectural patterns [3]. It is SOA because it is composed of a collection of services that provide the fundamental services, which can be mixed and matched to provide the universal set of capabilities ACPS will need over its life-span. These services use Publish/Subscribe interaction pattern by applying the first open international middleware standard—OMG Data-Distribution Service for Real-Time Systems [14]. The Middleware / Distributed System Software (Software + Service) and Distributed Application Software (Software as a Service) consists of subsystems (projects) and each subsystem consists of modules (packages), which in turn contains classes/components.

Each of the software layers are described below starting with the bottommost layer first:

Commercial of the shelf (COTS) Platform, Middleware, and Database: This layer of software represents the industry standard proven, tried and test products used as the foundation for ACPS DSF. The major technologies in this layer are:

- **Java Standard Edition:** Java was chosen as the software platform of choice for performance, versatility, portability, and security [15] that ACPS DSF requires.
- **OpenSplice Data Distribution Service (DDS):** OpenSplice DDS is the global leader in real-time data distribution middleware technology [9]. It is the strictest implementation of Object Management Group (OMG) DDS Open Standard providing high scalability, low latency, and fault-tolerance for real-time distributed systems.
- **Oracle Database:** Oracle database provides the foundation for high quality information storage and delivery [16]**.**

Middleware / Distributed System Software (Software + Service): This system layer consists of the system software and services (S+S) that should be sufficient to provide all of the system level capabilities ACPS will ever need over its entire life-span of about 10 years. This layer is complete in that by combining and configuring the universal critical services that use the core, all of the current and future system level needs can be met provided the project plan does not change radically. Other than sensor driver development, and shell script development to meet particular deployment need, no maintenance to this level is neither expected nor recommended. Overall, this layer provides the scalability, robustness, reliability, and fault-tolerance along with fundamental/core ACPS system (and significant application) logic. If ever required, the software and services may be extended without making any changes to the existing core, services, and emulator. This layer has the following subsystems:

- **Core (Kernel):** As the name implies, contains all of the core system (and some application level) logic fundamental to ACPS as a whole. In this subsystem, *Sensor Object Model* is the most important module that hosts the most important object oriented data structure / application programming interfaces (APIs) for all system level services. *Sensor Drivers* is the next most important module that hosts all of the derivers for sensors that define calibration functions. The *File System Object Model*, *Data Access*, and *Helpers* module contains relevant classes/APIs for files, database, and general helpers. The DDS Object Model, contains the APIs for talking to DDS that use data structures in *Data Msg Model*, *Event Msg Model*, and *Command Msg Model* modules. The *Publisher Object Model*, *Subscriber Object Model*, *Event Object Model*, and *Command Object Model* provide classes / wrapper APIs to DDS. The *Service Object Model*, contains the base class and exception class for any system level service.
- **Universal Critical Services:** These are the fundamental services that provide universal capabilities that ACPS will ever need at the system level. *Physical 2 Cyber*, *DDS 2 Generic File Format*, *DDS 2 Database*, and *Command Executer* classes represent the critical services.
- **Emulators:** At present only one *cRIO Emulator* is needed while B2 LEO is under construction. This emulator has the capability to read sensor definitions from database and generate data in exact established format specification between the physical and the cyber system.
- **Autonomic Managers (controllers):** The autonomic managers provide self-healing capabilities of Autonomic Computing to add fault tolerance at service level by *Autonomic Service Manager* module and monitoring/notification capabilities at sensor level by *Autonomic Sensor Manager* module.

 ISBN: 978-1-61208-230-1

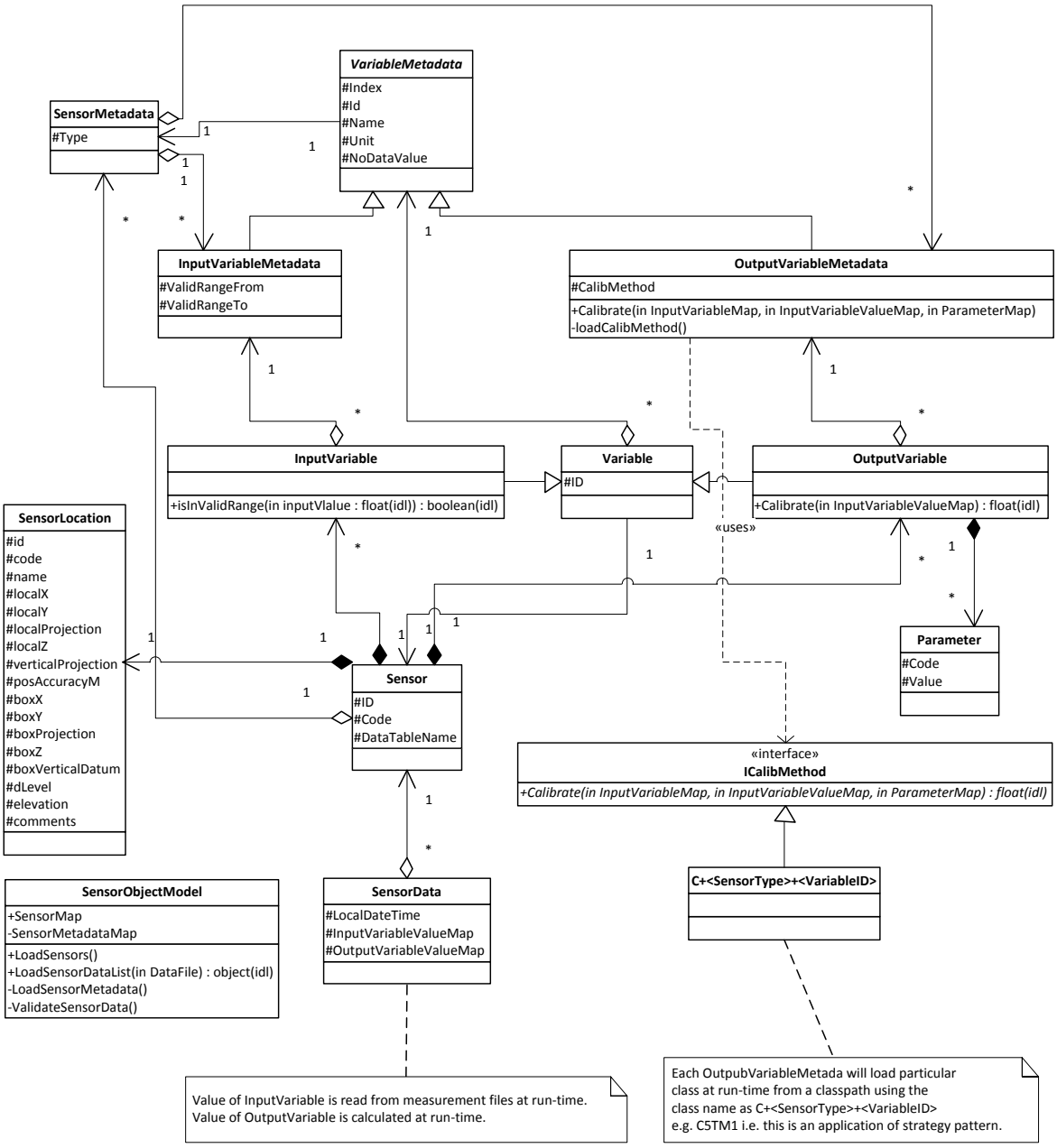

Figure 4. Sensor object model (simplified)

Distributed Application Software (Software as a Service): This layer is meant to be extended where all future work will take place. All applications in this layer are provided as service over internet and expected to be all deployed in private and/or public clouds i.e. this is the SaaS layer. Hence, all of the control panel and visualization applications are available to any portable device in the world with an internet connection and a web browser. Essentially, all of the web/cloud applications are JavaServer Pages (JSP) that read data from appropriate sources (files and database) and present them either in text or graphics format in appropriate format. *Autonomic Sensor Manager* refreshes itself periodically and monitors contents of a notification file for any sensor failures and present such failures to the user along with audio warning. *Surface Plot Viewer* displays the appropriate surface plot image generated by *DDS 2 Surface Plots* composite service. Text Data Viewer display the appropriate data from generic file format generated and updated by *DDS 2 Generic File Format* service and *Time Series Plot Viewer* shows time series data from database.

D. Design Overview

The design of ACPS DSF started with the notion of loosely coupled collection of publishers and subscribers interacting through a middleware keeping in mind the need for scalability due to the massive amount of data flow that is expected. In the design, file system was used, in addition to OpenSplic DDS, as a queue, and also as a shared memory. This way, different systems are decoupled from each other through the file system. Also, during the design, parallelism was taken as a key design criterion to provide scalability. For example, the finished ACPS DSF can be used in parallel by

splitting the load of one cRIO to multiple cRIOs, which in turn would mean splitting the queue, running services in parallel, and loading data to database in parallel either at the schema level or at the server level. Also, in the design, synchronization issues have been carefully assessed and file locks were always used for any writing operations. Overall, this ACPS DSF is designed to make it easier to develop ad-hoc applications, which can easily consume data either from the file system or from database. The OpenSplice DDS serve as a shared memory where most recent value of all sensors (from a hill slope) are kept up to date. Any interested application can also sample any particular number or types of sensors at the desired sampling rate.

E. Class Design

Object-oriented design has been used throughout ACPS DSF. Although most of the design evolved over time, the classes in Sensor Object Model (Figure 4) were carefully designed first, even before hitting a single key, and the design was always kept in sync with code. This object model is the most important data structure integral to ACPS DSF. In this design, it is assumed that a sensor has inputs (measurements) and outputs (calibrations) which can be a function of any number of inputs. In order to make these calibrations as general as possible, the strategy design pattern [17] was used. Thus, classes collectively referred to as sensor drivers, define calibration functions by implementing ICalibMethod interface. The next most important sets of classes (in respective packages) are those that form wrappers around OpenSplice DDS namely those in DDS Object Model, Publisher Object Model, Subscriber Object Model, Event Object Model, Command Object Model—all of these packages/modules are part of the ACPS DSF core. ACPS DSF itself is composed of a number of subsystems: Controllers, Core, Emulators, Services, and User Interface. As the name implies, Core/kernel is the most fundamental critical subsystem to the entire distributed system. Each of these subsystems is divided into packages/modules as required.

Design pattern [17] was used in three places: Strategy Pattern in implementing sensor drivers mechanism, Command Patten for commands and events mechanism, and Singleton Pattern was used to make sure that only one instance of Physical2Cyber can be executed per folder it is monitoring.

F. Message Structure Design

In this DDS-centric design, message structure is very important. In order to make sure that all of the kinds of messages that will ever flow through the middleware, are general message structures which are specialized over layers of software using command pattern. In OpenSplice DDS, these messages were defined using Interface Definition Language (IDL), and when passed through a tool (idlpp) that is part of the OpenSplice, the relevant Java classes were generated.

G. Database Design

Database is also an integral part of the system. In order to decouple the design of the database from the design of ACPS DSF, special view specifications were created that serve as interface between the full database and the view of the database in light of ACPS DSF. These views are prefixed with ACPS and all that matters to ACPS DSF is the exact number of attributes with proper data types. The query used to get these attributes may change and are not a concern for ACPS DSF as long as this interface is not broken.

V. TESTING, RESULTS, AND DISCUSSION

A. Testing Overview

The test cases designed were goal oriented as recommended by Fenton [19]. All of the test cases basically had nominal scale of measurement: Success or Failure. Since all of the test cases passed, the correctness and quality of ACPS DSF were successfully validated.

B. System Testing—Biosphere 2 LEO Server Deployment

In this deployment scenario, three physical cRIO are connected which provides the expected amount of load once the physical system is completed. In the actual physical system, each cRIO is expected to drop files in ‘Source’ folder every 10-15 mins. However, for the purpose of testing,

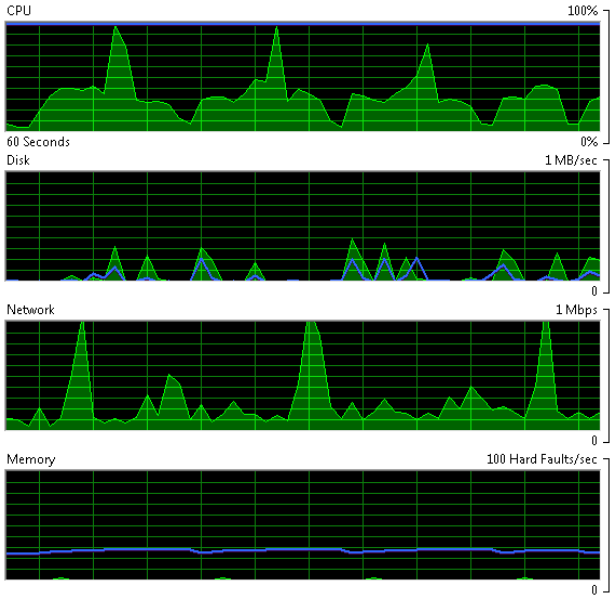

Figure 5. 'Subscriber' machine CPU, Disk, Network, and Memory usage (when DDS2SurfacePlot is running)

the connected cRIOs (x, y, and z) are configured to drop files every 2 mins. In addition to the physical cRIOs, three cRIOEmulators (x, y, and z) are also configured to drop files every 1.5 mins. In accordance with the System Architecture (Figure 2), the ‘Publisher’ machine is running Physical2Cyber, AutonomicSensorManager, and AutonomicServiceManager. Similarly, the ‘Subscriber’ machine is running DDS2GenericFileFormat, DDS2SurfacePlot, and AutonomicServiceManager.

 ISBN: 978-1-61208-230-1

Resource usage for 'Subscriber' machine is shown in Figure 5, and resource usage of 'Publisher' machine is shown in Figure 6. From these statistics, it is evident that resources in 'Publisher' machine can easily be more utilized by applying more load i.e., one 'Publisher' machine is capable of serving multiple hill slopes (physical systems).

Figure 6. 'Publisher' machine CPU, Disk, Network, and Memory usage

This demonstrates that the system has the capability to provide more throughput than required. However, in 'Subscriber' machine resource utilization is more intense when plotting routines of Matlab is in use as expected. Thus it may be reasonable to have one 'Subscriber' machine per hill slope. Another option could be to have a dedicated machine to run DDS2SurfacePlot, which is indeed the most resource intensive service and in this configuration, a single 'Subscriber' machine could serve multiple hill slopes.

C. System Testing—Amazon Web Services/Cloud Deployent

A test bed was created on Amazon Web Services (EC2) with minimal machine (m1.small) configuration. At present, a shared folder on 'ACPSServer' to be accessed by 'ACPSPublisher' and 'ACPSSubscriber' machines could not be created. So the backup plan was to test everything on the 'ACPSServer' machine with predefined m1.small configuration. Since this server is in Amazon EC2, loading time to database was taking much longer ~ 4 minutes for cRIOEmulatorX, which has the largest number of sensors attached to it. The time to load to database (including network latency) directly determines the throughput of the system. Hence for the purpose of testing, cRIOEmulatorX, cRIOEmulatorY, and cRIOEmulatorZ were configured to generate sensor measurement data files every 5 minutes. A total of three sets of application setup were tried: 'Publisher' setup, 'Subscriber' setup, and 'Web Server' setup.

D. Reliability Testing

Until present time, there has never been any software downtime. The best way to understand the reliability would be to have a look at a production release of Biosphere 2 Sensor Network Data Acquisition System (Bio2SNDAS), which is the precursor to Physical2Cyber service in ACPS DSF. In the ACPS DSF, in the final release, none of the universal critical services ever failed i.e. zero crashes so far during non-stop execution.

E. Robustness Testing

Error could be introduced into the ACPS DSF from three primary places: sensor measurement files, configuration files, and database sensor definitions. Error checking, sensor verification and validation and lots of try-catch blocks were used. As a result, the final release of ACPS DSF never crashed due to any input error.

F. Fault-tolerance Testing

ACPS DSF is configured using batch files in both 'Publisher' and 'Subscriber' machines with shortcut from startup folder such that incase of any machine restart, it will start automatically. Also, self-healing capabilities provided by AutonomicServiceManager ensures that if any service dies, it is restarted according to the defined policy. As of now, none of the service ever crashed. However, for the purpose of testing, they were manually terminated, and in all of the trials, AutonomicServiceManager was always able to do its job.

G. Scalability Testing

ACPS DSF has been tested thoroughly, and it can be parallelized at the cRIO level, service level, database schema level, and obviously at the database server level. As a matter of fact, the current deployment of ACPS DSF far exceeds the requirements. As long as the network and the database does not become a bottleneck, ACPS DSF is highly scalable. To overcome any network latency, the database should be as geographically close as possible to the cRIOs and the 'Publisher' machine. In order to overcome any possible database insertion inefficiency, various techniques including indexing that are common in database optimization should be applied. Hence, this software product supports both scaling up and scaling out both of which are limited only by the available resources and the imagination of the administrator of the product.

H. Maintainability Testing/Assessment

ACPS DSF requires near zero maintenance for the middleware / distributed system software level since only new sensor drivers need to be developed for new sensor types. The universal critical services can be mixed and matched to provide all of the system level capabilities ACPS will ever need. Hence, at this level, it is a complete product and is expected to be used like a commercial of the shelf product.

 ISBN: 978-1-61208-230-1

I. *Extensibility Testing/Assesment*

At the distributed application software level, ACPS DSF is expected to be extended throughout its entire lifespan. It has a service oriented architecture using publish/subscribe interaction pattern. Also, it has object-oriented design. Anyone can extend ACPS DSF both at the middleware / system and at the application level.

J. *Usability Testing/Assessment*

ACPS DSF is very easy to use once a user learns how to configure it, which is also easy. Basically, once it is configured, due to self-healing feature of autonomic computing, no human monitoring of the software is required. As a matter of fact, it can be considered as a fire and forget software system.

K. *Security Testing*

As of now, all user input is through button clicks. Web/Cloud application is completely decoupled from the middleware. Security was taken in to account throughout the entire design and development cycle. Also, no nice-to-have features were implemented, which often lead to security holes. Overall, ACPS DSF is highly a secured software product, however, the Web/Cloud app could have vulnerability in that user could try SQL injection from the address bar. At the same time, these critical web pages are not expected to be exposed to the general public but only restricted to the B2 employees. It would be nice to have a team of ethical hackers try to compromise it and provide further insight into security issues (if any) of the Web/Cloud. For now, no further tests were done.

Mean interarrival time (s):		600		
Mean service time (s):		4		
# cRIO per hill slope:		3		
# of Hill Slope	**# of cRIO**	**Interarrival rate, λ (s)**	**Service rate, μ (s)**	**Utilization, σ**
1	3	0.005	0.25	0.02
2	6	0.01	0.25	0.04
3	9	0.015	0.25	0.06
4	12	0.02	0.25	0.08
5	15	0.025	0.25	0.1
6	18	0.03	0.25	0.12
7	21	0.035	0.25	0.14
8	24	0.04	0.25	0.16
9	27	0.045	0.25	0.18
10	30	0.05	0.25	0.2
11	33	0.055	0.25	0.22
12	36	0.06	0.25	0.24
13	39	0.065	0.25	0.26
14	42	0.07	0.25	0.28
15	45	0.075	0.25	0.3
16	48	0.08	0.25	0.32
17	51	0.085	0.25	0.34
18	54	0.09	0.25	0.36
19	57	0.095	0.25	0.38
20	60	0.1	0.25	0.4
21	63	0.105	0.25	0.42
22	66	0.11	0.25	0.44
23	69	0.115	0.25	0.46
24	72	0.12	0.25	0.48
25	75	0.125	0.25	0.5
26	78	0.13	0.25	0.52
27	81	0.135	0.25	0.54
28	84	0.14	0.25	0.56
29	87	0.145	0.25	0.58
30	90	0.15	0.25	0.6
31	93	0.155	0.25	0.62
32	96	0.16	0.25	0.64
33	99	0.165	0.25	0.66
34	102	0.17	0.25	0.68
35	105	0.175	0.25	0.7
36	108	0.18	0.25	0.72
37	111	0.185	0.25	0.74
38	114	0.19	0.25	0.76
39	117	0.195	0.25	0.78
40	120	0.2	0.25	0.8
41	123	0.205	0.25	0.82
42	126	0.21	0.25	0.84
43	129	0.215	0.25	0.86
44	132	0.22	0.25	0.88
45	135	0.225	0.25	0.9
46	138	0.23	0.25	0.92
47	141	0.235	0.25	0.94
48	144	0.24	0.25	0.96
49	147	0.245	0.25	0.98
50	150	0.25	0.25	1

Figure 7. Finding utilization of 1.0

VI. EXPERIMENTATION, RESULTS, AND DISCUSSION

From all of the testing conducted in Section **Error! Reference source not found.**, it is evident that not only does ACPS DSF meets all of the functional and non-functional requirements of B2 LEO, but exceeds them by more than 10x. In B2 server setup, one deployed instance of ACPS DSF is already demonstrating capability to serve more than 10 hill slopes. The calculation for this load multiplier for the present experimental configuration on B2 LEO is provided in **Error! Reference source not found.**. Even with these experiments, it is evident from the resource usage profiles that the machine is not fully utilized. This is indeed an exploratory analysis. From this analysis, a theory is established that ACPS DSF have the capability by far more than 10x. To test the theory, a confirmatory analysis is conducted using Queuing Theory [20].

TABLE I. LOAD MULTIPLIER CALCULATION

	Actual system	**Experimental emulated system**	**Experimental physical system**
Sensor Measurement File interarrival time (minutes)	$t_{actual} = 10$	$t_{emulated} = 1.5$	$t_{physical} = 2$
Number of cRIO	$n_{actual} = 3$	$n_{emulated} = 3$	$n_{physical} = 3$
Load multiplier	$(t_{actual} / t_{actual}) * (n_{actual} / n_{actual}) = 1$	$(t_{actual} / t_{emulated}) * (n_{emulated} / n_{actual}) = 6.67$	$(t_{actual} / t_{physical}) * (n_{physical} / n_{actual}) = 5$
Total load multiplier (emulated + physical)	N / A	$(t_{actual} / t_{emulated}) * (n_{emulated} / n_{actual}) + (t_{actual} / t_{physical}) * (n_{physical} / n_{actual}) = 11.67 > 10$	

The Source Folder in File System can be considered as a queue where sensor measurement files arrive. The time it takes to process each measurement file, depends upon the amount of data it has, which in turn depends upon the number of sensors connected to the particular cRIO (X, Y, or Z). cRIO Y has the most number of sensors (little more than 1000). The time taken to process a measurement file from cRIO Y in the current B2 server is about 4s. For cRIO X, Z, the time taken is negligible. However, let us take the maximum service time for all of the measurement files X, Y, and Z. Hence, the interarrival rate, λ, is 3 measurement files every (10 * 60) seconds, and the service rate, μ, is 3 measurement files every 12 seconds. The utilization of server is given by equation (**Error! Reference source not found.**).

 ISBN: 978-1-61208-230-1

$$\rho = \lambda / \mu \quad (1)$$

By entering this information in an Excel Spreadsheet, the number of hill slopes for which utilization of the current system becomes one is found as shown in Figure 7.

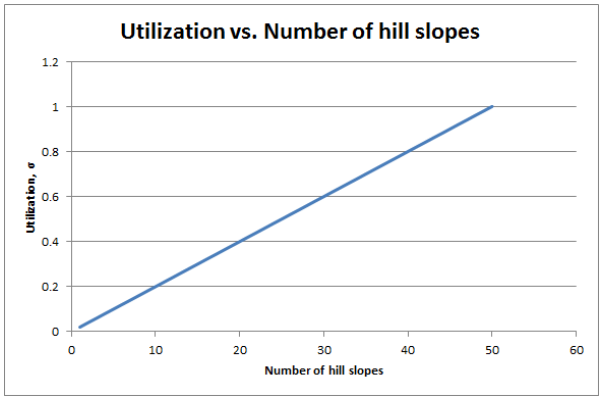

Figure 8. Graph of utilization vs. number of hill slopes

According to this calculation, the current system will attain an utilization of 1.0 when a load of 50 hill slopes i.e. 150 cRIO is provided (assuming they have the same size of sensor measurement files i.e. same number of sensors attached). A graph of utilization vs. number of hill slopes is also provided in Figure 8. It is evident that the system will have utilization of 0.9 for 45 hill slopes, even this is way more than the amount of load each cyber system is expected to have when augmented sensors are introduced in future. In order to simulate a 45x load, the cRIOEmulators (X, Y, and Z) can be configured with lower interarrival time and verify that the cyber system is still stable. Hence, an experiment is conducted with each cRIOEmulators configured to have an interarrival time of 600 / 45 = 13.33 seconds on B2 server 'Publisher' machine and are configured to use a development shared folder so that the files from physical cRIOs are not included in the experiment. The outcome of this experiment is exactly as expected. The queue (source folder) goes almost full, then Physical2Cyber completes processing them all and as soon as it finishes, more measurements file arrive. This is an excellent **utilization (0.9)** of the 'Publisher' machine and this utilization represents **45 hill slopes**. This experiment confirms the theory that ACPS DSF is not only capable of meeting the non-functional requirements but goes above and beyond—it has the ability to process the load of as many as 45 hill slopes with a server utilization of 0.9 and each hill slope having three cRIOs with each cRIO connected to about 1100 sensors (i.e. a total of about 45 * 3 * 1100 = 148,500 sensors).

It is important to note that to load measurement file from cRIO Y from Amazon cloud to LEO database server, it took about 4 minutes. Hence, both the ACPS DSF and database are highly optimized. The network latency is a major bottleneck and should be resolved by locating databases as close to the cRIO as possible to reduce latency.

VII. CONCLUSION AND FUTURE WORK

In conclusion, this was a very exciting and challenging project. Turning various ideas into reality, shaping the ideas and finding the best architecture, design, and implementation with the perfect technology mix was a great accomplishment in an interdisciplinary team like this, where maintaining concept consistency itself was a great challenge. ACPS DSF provide the ultimate technical solution for B2 LEO so that it can now focus on integrating computational models with the confidence that the underlying middleware / distributed system software is organized physically into the standard system architecture will always provide the performance and the reliability, robustness, fault-tolerance, scalability, and extendibility it needs. This ACPS DSF provide the strategic directions for all kinds of application development—applications that all fit into the distributed software framework. No more maintenance is needed (at the distributed system software level) once all of the different sensor types are purchased, and all of the sensor drivers are developed. OpenSplice DDS and Oracle Database make real-time, near-real-time, and offline analyses possible. Moreover, this optimum technology mix found through this research project resolves all of the critical technological risks. The best architecture, design, and implementation methods are clearly demonstrated through this distributed software framework, which is expected to last for the entire life-span of B2 LEO project of about 10 years. In this research project how Matlab can be used creatively to generate plots that can be delivered over the web through simple techniques as just pooling plot files generated by Matlab from JSP pages using auto refresh mechanism is clearly demonstrated. Such simple mechanisms allow thin clients to access all of the applications as a service over web/cloud. The audio alarm in addition to text alarm are extremely helpful and provide such off-site monitoring of sensor from anywhere in the world with just a portable device with internet connection and a browser. This Service Oriented Architecture allows very loose coupling enabling each subsystem to decouple from one another. In case of maintenance only the required machines can be turned down for maintenance ACPS DSF can be configured to auto-start. The Source folder where all files are dropped from the physical system act as a queue, and the Sink folder where all data and plots are dropped serve as a shared memory. The Source folder decouples the physical system from the middleware, and the Sink folder decouples any application from the middleware. Likewise the database also decouples applications from the middleware. Hence, any physical system application and/or any application software do not have to know the details of the middleware, but need to know how to access text files, pictures, and database. The middleware in the meantime provide all of the non-functional capabilities that B2 LEO ACPS will ever need.

Overall, ACPS DSF resolves all of the technology risks by providing the middleware and by providing efficient application solutions; it provides concept consistency through the framework in which all future applications can evolve. ACPS DSF is a very successful real-time product

 ISBN: 978-1-61208-230-1

with meets all of the functional and non-functional requirements. As a matter of fact, since one instance of ACPS DSF can handle the load of as many as 45 hill slopes having a total of 148,500 sensors with a utilization of 0.9, B2 LEO will never have to worry about any development in ACPS DSF middleware / distributed system software. Also, the hope is that JUP will add value to both industry and academia for software engineering of self-managing distributed software systems.

Future work consists of extending ACPS DSF by building new applications at the distributed application level. The immediate next set of work include creating services that read data from database and transform them into NetCDF format so that the very first model known as Kathy can be run for analysis. Along with the models simulation, new visualizations will be required. From there, it will all depend upon the priorities and progress of other new models. If there is new sensor types, drivers for them should be developed. Any change to ACPS DSF middleware / distributed system software is neither expected nor recommended. ACPS DSF is a finished product and should be used like a commercial-of-the-shelf software framework.

ACKNOWLEDGMENT

Thanks to Biosphere 2 LEO for funding this research and development.

REFERENCES

[1] P. Kruchten, The Rational Unified Process: An Introduction (2nd Edition), Addison-Wesley Professional, 2000.

[2] NSF, Cyberinfrastructure Vision for 21st Century Discovery, National Science foundation Cyberinfrastructure Council, 2007.

[3] A. S. Tanebaum and M. V. Steen, Distributeds Systems: Principles and Paradigms (2nd edition), Upper Saddle River: Pearson, Prentice Hall, 2006.

[4] P. Veríssimo and L. Rodrigues, Distributed Systems for System Architects, Springer, 2001.

[5] J. Kephart, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41-50, 2003.

[6] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, and G. Lee, "A View of Cloud Computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50-58, 04 2010.

[7] M. Butrico and D. Da Silva, "Toward a Unified Ontology of Cloud Computing," in *Grid Computing Environments workshop*, Santa Barbara, 2008.

[8] G. Pardo-Castellote, "OMG Data-Distribution Service: architectural overview," in *Distributed Computing Systems Workshops*, 2003.

[9] "OpenSplice DDS Overview," PrismTech, [Online]. Available: http://www.prismtech.com/opensplice/products/opensplice-dds-overview. [retrieved: September, 2012].

[10] B. W. Bohem, "A spiral model of software development and enhancement," *Computer*, vol. 21, no. 5, pp. 61-72, 1988.

[11] M. Poppendieck and P. Tom, Lean Software Development: An Agile Toolkit, Upper Saddle River: Addison Wesley, 2003.

[12] S. Schach, Object-Oriented Software Engineering, McGraw-Hill Science/Engineering/Math, 2007.

[13] S. Islam, A. Robertson, C. M. Kartchner, D. V. Sickinger, and J. L. Eyre, "Situational Awareness Detection and Warning for Airport Operations," The University of Arizona., Tucson, 2010.

[14] "Data Distribution Service Portal," Object Management Group, [Online]. Available: http://portals.omg.org/dds/. [retrieved: September, 2012].

[15] "Java SE Overview - at a Glance," Oracle, [Online]. Available: http://www.oracle.com/technetwork/java/javase/overview/index.html. [retrieved: September, 2012].

[16] "Oracle Database 11g Release 2," Oracle, [Online]. Available: http://www.oracle.com/technetwork/database/enterprise-edition/overview/index.html. [retrieved: September, 2012].

[17] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Professional, 1994.

[18] S. McConnell, Code complete: A Practical Handbook of Software Construction (2nd Edition), Microsoft Press, 2004.

[19] N. E. Fenton and S. L. Pfleege, Software Metrics: A Rigorous and Practical Approach, Revised (2nd edition), Course Technology, 1998.

[20] L. L. Peterson and B. S. Davie, Computer Networks, Elsevier Science, 2007.

 ISBN: 978-1-61208-230-1