Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, 37203

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

Subhav Pradhan, William Otte, Abhishek Dubey, Csanad Szabo, Aniruddha Gokhale, Gabor Karsai

TECHNICAL REPORT

ISIS-14-102

6-19-2014

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 lifecycle of software applications. Fluctuating resources, volatile cluster membership and changing environmental conditions require resilience. However, due to the dynamic nature of the system, human intervention is often infeasible. This necessitates a self-adaptive D&C infrastructure that supports autonomous resilience. Such an infrastructure must have the ability to adapt existing applications on the fly in order 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, D&C infrastructure for CPSs. Previous efforts in this area have resulted in D&C infrastructures that support 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.

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

Subhav Pradhan William Otte Abhishek Dubey Csanad Szabo
Aniruddha Gokhale Gabor Karsai

Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN 37203, USA

I. 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 which represents the target domain of our work presented in this paper. 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, human intervention is extremely limited therefore resilience should be autonomic. Consequently, the system should be self-adaptive [1] 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.

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) [2], 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.

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 [3]. However, existing D&C infrastructures 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* [4], these solutions are not self-adaptive.

This paper overcomes limitations of 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 work presented in this paper lies in a new D&C infrastructure which is self-adaptive and therefore supports autonomous resilience. We have identified the following as contributions of this paper:

- We present the key challenges in achieving a self-adaptive, D&C infrastructure for highly dynamic CPSs,
- We present an architecture for a self-adaptive D&C infrastructure for component-based applications that addresses these key challenges, and
- 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 II presents previous work related to this paper and explains why our approach is different; Section III describes the problem at hand alluding to the system model and the key challenges in realizing a self-adaptive D&C infrastructure; Section IV presents detailed description of our work by describing the overall architecture of our solution, and we also describe how our solution addresses aforementioned challenges; Section V presents experimental results; finally, Section VI provides concluding remarks and alludes to future work.

II. 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 [5]. 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 adaptivity. 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.

A. 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** [6] and **GoDIET** [7] are general-purpose deployment frameworks targeted towards deploying large-scale, hierarchically composed, Fractal [8] 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.

The Object Management Group (OMG) has standardized the Deployment and Configuration (D&C) specification [9]. Our prior work on the Deployment And Configuration Engine (**DAnCE**) [10], [11] describes a concrete realization of the OMG D&C specification for the Lightweight CORBA Component Model (LwCCM) [12]. **LE-DAnCE** [11] and **F6 DeploymentManager** [13] are some of our other previous works that extends the OMG's D&C specification. LE-DAnCE deploys and configures components based on the Lightweight CORBA Component Model [12] whereas the F6 Deployment Manager does the same for components based on F6-COM component model [14]. The F6 Deployment Manager, in particular, focused 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 D&C infrastructure adaptation.

B. Dynamic Re-configuration

A significant amount of research has been conducted in the field of dynamic reconfiguration of component-based applications. In [15], 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 [16] 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 [17], [18], [19], [20] all support application adaptation but not resilience of the adaptation engine itself. Another work presented in [19], supports dynamic reconfiguration of applications based on J2EE components. In [20], the authors present a framework that supports multiple extensible reconfiguration algorithms for run-time adaptation of component-based applications.

Finally, in [21], 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.

III. Problem Description

This section describes the problem at hand by first presenting the target system model. Second, we present the Deployment and Configuration (D&C) model. Third, we present the fault model related to system model. Finally, we describe the problem of self-adaptation in context of the D&C infrastructure.

A. CPS System Model

The work described in this paper assumes a distributed CPS consisting of multiple interconnected computing nodes that host distributed applications. For example, we consider a distributed system of fractionated spacecraft [13] 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[1] and hosted within a "component" server. This graph is then mapped to interconnected computing nodes. The interaction relationships between the components are defined using

[1]Components hosted within a process are located within the same address space

established interaction patterns such as (a) synchronous and asynchronous remote method invocation, and (b) group-based publish-subscribe communication.

B. Deployment and Configuration Model

To deploy distributed component-based applications [2] 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[3], 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.

A Deployment and Configuration (D&C) infrastructure serves this purpose; it is responsible for instantiating application components on individual nodes, configuring their interactions, and then managing their lifecycle. 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 each node.

OMG's D&C specification [9] is a standard for deployment and configuration of component-based application. Our prior work on the Locality-Enabled Deployment And Configuration Engine (LE-DAnCE) [11] is an open-source implementation of this specification. As shown in Figure 1, LE-DAnCE 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.

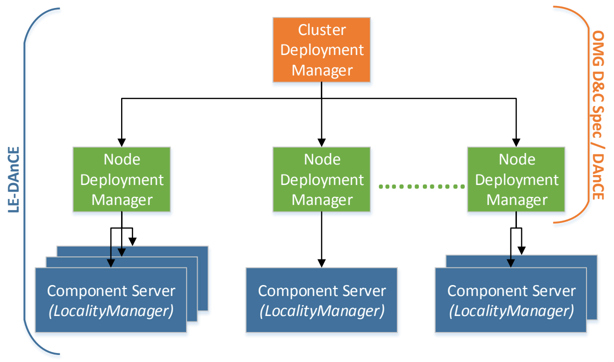
Cluster Deployment Manager
Node Deployment Manager
Node Deployment Manager
Node Deployment Manager
Component Server (LocalityManager)
Component Server (LocalityManager)
Component Server (LocalityManager)
LE-DAnCE
OMG D&C Spec / DAnCE

Fig. 1. Orchestrated Deployment Approach in LE-DAnCE [11]

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) is capable of self-adaptation in order to support autonomous resilience, and (2) supports run-time application adaptation.

[2]Although we use the component model described in [12], our work is not constrained by this choice and can be applied to other component models as well

[3]For instance, a satellite cluster may be in range of a ground station for only 10 minutes during every 90 minute orbit

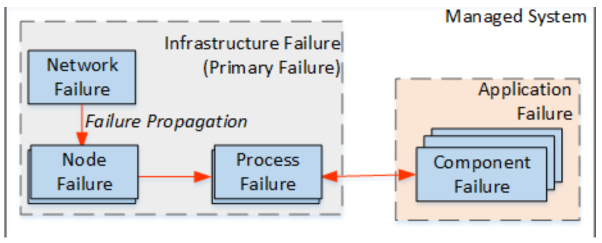

Fig. 2. Potential failure propagations.

C. 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 [22].

In the CPSs under consideration, we observe that in general the subsystem 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 the D&C engine uses infrastructural elements of

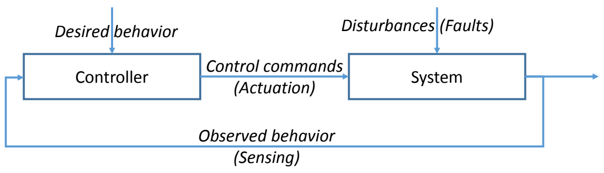

Fig. 3. Self-adaptive System as a Control System

the CPS that can fail, the problem boils down to making the 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.

D. 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 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 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.

IV. Self-adaptive D&C Infrastructure

Figure 4 shows the outline of our solution. Infrastructure failures are **detected** using the *Group Membership Monitor* (GMM). Section IV-A 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 [22] 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.

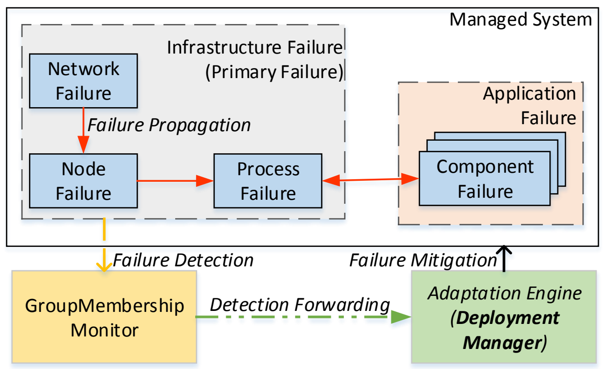

Fig. 4. Self-adaptive Deployment and Configuration Architecture.

We discuss the specifics of this adaption engine next. Then 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.

A. An Architecture for Self-adaptive D&C

Figure 5 presents the architecture of our self-adaptive D&C infrastructure. Each node consists of a single Deployment Manager (DM) and collection of these DMs forms the overall D&C infrastructure. Our solution D&C infrastructure uses a choreographed approach to D&C of applications. This results in an infrastructure that supports distributed, peer-to-peer application deployment, where each node controls its local deployment process.

Each DM, if required, spawns one or more Component Servers (CSs). These CSs are processes that are responsible for managing 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 are independent and 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 liveliness, 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

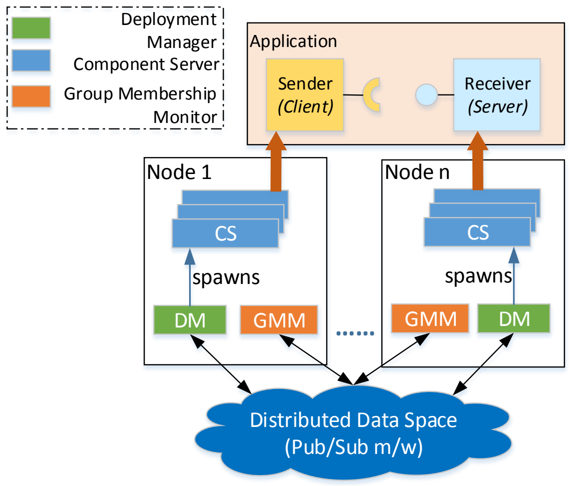

Fig. 5. Self-adaptive D&C architecture

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 deployment process of our new D&C infrastructure.

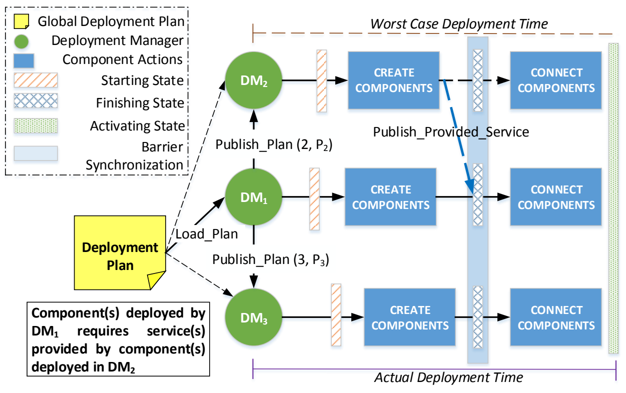

Fig. 6. A Three-node Choreographed Deployment

As seen from the figure, an application deployment is initiated by submitting a *global deployment plan* to one of the three DMs in the system. 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* [4] for that particular deployment plan. Two different global deployment plans can be deployed by two different deployment leaders; we do not have a static orchestrator that governs D&C of different applications.

[4]A deployment leader is only responsible for initiating the deployment process for a given deployment plan by analyzing the plan and allocating deployment actions to other DMs in the system. The deployment leader is not responsible for other cluster-wide operations such as *failure mitigation*; these cluster-wide operations are handled by a *cluster leader*.

Deployment and configuration in the choreographed approach follows a multi-staged approach. Table I lists the different D&C stages in our approach. The `INITIAL` stage is where a deployment plan gets submitted to a DM and ACTIVATED stage is where the application components in the deployment plan is active.

B. 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 there is a node failure. For example, in the case of scenario presented in Figure 6 where DM-1 is the deployment leader as well as the cluster leader, if we assume that node-2 fails at some point during deployment process, it implies that DM-2 has failed too. This DM failure is detected by other DMs and in order to handle this failure the D&C infrastructure, which now only consists of DM-1 and DM-2, needs to self-adapt.

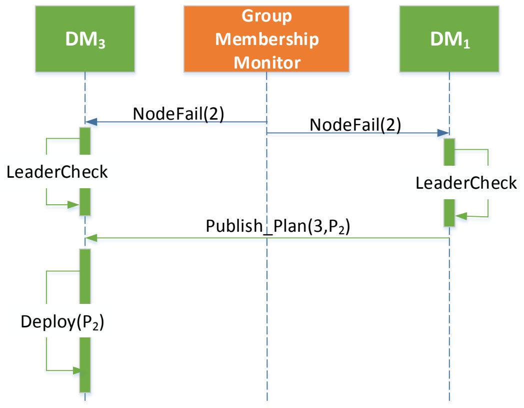

Fig. 7. Desired Self-adaptation Sequence Diagram

1) Challenge 1: Distributed Group Membership: 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 or as part of a planned or unplanned 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, shown in figure 6, if one out of three node fails at any point, then the DMs in the remaining two nodes should be able to detect this failure based on

TABLE I

D&C STAGES

Stage	Description
INITIAL	(1) Global deployment plan is provided to one of the DMs. (2) DM that is provided with a global deployment plan becomes the leader DM and loads that deployment plan and stores it in a binary format.
PREPARING	(1) Plan loaded in previous stage is split into node-specific plans and published to the distributed data space using pub/sub middleware. (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.
STARTING	(1) CS-specific plans created in previous stage are used to create CSs (if required) and components. (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. (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.
FINISHING	(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 facet-receptacle connections.
ACTIVATING	(1) Synchronization stage to make sure all components are created and connected (if required) before activation.
ACTIVATED	(1) Stage where a deployment plan is activated by activating all the related components. (2) At this point all application components are running.
TEARDOWN	(1) De-activation stage.

group membership information and they must collaborate to perform a failure recovery mechanism by adapting the D&C infrastructure as shown in Figure 7. Section IV-C.1 describes our solution to address this challenge.

2) Challenge 2: Leader Election: 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 system should always have a *cluster leader* that will be responsible for making decisions and performing other tasks that impact the entire cluster[5]. However, a node that hosts the DM acting as the cluster leader can fail at any time; in this scenario, the remaining DMs in the system should decide amongst themselves regarding who the new cluster leader should be. This process needs to be facilitated by a leader election algorithm.

Again, going back to our example shown in figure 6 where DM-1 is the cluster leader and node-2 fails causing DM-2 to fail, there is no requirement for electing a new leader since DM-1 is alive and therefore can handle failure mitigation process by redeploying application parts affected by node-2 failure. However, if DM-1 fails instead of DM-2 then, DM-2 and DM-3 should be able to run the leader election algorithm and elect, amongst themselves, a new leader who can then handle the failure mitigation process. Section IV-C.2 describes our solution to address this challenge.

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. Depending on the application, some might require stronger sequencing semantics whereby all components of the application need to be activated simultaneously.

In our example scenario shown in figure 6, we require two synchronization points. The first one to make sure that all components of an application are created before the D&C infrastructure tries to establish connections between components that require a service and components that provide the service. We require the second synchronization point to make sure all components of an application is activated simultaneously. Section IV-C.3 describes our solution to address this challenge.

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, (3) component-to-implementation mapping[6], (2) component-to-node mapping, (3) inter-component connection information, (4) component state information, and (5) the current group

[5]Achieving a consensus-based agreement for each adaptation decision would likely be inefficient and violate the real-time constraints of the cluster

[6]A component could possibly have more than one implementation available.

membership information. Such state preservation is especially important for a newly elected cluster leader.

In our example, since DM-1 is the cluster 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 (node-3). Section IV-C.4 describes our solution to address this challenge.

C. Addressing Self-adaptive D&C Challenges

We now discuss how our architecture resolves the key challenges identified in Section IV-B.

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 solution 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 [23]. 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 III-C. 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 cluster leader 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 [16] as a tool that could be used to implement these advanced heuristics since it is a distributed knowledge sharing and reasoning middleware.

2) Resolving Challenge 2: Leader Election: Implementing a robust leader election algorithm is part of our future work. Our current implementation does not include the notion of a cluster leader; we only have the notion of deployment leaders where the system can only have one deployment leader at a given time. A DM that is provided with a deployment plan becomes the deployment leader for that plan and two different deployment plans can be deployed by two different deployment leaders. We also assume that a deployment leader doesn't fail since it is the only DM in the system that stores the deployment plan. We can easily overcome this particular restriction by storing redundant copies of deployment plans as further explained in Section IV-C.4.

One of our immediate future goal is to implement a robust leader election algorithm such that the D&C infrastructure can handle deployment leader failures. To that end, we are evaluating and extending existing algorithms [24], [25] to suit our requirements.

3) Resolving Challenge 3: Proper Sequencing of Deployment: Our D&C infrastructure implements deployment synchronization using a distributed *barrier synchronization* algorithm. 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. We realize that this might be too strong of a requirement and therefore we intend to further relax this requirement by making sure that only components that require a service wait for synchronization.

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. However, it is entirely possible that an application does not care about simultaneous activation and therefore does not require this synchronization.

4) Resolving Challenge 4: D&C State Preservation: In our current state of implementation, for a single deployment plan, only the DM that is provided with this deployment plan - i.e. the deployment leader - stores the plan and all other DMs that part-take in deployment of that plan only know about node-specific sub-plans provided to them by the deployment leader. This means that our current implementation is not robust enough to handle deployment leader failures.

In future, we need to design an efficient mechanism using which all of the DMs in the system store every single deployment plan provided to different deployment leaders such that we have enough redundancy to handle deployment leader failures. In addition to storing redundant deployment plans across all the DMs, we also need to efficiently store different component's state information. This will also be part of our future work.

V. 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 applications 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.

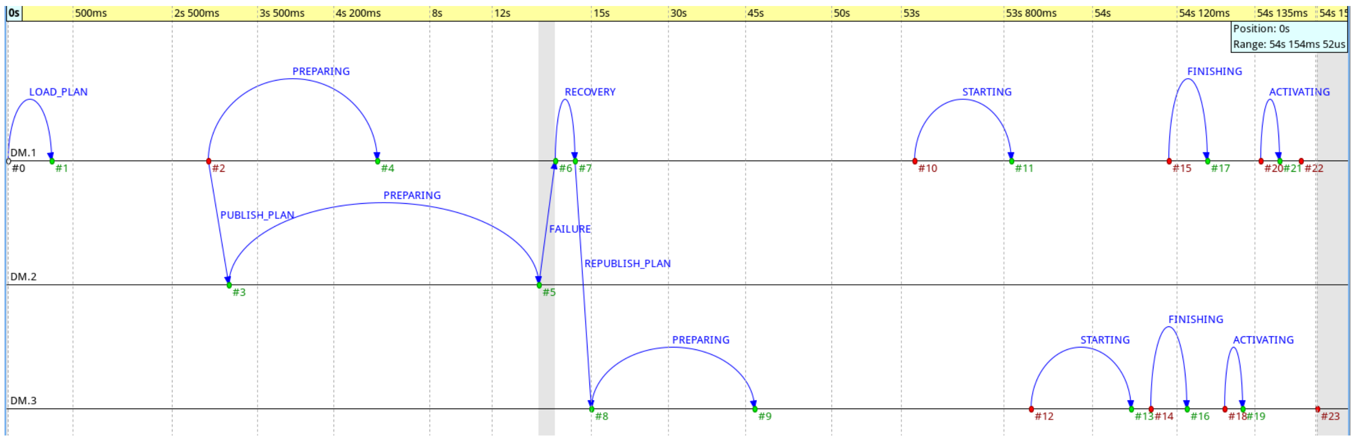

Fig. 8. Node Failure During Deployment-time

A. 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 V-B and V-C 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.

B. 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.

C. 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.

VI. 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 IV-C.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* [26] as a mechanism to achieve distributed consensus before taking any failure mitigation actions; and (2) As mentioned in Section IV-C.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.

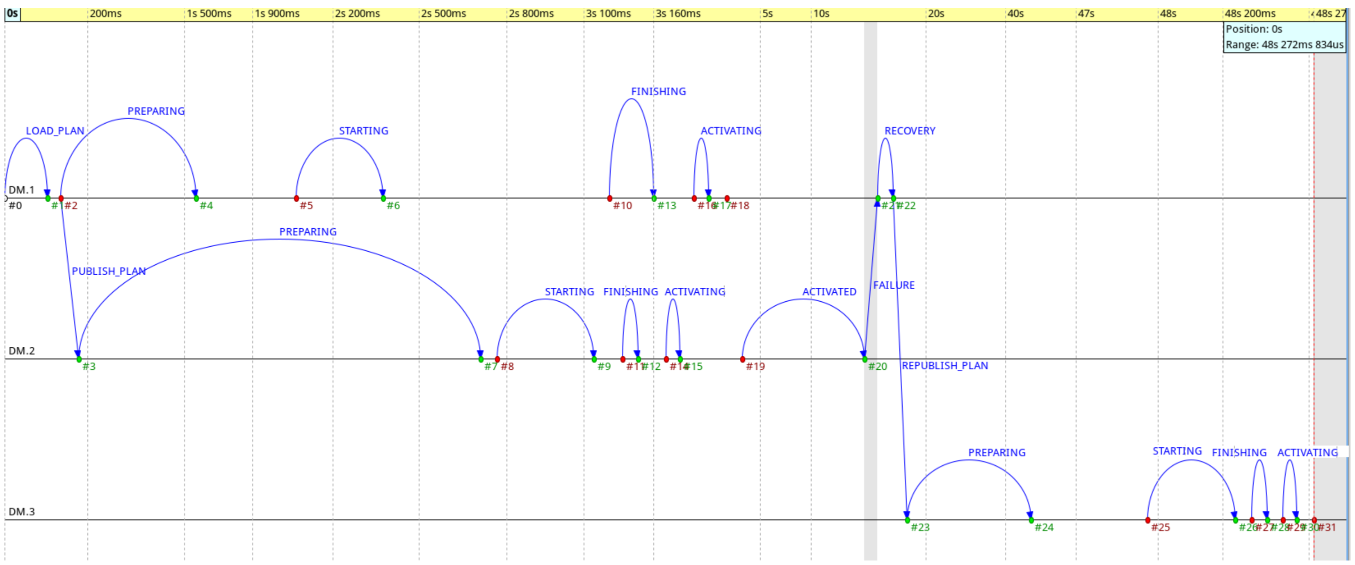

Fig. 9. Node Failure During Application Run-time

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

References

[1] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 4, no. 2, p. 14, 2009.

[2] G. T. Heineman and W. T. Councill, Eds., *Component-based Software Engineering: Putting the Pieces Together*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.

[3] S. Pradhan, A. Gokhale, W. Otte, and G. Karsai, "Real-time Fault-tolerant Deployment and Configuration Framework for Cyber Physical Systems," in *Proceedings of the Work-in-Progress Session at the 33rd IEEE Real-time Systems Symposium (RTSS '12)*. San Juan, Puerto Rico, USA: IEEE, Dec. 2012.

[4] Y. D. Liu and S. F. Smith, "A formal framework for component deployment," in *ACM SIGPLAN Notices*, vol. 41, no. 10. ACM, 2006, pp. 325–344.

[5] B. H. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic *et al.*, *Software engineering for self-adaptive systems: A research roadmap*. Springer, 2009.

[6] A. Flissi, J. Dubus, N. Dolet, and P. Merle, "Deploying on the grid with deployware," in *Cluster Computing and the Grid, 2008. CCGRID'08. 8th IEEE International Symposium on*. IEEE, 2008, pp. 177–184.

[7] E. Caron, P. K. Chouhan, H. Dail *et al.*, "Godiet: a deployment tool for distributed middleware on grid'5000," 2006.

[8] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.-B. Stefani, "The fractal component model and its support in java," *Software: Practice and Experience*, vol. 36, no. 11-12, pp. 1257–1284, 2006.

[9] OMG, "Deployment and Configuration Final Adopted Specification." [Online]. Available: http://www.omg.org/members/cgi-bin/doc?ptc/03-07-08.pdf

[10] W. R. Otte, D. C. Schmidt, and A. Gokhale, "Towards an Adaptive Deployment and Configuration Framework for Component-based Distributed Systems," in *Proceedings of the 9th Workshop on Adaptive and Reflective Middleware (ARM '10)*, Bengarulu, India, Nov. 2010.

[11] W. Otte, A. Gokhale, and D. Schmidt, "Predictable deployment in component-based enterprise distributed real-time and embedded systems," in *Proceedings of the 14th international ACM Sigsoft symposium on Component based software engineering*. ACM, 2011, pp. 21–30.

[12] *Light Weight CORBA Component Model Revised Submission*, OMG Document realtime/03-05-05 ed., Object Management Group, May 2003.

[13] A. Dubey, W. Emfinger, A. Gokhale, G. Karsai, W. Otte, J. Parsons, C. Szabo, A. Coglio, E. Smith, and P. Bose, "A Software Platform for Fractionated Spacecraft," in *Proceedings of the IEEE Aerospace Conference, 2012*. Big Sky, MT, USA: IEEE, Mar. 2012, pp. 1–20.

[14] W. R. Otte, A. Dubey, S. Pradhan, P. Patil, A. Gokhale, G. Karsai, and J. Willemsen, "F6COM: A Component Model for Resource-Constrained and Dynamic Space-Based Computing Environment," in *Proceedings of the 16th IEEE International Symposium on Object-oriented Real-time Distributed Computing (ISORC '13)*, Paderborn, Germany, Jun. 2013.

[15] N. Arshad, D. Heimbigner, and A. L. Wolf, "Deployment and dynamic reconfiguration planning for distributed software systems," in *Tools with Artificial Intelligence, 2003. Proceedings. 15th IEEE International Conference on*. IEEE, 2003, pp. 39–46.

[16] J. Edmondson, A. Gokhale, and D. Schmidt, "Approximation techniques for maintaining real-time deployments informed by user-provided dataflows within a cloud," *Reliable Distributed Systems, IEEE Symposium on*, vol. 0, pp. 372–377, 2012.

[17] N. Shankaran, J. Balasubramanian, D. Schmidt, G. Biswas, P. Lardieri, E. Mulholland, and T. Damiano, "A framework for (re) deploying components in distributed real-time and embedded systems," in *Proceedings of the 2006 ACM symposium on Applied computing*. ACM, 2006, pp. 737–738.

[18] S. S. Andrade and R. J. de Araújo Macêdo, "A non-intrusive component-based approach for deploying unanticipated self-management behaviour," in *Software Engineering for Adaptive and Self-Managing Systems, 2009. SEAMS'09. ICSE Workshop on*. IEEE, 2009, pp. 152–161.

[19] A. Akkerman, A. Totok, and V. Karamcheti, *Infrastructure for automatic dynamic deployment of J2EE applications in distributed environments*. Springer, 2005.

[20] J. Hillman and I. Warren, "An open framework for dynamic reconfiguration," in *Proceedings of the 26th International Conference on Software Engineering*. IEEE Computer Society, 2004, pp. 594–603.

[21] D. Hoareau and Y. Mahéo, "Middleware support for the deployment of ubiquitous software components," *Personal and ubiquitous computing*, vol. 12, no. 2, pp. 167–178, 2008.

[22] N. Mahadevan, A. Dubey, and G. Karsai, "Application of software health management techniques," in *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. ACM, 2011, pp. 1–10.

[23] *Data Distribution Service for Real-time Systems Specification*, 1.0 ed., Object Management Group, Mar. 2003.

[24] N. A. Lynch, *Distributed Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996.

[25] S. Vasudevan, B. DeCleene, N. Immerman, J. Kurose, and D. Towsley, "Leader election algorithms for wireless ad hoc networks," in *Proceed-*

ings of DARPA Information Survivability Conference and Exposition, vol. 1, 2003, pp. 261–272 vol.1.

[26] L. Lamport, “Paxos made simple,” *ACM Sigact News*, vol. 32, no. 4, pp. 18–25, 2001.