Towards a Self-adaptive Deployment and Configuration Infrastructure for Cyber-Physical Systems

Subhav Pradhan, William Otte, Abhishek Dubey, Csanad Szabo, Aniruddha Gokhale and Gabor Karsai
Institute for Software Integrated Systems, Vanderbilt University
Nashville, TN, 37212, USA
{pradhasm,wotte,dabhishe,csanad,gokhale,gabor}@isis.vanderbilt.edu

ABSTRACT

Multi-module Cyber-Physical Systems (CPSs), such as satellite clusters, swarms of Unmanned Aerial Vehicles (UAV), and fleets of Unmanned Underwater Vehicles (UUV) are examples of managed distributed real-time systems where mission-critical applications, such as sensor fusion or coordinated flight control, are hosted. These systems are dynamic and reconfigurable, and provide a “CPS cluster-as-a-service” for mission-specific scientific applications that can benefit from the elasticity of the cluster membership and heterogeneity of the cluster members. Distributed and remote nature of these systems often necessitates the use of Deployment and Configuration (D&C) services to manage the software applications. Fluctuating resources, dynamic cluster membership and changing environmental conditions require resilience. Often human intervention is infeasible, which calls for a self-adaptive D&C infrastructure that supports autonomous resilience. Such an infrastructure must have the ability to adapt existing applications on the fly to provide application resilience and must itself be able to adapt to account for changes in the system as well as tolerate failures.

This paper describes the design and architectural considerations to realize a self-adaptive, component-based D&C infrastructure for CPSs. Previous efforts in this area have resulted in a D&C infrastructure that supports application adaptation via dynamic re-deployment and re-configuration mechanisms. Our work, presented in this paper, improves upon these past efforts by implementing a self-adaptive D&C infrastructure which itself is resilient. The paper concludes with experimental results that demonstrate the autonomous resilience capabilities of our new D&C infrastructure.

Categories and Subject Descriptors
C.2.4 [Distributed Systems]: Distributed applications;
C.4 [Performance of Systems]: Reliability, availability, and serviceability

General Terms
Adaptation, Resilience, Cyber-Physical Systems

Keywords
Self-managing Systems, Autonomous Resilience, Resilient D&C Infrastructure

1. INTRODUCTION

Cyber-Physical Systems (CPS) are a class of distributed, real-time and embedded systems that tightly integrate the cyber dimension with the physical dimension whereby the physical system and its constraints control the way the cyber infrastructure operates and in turn the latter controls the physical objects. Fractionated spacecraft, swarms of Unmanned Aerial Vehicles (UAVs) and fleets of Unmanned Underwater Vehicles (UUVs), represent a new class of highly dynamic, cluster-based, distributed CPSs. These systems often operate in unwieldy environments where (1) resources are very limited, (2) the dynamic nature of the system results in ever-changing cluster properties, such as membership, (3) failures and fluctuation in resource availabilities is common, and (4) human intervention to address these problems is rarely feasible. Owing to these traits, the system property of resilience is increasingly becoming a critical aspect for CPSs.

A resilient system is defined as a system that is capable of maintaining and recovering its functionality when faced with (1) expected as well as unexpected faults, (2) changes in the system’s environment which, at times, can result in failures due to the environment either producing unexpected inputs or not reacting to outputs as expected, or (3) errors encountered during planned system updates. In other words, a resilient system can adapt to both internal and external anomalies by modifying its normal behavior while still remaining functional. In the case of dynamic distributed systems where human intervention is extremely limited, resilience should be autonomous. Consequently, the system should be self-adaptive [25] for which it requires an adaptation engine capable of maintaining and recovering the system’s functionality by (1) adapting applications hosted on the system, and (2) adapting itself as well as other services provided by the system.

*This work was supported by the DARPA System F6 Program under contract NNA11AC08C and USAF/AFRL under Cooperative Agreement FA8750-13-2-0050. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA or USAF/AFRL.
To realize a self-adaptive CPS, we first need to understand how these systems and their applications are architected because any solution for resilience must seamlessly integrate with the system architecture. In this context, we observe that applications for CPSs are increasingly being designed using the methods of Component-Based Software Engineering (CBSE) [10] where applications are realized by composing, deploying and configuring software components in a target environment. Deployment and Configuration (D&C) of component-based software is a well studied field of research that has lead to realization of Deployment and Configuration (D&C) infrastructures that helps system designers deploy and configure large-scale component-based applications. In general, a D&C infrastructure is responsible for managing an application’s lifecycle which includes initial deployment and configuration of the application as well as run-time modifications. The Object Management Group (OMG) has standardized the Deployment and Configuration (D&C) specification [19]. Our prior work on the Deployment And Configuration Engine (DAnCE) [23, 20] describes a concrete realization of the D&C specification for the Lightweight CORBA Component Model (LwCCM) [18].

Since the D&C capability is a key artifact of any component-based system, we surmise that autonomous resilience in CPSS can be achieved by enhancing the D&C infrastructure so that it can perform the role of the adaptation engine. In turn, this means that the D&C infrastructure should manage the application lifecycle as well as handle application failures and itself be resilient via self-adaptive capabilities. These requirements have been identified in our previous work [24]. However, existing D&C infrastructures including our prior efforts do not yet support these requirements. Even though some solutions address the requirement for a D&C infrastructure that is capable of application adaptation via hot deployment [14], these solutions are not self-adaptive.

This paper overcomes limitations with existing solutions by presenting a novel solution to realize a self-adaptive D&C infrastructure to manage component-based applications for CPSS. The primary novelty of our approach lies in a choreographed D&C infrastructure, similar to a dance troupe.

We present experimental results to demonstrate application adaptability as well as self-adaptive capability of our new D&C infrastructure.

The remainder of this paper is organized as follows: Section 2 presents previous work related to this paper and explains why our approach is different; Section 3 describes the problem at hand alluding to the system model and the key challenges in realizing a self-adaptive D&C infrastructure; Section 4 presents a brief description of OMG’s D&C infrastructure, detailed description of our work by describing the overall architecture of our solution, and we also describe how our solution addresses aforementioned challenges; Section 5 presents experimental results; finally, Section 6 provides concluding remarks and alludes to future work.

2. RELATED WORK

Our work presented in this paper is related to the field of self-adaptive software systems for which a research roadmap has been well-documented in [6]. Our work falls under the general umbrella of self-adaptive systems as highlighted in the roadmap and implements all steps in the collect/analyze/decide/act loop.

In this section we compare our work specifically with existing efforts in the area of distributed software deployment, configuration, and adaptively. These existing efforts can be differentiated into two perspectives. The first being the existing research done in achieving D&C infrastructure for component-based application; and the second being the variety of work done in the field of dynamic reconfiguration of component-based applications.

2.1 Deployment and Configuration Infrastructure

Deployment and configuration of component-based software is a well-researched field with existing works primarily focusing on D&C infrastructure for grid computing and Distributed Real-time Embedded (DRE) systems. Both DeployWare [9] and GoDIET [5] are general-purpose deployment frameworks targeted towards deploying large-scale, hierarchically composed, Fractal [4] component model-based applications in a grid environment. However, both of these deployment frameworks lack autonomous resilience since neither of them support application adaptation nor self-adaptation.

DAnCE [23], LE-DAnCE [20] and F6 Deployment Manager [7] are three results from our previous work that present D&C infrastructures for DRE systems. DAnCE is an implementation of the OMG’s D&C specification itself while LE-DAnCE and F6 Deployment Manager are extensions of the OMG’s D&C specification. LE-DAnCE deploys and configures components based on the Lightweight CORBA Component Model [18] whereas the F6 Deployment Manager does the same for components based on F6-COM component model [21]. The F6 Deployment Manager, in particular, focuses on the deployment of real-time component-based applications in highly dynamic DRE systems, such as fractionated spacecraft. However, similar to the work mentioned above, these infrastructures also lack support for application adaptation and infrastructure adaptation.
2.2 Dynamic Re-configuration
A significant amount of research has been conducted in the field of dynamic reconfiguration of component-based applications. In [3], the authors present a tool called Planit for deployment and reconfiguration of component-based applications. Planit uses AI-based planner, to be more specific temporal planner, to come up with application deployment plan for both - initial deployment, and subsequent dynamic reconfigurations. Planit is based on a sense-plan-act model for fault detection, diagnosis and reconfiguration to recover from run-time application failures. Both these approaches are capable of hot deployment, that is, they both support dynamic reconfiguration; and therefore support application adaptation. However, neither of them supports a resilient adaptation engine.

Our prior work on the MADARA knowledge and reasoning engine [8] has focused on dynamic reconfiguration of DRE applications in a cloud environment. This work focuses on optimizing initial deployment and subsequent reconfiguration of distributed applications using different pluggable heuristics. Here, MADARA itself is used as an adaptation engine, however, it does not focus on resilience and therefore does not support self-adaptability.

Similarly, results presented in [26, 2, 1, 11] all support application adaptation but not resilience of the adaptation engine itself. Another work presented in [1], supports dynamic reconfiguration of applications based on J2EE components. In [11], the authors present a framework that supports multiple extensible reconfiguration algorithms for run-time adaptation of component-based applications.

Finally, in [12], the authors present a middleware that supports deployment of ubiquitous application components that are based on Fractal component model, in dynamic network. This work also supports autonomic deployment and therefore run-time application adaptation, but does not focus on resilience of the adaptation engine.

3. PROBLEM DESCRIPTION
This section describes the problem we address in this paper. First, we describe the system and the fault model. Next, we describe the problem of self-adaptation.

3.1 CPS System Model
The work described in this paper assumes a CPS that has a distributed system architecture consisting of multiple interconnected computing nodes that host distributed applications. For example, we consider a distributed system of fractionated spacecraft [7] that hosts mission-critical component-based applications with mixed criticality levels and security requirements. Fractionated spacecraft represents a highly dynamic CPS because it is a distributed system composed of nodes (individual satellites) that can join and leave a cluster at any time resulting in volatile group membership characteristics.

A distributed application in our system model is a graph of software components that are partitioned into processes and hosted within a "component" server. This graph is then mapped to interconnected computing nodes. The interaction relationships between the components are defined using established interaction patterns such as (a) synchronous and asynchronous remote method invocation, and (b) group-based publish-subscribe communication.

3.2 Deployment and Configuration Model
To deploy the distributed component-based applications of the CPS onto a target environment, the system needs to provide a software deployment service. Since we are considering a highly dynamic CPS that operates in resource-constrained environments and has severely limited availability for human intervention via remote access, we require that the software deployment service be able to adapt itself when faced with failures. In other words, it should be self-adaptive and therefore support autonomous resilience.

The Deployment and Configuration (D&C) infrastructure serves this purpose. This service is responsible for instantiating the components as processes on individual nodes, configuring their interactions, and then managing the life-cycle of the applications. The D&C infrastructure should be viewed as a distributed system composed of multiple deployment entities, called Deployment Managers (DM), with one DM residing on every node.

OMG’s D&C specification [19] is a standard for deployment and configuration of component-based application. Our prior work on the Locality-Enabled Deployment And Configuration Engine (LE-DAnCE) [20] is an open-source implementation of this specification. As shown in Figure 1, LE-DAnCE uses an orchestrated approach for software deployment and therefore implements a very strict two-layered approach for software deployment. A single orchestrator, i.e. the Cluster Deployment Manager (CDM) controls cluster-wide deployment process by coordinating deployment activities amongst different Node Deployment Managers (NDMs). Similarly, a NDM controls node-specific deployment process by instantiating required component servers, which in turn creates and manages application components.

LE-DAnCE, however, is not self-adaptive and it does not support run-time application adaptation as well. Therefore, our work presented in this paper modifies various aspects of LE-DAnCE in order to achieve a D&C infrastructure that (1) follows a choreographed approach for software deployment, (2) is capable of self-adaptation in order to support autonomous resilience, and (3) supports run-time application adaptation.

The D&C model considered for this research uses the choreographed approach to deployment and configuration. Unlike the orchestrated approach, implementing the choreographed approach results in a D&C infrastructure that performs peer-to-peer deployment with each node controlling its local deployment process. A key feature of the choreographed approach is the absence of a single orchestrator node. This

2 Components hosted within a process are located within the same address space

3 Although we use the component model described in [18], our work is not constrained by this choice and can be applied to other component models as well

4 For instance, a satellite cluster may be in range of a ground station for only 10 minutes during every 90 minute orbit
LE-DAnCE is an application adaptation scheme, which involves switching between components that provide the same functionality. Our solution presented in this paper uses this approach.

Structural adaptation refers to the adaptation scheme which changes the location of application components. It involves actions such as migration of application components from one component server to another or from one physical node to another. Our solution presented in this paper uses this approach. Structural adaptation is another application adaptation scheme, which involves switching between components that provide the same functionality. State/attribute adaptation is an application adaptation scheme, which involves making changes to a component’s state or its attributes. This approach was used in one of our previous works.

In the CPSs under consideration, we observe that in general failures can be perceived as having caused all nodes that are part of the network to have failed since those nodes become unreachable after failure, and node failure causes all the processes running on that node to fail. However, a process can fail without its host node failing and a node can fail due to reasons other than network separation. In our work, since our choreographed D&C system uses the same infrastructure elements of the CPS that can fail, the problem boils down to making the choreographed D&C engine self-adaptive thereby supporting autonomous resilience. Usually, infrastructure failures can be classified as primary failures.

Application failures are failures pertaining to the application itself. We assume that application components have been thoroughly tested before deployment and therefore classify application failures as secondary failures that are caused due to infrastructure failures. Some environmental changes could also lead to application failures, where the changes in the environment can cause an application to receive unexpected input or the environment might not react, as expected, to an application’s output. Figure 2 presents a failure propagation graph of our failure model that illustrates how failures may cascade through the system.

We do not claim that the choreographed approach is a requirement for a D&C infrastructure to be self-adaptive and resilient. An orchestrated approach can be made self-adaptive and resilient as well (for example, by using redundant orchestrators and delegating application adaptation to the active orchestrator) but we chose the choreographed approach since we argue that it is conceptually simpler to understand and extend it, even though it has significant implementation complexities as discussed in Section 4.

3.3 CPS Fault Model Related to D&C

Failure can be defined as a loss of functionality in a system. The goal of a fault management system is to ensure that subsystem or component-level faults do not lead to loss of system functionality, i.e. a failure, for an unacceptable length of time. The system is expected to recover from a failure, and the threshold on time to recovery is typically a requirement on the system. Recovering from failures involve adapting the failed subsystem such that its functionality is restored. For example in software intensive systems this process primarily involves adaptation of applications that are deployed in, and services that are provided by, the failed subsystem.

Application adaptation can be viewed in three different dimensions: (1) resource allocation adaptation, (2) structural adaptation, and (3) state/attribute adaptation. Resource allocation adaptation refers to the adaptation scheme which changes the location of application components. It involves actions such as migration of application components from one component server to another or from one physical node to another. Our solution presented in this paper uses this approach. Structural adaptation is another application adaptation scheme, which involves switching between components that provide the same functionality. State/attribute adaptation is an application adaptation scheme, which involves making changes to a component’s state or its attributes. This approach was used in one of our previous works.

In the CPSs under consideration, we observe that in general infrastructure failures can be categorized as either infrastructure failures or application failures.

Infrastructure failures are failures that arise due to faults affecting a system’s (1) network, (2) participating nodes, or (3) processes that are running in these nodes. As shown in Figure 2, there exists a causality between the three different kinds of infrastructure failures. To be more specific, network failures can be perceived as having caused all nodes that are part of the network to have failed since those nodes become unreachable after failure, and node failure causes all the processes running on that node to fail. However, a process can fail without its host node failing and a node can fail due to reasons other than network separation. In our work, since our choreographed D&C system uses the same infrastructure elements of the CPS that can fail, the problem boils down to making the choreographed D&C engine self-adaptive thereby supporting autonomous resilience. Usually, infrastructure failures can be classified as primary failures.

Application failures are failures pertaining to the application itself. We assume that application components have been thoroughly tested before deployment and therefore classify application failures as secondary failures that are caused due to infrastructure failures. Some environmental changes could also lead to application failures, where the changes in the environment can cause an application to receive unexpected input or the environment might not react, as expected, to an application’s output. Figure 2 presents a failure propagation graph of our failure model that illustrates how failures may cascade through the system.

3.4 Problem Statement

For the prescribed system and fault model, the D&C infrastructure should be capable of self-adaptation to tolerate the infrastructure failures and to manage application failures. Conceptually, a self-adaptive infrastructure can be modeled as a feedback control loop that observes the system state and compensates for disturbances in the system to achieve a desired behavior, as shown in Figure 3.
To find similarities with the traditional self-adaptive loop and the system under discussion, consider that a failure in the infrastructure or the application can be considered a disturbance. This failure can be detected by behavior such as ‘node is responding to pings’ (indicating there is no infrastructure failure), or ‘application is functioning’ (indicating there are no application failure) or not. Once the failure has been detected, the loss of the functionality needs to be restored by facilitating reconfiguration, e.g. re-allocating components to a functioning node, etc. The presence of the controller and its actuation ability enables the self-adaptive property needed of an autonomous resilient system.

4. SELF-ADAPTIVE D&C INFRASTRUCTURE

Figure 4 shows the outline of our solution. Infrastructure failures are detected using the Group Membership Monitor (GMM). Section 4.1 describes, in detail, how the GMM works. Application failure detection is outside the scope of this paper, however, we refer readers to our earlier work [16] in this area. The controller is in fact a collection of deployment managers working together as an adaptation engine to restore functionality when failures are detected. Specific actuation commands are redeployment actions taken by the deployment managers.

We discuss the specifics of this adaption engine next. Next we present the key challenges in realizing the self-adaptive properties for such an architecture. Finally, we describe how our approach is addressing these challenges.

4.1 An Architecture for Self-adaptive, Choreographed D&C

Figure 5 presents the architecture of our self-adaptive D&C infrastructure. Each node consists of a single Deployment Manager (DM) which, if required, spawns one or more Component Servers (CSs). These CSs are processes that are responsible for managing the lifecycle of application components. When compared to the architecture of existing D&C infrastructures like DAnCE and LE-DAnCE (See Figure 1), we can observe that this architecture lacks a central orchestrator as it follows a choreographed approach for software deployment and configuration where DMs use a publish/subscribe middleware to communicate with each other.

In our architecture, we use GMM for two things - (1) maintaining up-to-date group member information, and (2) detecting failure via periodic heartbeat monitoring mechanism. Failure detection aspect of GMM relies on two important parameters - heartbeat period and failure monitoring period. These parameters are configurable. Configuring heartbeat period allows us to control how often each DM assert their liveness, whereas configuring failure monitoring period allows us to control how often each DM triggers their fault monitoring mechanism and what is the worst case latency when a missed heartbeat will be detected.

For a given failure monitoring period, lower heartbeat period results in higher network traffic but lower failure detection latency, whereas higher heartbeat period results in lower network traffic but higher failure detection latency.

Tuning these parameters appropriately can also enable the architecture to tolerate intermittent failures where a few heartbeats are only missed for a few cycles and are established later. This can be done by making the fault monitoring window much larger compared to the heart beat period.

Figure 6 presents an event diagram demonstrating a three node choreographed deployment process of our new D&C infrastructure. As seen from the figure, an application deployment is initiated by submitting a global deployment plan to one of the three DMs. This global deployment plan contains information about different components (and their implementation) that make up an application. It also contains information about how different components should be connected. Once this global deployment plan is received by a DM, that particular DM becomes the deployment leader for that particular deployment plan. Two different global deployment plans can be deployed by two different deployment leaders; we do not have a static leader that controls deployment of all applications.
Table 1: D&C Stages

<table>
<thead>
<tr>
<th>Stage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INITIAL</td>
<td>(1) Global deployment plan is provided to one of the DMs.</td>
</tr>
<tr>
<td></td>
<td>(2) DM that is provided with a global deployment plan becomes the leader DM and stores it in a binary format.</td>
</tr>
<tr>
<td>PREPARING</td>
<td>(1) Plan loaded in previous stage is split into node-specific plans and published to the distributed data space using pub/sub middleware.</td>
</tr>
<tr>
<td></td>
<td>(2) Node-specific plans published above are received by respective DMs, which in turn further split the node-specific plans into component server (CS)-specific plans.</td>
</tr>
<tr>
<td>STARTING</td>
<td>(1) CS-specific plans created in previous stage are used to create CSs (if required) and components.</td>
</tr>
<tr>
<td></td>
<td>(2) For components that provide service via a facet, the DM will publish its connection information so that other components that require this service can connect to it using their receptacle. This connection however is not established in this stage.</td>
</tr>
<tr>
<td></td>
<td>(3) In this stage, barrier synchronization is performed to make sure that no individual DMs can advance to the next stage before all of the DMs have reached this point.</td>
</tr>
<tr>
<td>FINISHING</td>
<td>(1) Components created in the previous stage are connected (if required). In order for this to happen, the components that require a service use connection information provided in the previous state to make facets-receptacle connections.</td>
</tr>
<tr>
<td>ACTIVATING</td>
<td>(1) Synchronization stage to make sure all components are created and connected (if required) before activation.</td>
</tr>
<tr>
<td>ACTIVATED</td>
<td>(1) Stage where a deployment plan is activated by activating all the related components.</td>
</tr>
<tr>
<td></td>
<td>(2) At this point all application components are running.</td>
</tr>
<tr>
<td>TEARDOWN</td>
<td>(1) De-activation stage.</td>
</tr>
</tbody>
</table>

![Deployment Diagram](image_url)

Figure 6: A Three-node Choreographed Deployment

Deployment and configuration in the choreographed approach follows a staged approach. Table 1 lists the different D&C stages in our approach. The INITIAL stage is where a deployment plan gets submitted to a DM.

4.2 Challenges

To correctly provide self-adaptive choreographed D&C services to a CPS cluster, the D&C infrastructure must resolve a number of challenges that are not well addressed by traditional orchestrated deployment approaches. These challenges are described below referring to the desired self-adaptive capability shown in Figure 7. This figure illustrates the desired steps when, for example, using the scenario presented in Figure 6, we assume that node-2 was to fail, which in turn implies that DM-2 has failed during the deployment process.

4.2.1 Challenge 1: Distributed Group Membership

4.2.2 Challenge 2: Leader Election

Recall that the CPS domain illustrates a highly dynamic environment in terms of resources that are available for application deployment: nodes may leave unexpectedly as a result of a failure, as part of a planned partitioning of the cluster, and nodes may also join the cluster as they recover from faults or are brought online. To provide resilient behavior, DMs in the cluster must be aware of changes in group membership, i.e., they must be able to detect when one of their peers has left the group (either as a result of a fault or planned partitioning) and when new peers join the cluster. In our example, Now, in order to recover from this failure, the remaining two DMs must collaborate to perform a failure recovery mechanism by adapting the D&C infrastructure as shown in Figure 7. For that the remaining DMs must detect the failure and group membership changes. Section 4.3.1 describes our solution to address this challenge.
As faults occur in CPSs, a resilient system must make definitive decisions about the nature of that fault and the best course of action necessary to mitigate and recover from that fault. Since CPS clusters often operate in mission- or safety-critical environments where delayed reaction to faults can severely compromise the safety of the cluster, such decisions must be made in a timely manner. In order to accommodate this requirement, the D&C infrastructure must elect a “leader” that will be responsible for making decisions and performing other tasks that impact the entire cluster. Since the node that is elected as a leader itself may fail, such leader election should recur each time the group membership changes. Once the failure is detected, both DMs first check to see if they are the deployment leader and only the deployment leader takes recovery measures by redeploying application parts affected by the failure to a healthy node. In our example, since DM-1 is the deployment leader, it should take appropriate actions to adapt the system. On the contrary, if DM-2 were the original leader, then a new leader must be formed among the remaining DMs. Section 4.3.2 describes our solution to address this challenge.

4.2.3 Challenge 3: Proper Sequencing of Deployment
Applications in CPS may be composed of several cooperating components with complex internal dependencies that are distributed across several nodes. Deployment of such an application requires that deployment activities across several nodes proceed in a synchronized manner. For example, connections between two dependent components cannot be established until both components have been successfully instantiated. Moreover, components should not be activated until all other components in the application are prepared for activation. Section 4.3.3 describes our solution to address this challenge.

4.2.4 Challenge 4: D&C State Preservation
Nodes in a CPS may fail at any time and for any reason; a D&C infrastructure capable of supporting such a cluster must be able to reconstitute the portions of the distributed application deployed on the failed node. Supporting self-adaptation requires the D&C infrastructure to keep track of the global system state, which consists of (1) component-to-application mapping, (2) component-to-node mapping, (3) inter-component connection information, (4) component state information, and (5) the current group membership information. Such state preservation is especially important for the currently elected cluster leader. In our example, since DM-1 is the deployment leader, it is responsible for determining the parts of application that were previously deployed by DM-2 on node-2 and redeploy them to a healthy node (DM-3). Section 4.3.4 describes our solution to address this challenge.

4.3 Addressing Self-adaptive D&C Challenges
We now discuss how our architecture resolves the key challenges identified in Section 4.2.

4.3.1 Resolving Challenge 1: Distributed Group Membership
To support distributed group membership, our solution requires a mechanism that allows detection of joining members and leaving members. To that end our solution uses a discovery mechanism to detect the former and a failure detection mechanism to detect the latter described below.

Discovery Mechanism: Since our choreographed approach relies on an underlying pub/sub middleware, the discovery of nodes joining the cluster leverages existing discovery services provided by the pub/sub middleware. To that end we have used OpenDDS (http://www.opendds.org) – an open source pub/sub middleware that implements OMG’s Data Distribution Service (DDS) specification [17]. To be more specific, we use the Real-Time Publish Subscribe (RTPS) peer-to-peer discovery mechanism supported by OpenDDS.

Failure Detection Mechanism: To detect the loss of existing members, we need a failure detection mechanism that detects different kinds of failures described in Section 3.3. In our architecture this functionality is provided by the GMM. GMM residing on each node uses simple heartbeat-based protocol to detect DM (process) failure. Recall that any node failure, including the ones caused due to network failure, results in the failure of its DM. This means that our failure detection service uses the same mechanism to detect all three different classes of infrastructure failures.

Upon failure detection, as shown in Figure 7, only the lead DM takes recovery action which involves redeployment of application portion that was previously deployed in the failed node. In our current implementation, the lead DM determines redeployment location by checking for nodes that do not have any previously deployed applications. However, more advanced heuristics could be used to determine redeployment location based on different metrics such as available resources and application collocation requirements. We identify MADARA [8] as a tool that could be used to implement these advanced heuristics since it provides a distributed knowledge sharing and reasoning middleware.

4.3.2 Resolving Challenge 2: Leader Election
Our current implementation supports a rudimentary leader-election strategy based on a unique identifier assigned to each DM: from among the functioning nodes the one with the highest id is elected. Moreover, leader designation is initiated by an external entity, such as a ground station of the fractionated spacecraft cluster. Depending on which DM is given the initial global deployment plan, a leader can be manually changed from one deployment to another. However, this approach obviously does not support an event-triggered leader election, such as when the node hosting the DM fails. Implementing a robust leader election algorithm is part of our future work. To that end, we are evaluating and extending existing algorithms [15, 27] to suit our requirements.

4.3.3 Resolving Challenge 3: Proper Sequencing of Deployment
Our D&C infrastructure implements deployment synchronization using a distributed barrier synchronization algo-
This mechanism is specifically used during the STARTING stage of the D&C process to make sure that all DMs are in the STARTING stage before any of them can advance to the FINISHING stage. This synchronization is performed to ensure that all connection information of all the components that provide a service is published to the distributed data space before components that require a service try to establish a connection.

In addition, our current solution also uses barrier synchronization in the ACTIVATING stage to make sure all DMs advance to the ACTIVATED stage simultaneously. This particular synchronization ensures the simultaneous activation of a distributed application.

4.3.4 Resolving Challenge 4: D&C State Preservation

In our current implementation, only the leader DM stores information about where different application components have been deployed and how these components are connected to each other. In some sense, this is similar to the orchestrated approach. For our rudimentary leader election strategy, this simple solution is sufficient since our external source that elects the leader also preserves the state. We are currently working on a more dynamic approach by allowing each and every DM to follow a distributed checkpointing mechanism in order to simultaneously checkpoint and store their state such that in case of any failure they can either themselves rollback to their previously known good state or if that is not possible then the load DM can use a failed DM’s state information to restart the DM and advance it to its previously known good state. To achieve this, we foresee using an approach which will include the durability QoS option of DDS pub/sub or implement something similar, which will allow persistent storage of DM states across different nodes.

5. EXPERIMENTAL RESULTS

This section presents results of empirical studies evaluating the self-adaptive capabilities of our D&C infrastructure. First we present time sequence graphs to show how our choreographed D&C infrastructure adapts itself to tolerate failures during deployment-time and as well as itself after encountering a node failure during (1) application deployment-time, and (2) application run-time. Second, we present a discussion section for performance comparison between LE-DAnCE and our solution.

5.1 Testbed

For all of our experiments, we used a multi-computing node cluster setup that consisted of three nodes, each with a 1.6 GHz Atom N270 processor and 1GB of RAM. Each node runs vanilla Ubuntu server image 13.04 which uses Linux kernel version 3.8.0-19.

The application we used for self-adaptability experiments presented in Sections 5.2 and 5.3 is a simple two-component client-server experiment presented earlier in Figure 5. The Sender component (client) is initially deployed in node-1, the Receiver component (server) is initially deployed in node-2, and node-3 has nothing deployed on it. For both experiments, we consider node-2 to be the node that fails. Furthermore, we configure our infrastructure with heartbeat period set to 2 seconds and failure monitoring period set to 5 seconds.

5.2 Node Failure During Deployment-time

Figure 8 presents a time sequence graph of how our D&C infrastructure adapts itself to tolerate failures during deployment-time. As seen from the figure, node-2 and therefore DM-2 fails at Event #5. Once the failure is detected by both DM-1 in node-1 and DM-3 in node-3, DM-1 being the leader initiates the recovery process (Event #6 - Event #7). During this time, DM-1 determines the part of the application that was supposed to be deployed by DM-2 in node-2, which is the Receiver component. Once DM-1 determines this information, it completes the recovery process by republishing information about the failure affected part of application (Receiver component) to DM-3. Finally, DM-3 deploys the Receiver component in node-3 and after this point, the deployment process resumes normally.

5.3 Node Failure During Application Run-time

Figure 9 presents another time sequence graph that demonstrates how our D&C infrastructure adapts applications at run-time to tolerate run-time node failures. Unlike the scenario presented before where the initial deployment of the application has to be adapted to tolerate deployment-time
failure, here the initial deployment completes successfully at Event #19 after which the application is active. However, node-2 and therefore DM-2 fails at Event #20 and the notification of this failure is received by DM-1 at Event #21 after which DM-1 performs the recovery process almost exactly the same way like it did for deployment-time failure.

The one significant difference between the deployment-time failure mitigation and run-time failure mitigation is that dynamic reconfiguration of application components is required to mitigate application run-time failure. To elaborate, once DM-3 deploys the Receiver component in node-3 it needs to publish new connection information for the Receiver component allowing DM-1 to update Sender the component’s connection.

5.4 Discussion
During our experiments, we discovered that the deployment latency of LE-DAnCE is lower than the solution described in this paper. For example, using LE-DAnCE, an application consisting of 10 components can be deployed in 0.27 seconds, whereas performing the same task with our solution architecture takes 2.33 seconds. There are two factors that contribute to this result. First, this isn’t fair comparison: LE-DAnCE is a mature and highly optimized tool-chain, with particular focus on reducing deployment latency [22], while the approach under discussion is still a research tool with focus on functionality rather than performance.

Second, this result helps us to explore the tradeoffs between choreographed and orchestrated approach for software deployment and configuration. Furthermore, it also provides an opportunity to identify the existing performance bottlenecks so that we can optimize them in our future work. The orchestrated approach (embodied by LE-DAnCE) is relatively easy to optimize; no group-based consensus is necessary, the CDM simply instructs the NDMs when to proceed with each phase of deployment. It is, however, less flexible when compared to the choreographed approach; we believe that attempting to add resilience to this architecture would compromise the conceptual simplicity and ability to easily optimize. The increased flexibility of the choreographed approach, and the fact that resilience is an intrinsic property of this approach, more than compensates for any discrepancies in deployment latency. Moreover, much of this additional overhead could later be optimized.

6. CONCLUSIONS AND FUTURE WORK
This paper described a self-adaptive Deployment and Configuration (D&C) infrastructure for highly dynamic CPS. This nature of CPS and the infeasibility of human intervention calls for autonomous resilience in such systems. The D&C infrastructure is the right artifact to architect such a solution because of the increasing trend towards component-based CPS. To that end we showed an approach that uses a decentralized, choreographed approach to self-adaptive D&C. Experimental results presented in this paper support our claim regarding the notion that D&C infrastructures can be used as adaptation engines to support autonomous resilience.

The work presented in this paper incurs a few limitations: (1) the failure detection mechanism presented in Section 4.3.1 is not robust enough to handle byzantine failures where a failure might be wrongly reported by some of the members of a group. In order to handle this scenario, we will extend the existing failure detection mechanism by using Paxos [13] as a mechanism to achieve distributed consensus before taking any failure mitigation actions; and (2) As mentioned in Section 4.3.4, our current implementation for DM state preservation is sufficient but not ideal. However, achieving our ideal goal requires significant amount of additional work and hence forms the contours of our future work.

Source code for all of our work presented in this paper can be made available upon request.

7. REFERENCES


