Designing a Resilient Deployment and Reconfiguration Infrastructure for Remotely Managed Cyber-Physical Systems

Subhav Pradhan(✉), Abhishek Dubey, and Aniruddha Gokhale

Department of Electrical Engineering and Computer Science,
Vanderbilt University, Nashville, TN, USA
{subhav.m.pradhan,abhishek.dubey,a.gokhale}@vanderbilt.edu

Abstract. Multi-module Cyber-Physical Systems (CPS), such as satellite clusters, swarms of Unmanned Aerial Vehicles (UAV), and fleets of Unmanned Underwater Vehicles (UUV) provide a CPS cluster-as-a-service for CPS applications. The distributed and remote nature of these systems often necessitates the use of Deployment and Configuration (D&C) services to manage the lifecycle of these applications. Fluctuating resources, volatile cluster membership and changing environmental conditions necessitate resilience. Thus, the D&C infrastructure does not only have to undertake basic management actions, such as activation of new applications and deactivation of existing applications, but also has to autonomously reconfigure existing applications to mitigate failures including D&C infrastructure failures. This paper describes the design and architectural considerations to realize such a D&C infrastructure for component-based distributed systems. Experimental results demonstrating the autonomous resilience capabilities are presented.

Keywords: Self-reconfiguration · Autonomous resilience · Deployment and reconfiguration · Component-based distributed systems

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 [10]. 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 CPS. These systems often operate in unwieldy environments where resources are very limited, the dynamic nature of the system results in ever-changing cluster properties, such as membership, failures and fluctuation in resource availability is common, and human intervention to address these problems is rarely feasible.

I. Crnkovic and E. Troubitsyna (Eds.): SERENE 2016, LNCS 9823, pp. 88–104, 2016.
DOI: 10.1007/978-3-319-45892-2_7

Resilience is thus a key requirement for such cyber physical systems. A resilient system is defined as a system that is capable of maintaining and recovering its functionality when faced with failures and anomalies. Since human intervention is extremely limited resilience should be autonomous. As such, resilience can be provided either by using redundancy or by using a self-reconfiguration (self-adaptation) mechanism. In this paper, we are concerned with the self-reconfiguration aspect. The goal is to achieve a self-adaptive system [20] for which following requirements must be met:

- Requirement 1: an adaptation capability that can maintain and recover the system's functionality by adapting applications hosted on the system.
- Requirement 2: the adaptation capability itself should be resilient such that any failure or anomaly does not effect the adaptability of the overall system.

We are concerned with those CPS where the cyber functionalities are implemented using the Component-Based Software Engineering (CBSE) [8] approach, where applications are realized by composing, deploying and configuring software components with well-defined interaction ports. A number of different component models (providing the interaction and execution semantics for the components) exist: Fractal [3], CORBA Component Model (CCM) [14], LwCCM [13] etc. Similarly, there exists different Deployment and Configuration (D&C) infrastructures that are compatible with different component modes.

Since the D&C capability is a key artifact of any component-based system, we surmise that resilience can be improved by enhancing the D&C infrastructure so that it can provide the adaptation capability. This means the D&C infrastructure should not only be able to manage the lifecycle of applications, it should also be able to reconfigure existing applications and do so in a resilient manner [19]. However, existing D&C infrastructures do not support both these requirements. Either they are not capable of performing runtime reconfiguration [5,7,16,17] or others that are capable of performing runtime reconfiguration are themselves not resilient [1,2].

This paper overcomes limitations of existing solutions by presenting a novel and resilient D&C infrastructure that satisfies both the aforementioned requirements. In doing so, we make the following contributions:

- We present the key challenges in achieving a resilient D&C infrastructure.
- We present an architecture for a resilient D&C infrastructure that addresses these key challenges.
- We present experimental results to demonstrate application adaptability of our new D&C infrastructure.

The remainder of this paper is organized as follows: Sect. 2 presents existing work related to this paper and explains how our approach is different; Sect. 3 describes the target system model, D&C model, and fault model to present the problem at hand; Sect. 4 presents the key challenges that needs to be addressed in order to achieve a resilient D&C infrastructure; Sect. 5 presents detailed description of our solution and how it addresses aforementioned challenges; Sect. 6

presents experimental results; finally, Sect. 7 provides concluding remarks and alludes to future work.

2 Related Work

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 [7] and GoDIET [5] are general-purpose deployment frameworks targeted towards deploying large-scale, hierarchically composed, Fractal [3] component model-based applications in a grid environment. However, both of these deployment frameworks are not resilient and they lack support for application reconfiguration. As such, they do not satisfy the two requirements essential for realizing autonomous resilience.

The Object Management Group (OMG) has standardized the Deployment and Configuration (D&C) specification [15]. Our prior work on the Deployment And Configuration Engine (DAnCE) [17] describes a concrete realization of the OMG D&C specification for the Lightweight CORBA Component Model (LwCCM) [13]. LE-DAnCE [17] and F6 DeploymentManager [6] are some of our other previous works that extend the OMG's D&C specification. LE-DAnCE deploys and configures components based on the Lightweight CORBA Component Model [13] whereas the F6 Deployment Manager does the same for components based on F6-COM component model [16]. 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 resilience.

A significant amount of research exists in the field of dynamic reconfiguration of component-based applications. In [2], the authors present a tool called Planit for deployment and reconfiguration of component-based applications. Planit uses AI-based 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 failures. Another work presented in [1], supports dynamic reconfiguration of applications based on J2EE components. Although these solutions support application reconfiguration, none of them focus on resilience of their respective adaptation engine.

The authors in [4] present the DEECo (Distributed Emergent Ensembles of Components) component model, which is based on the concept of Ensemble-Based Component System (EBCS). In general, this approach replaces traditional explicit component architecture by the composition of components into *ensembles*. An ensemble is an implicit, inherently dynamic group of components where each component is an autonomic entity facilitating self-adaptive and resilient operation. In [9], authors present a formal foundation for ensemble modeling. However, they do not focus on the management infrastructure required to deploy and reconfigure these components.

3 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 the context of the D&C infrastructure.

3.1 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 [6] 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 relationship 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 distributed component-based applications[2] onto a target environment, the system needs to provide a software deployment service. 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 infrastructure composed of multiple deployment entities, with one entity residing on each node.

OMG's D&C specification [15] is a standard for deployment and configuration of component-based applications. Our prior work on the Locality-Enabled Deployment And Configuration Engine (LE-DAnCE) [17] is an open-source implementation of this specification. As shown in Fig. 1, LE-DAnCE implements a strict two-layered approach comprising different kinds of Deployment Managers (DM). A DM is a deployment entity. The Cluster Deployment Manager (CDM) is the single orchestrator that controls cluster-wide deployment process by coordinating deployment among different Node Deployment Managers (NDM).

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

[2] Although we use the component model described in [13], our work is not constrained by this choice and can be applied to other component models as well.

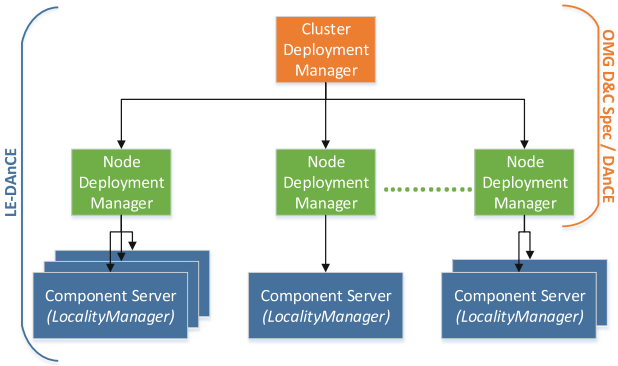

Fig. 1. Orchestrated deployment approach in LE-DAnCE [17]

Similarly, a NDM controls node-specific deployment process by instantiating component servers that create and manage application components.

LE-DAnCE, however, is not resilient and it does not support run-time application adaptation as well. Therefore, our work presented in this paper modifies and extends LE-DAnCE to achieve a D&C infrastructure capable of facilitating autonomous resilience.

3.3 Fault Model

Failure can be defined as a loss of functionality in a system. The goal of a resilient 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 involves 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 the failed subsystem.

In the systems under consideration, we observe that subsystem failures can be categorized as infrastructure or application failures. Infrastructure failures are failures that arise due to faults affecting a system's network, participating nodes, or processes that are running in these nodes. Usually, infrastructure failures can be classified as *primary failures*. Whereas, 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* caused due to infrastructure failures.

3.4 Problem Statement

For the prescribed system and fault model, the D&C infrastructure should, first and foremost, be capable of dealing with infrastructure failures. Conceptually, a

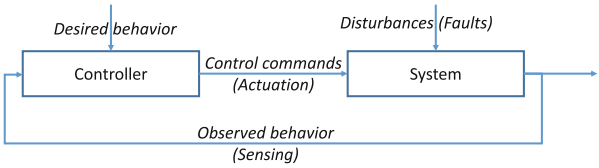

Fig. 2. Self-adaptive system as a control system

resilient infrastructure can be modeled as a resilient feedback control loop that observes the system state and compensates for disturbances in the system to achieve a desired behavior as shown in Fig. 2.

To find similarities with the traditional self-adaptive loop and the system under discussion, consider that a failure in the infrastructure can be considered as a disturbance. This failure can be detected by behavior such as "node is responding to pings" (indicating there is infrastructure failure) or not. Once the failure has been detected, the loss of functionality needs to be restored by facilitating reconfiguration, for example, re-allocating components to a functioning node, etc.; this needs to be done in a resilient manner. The presence of the controller and its actuation ability enables the self-adaptive property needed of an autonomously resilient system.

4 Key Considerations and Challenges

To correctly provide resilient D&C services to a CPS cluster, the D&C infrastructure must resolve the challenges described below:

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, the 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.

Challenge 2 (Leader election): As faults occur in CPS, 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.[3] 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 among themselves regarding the identity of the new cluster leader. This process needs to be facilitated by a leader election algorithm.

Challenge 3 (Deployment sequencing): 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.

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 those portions of the distributed application that were deployed on the failed node. Supporting resilience requires the D&C infrastructure to keep track of the global system state, which consists of (a) component-to-application mapping, (b) component-to-implementation mapping[4], (c) component-to-node mapping, (d) inter-component connection information, (e) component state information, and (f) the current group membership information. Such state preservation is particularly important for a new leader.

5 A Resilient D&C Infrastructure

Figure 3 presents an overview of our solution. Infrastructure failures are detected using the Group Membership Monitor (GMM). Application failure detection is outside the scope of this paper, however, we refer readers to our earlier work [11] in this area. The controller is in fact a collection of DMs working together to deploy and configure as well as reconfigure application components. The specific actuation commands are redeployment actions taken by the DMs.

5.1 Solution Architecture

Figure 4 presents the architecture of our resilient D&C infrastructure. Each node consists of a single Deployment Manager (DM). A collection of these DMs forms the overall D&C infrastructure. Our approach supports distributed, peer-to-peer application deployment, where each node controls its local deployment process. Each DM spawns one or more Component Servers (CSs), which are processes responsible for managing the lifecycle of application components. Note that our

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

[4] A component can have multiple implementations.

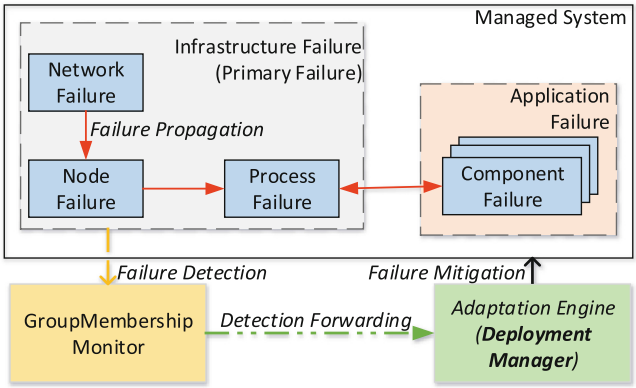

Fig. 3. Overview of a resilient D&C infrastructure.

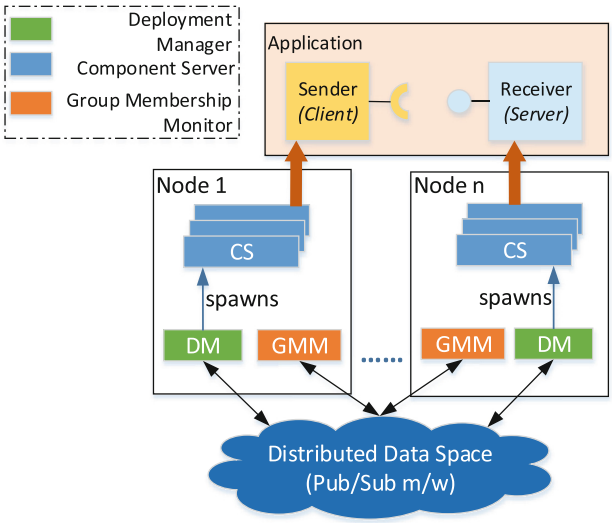

Fig. 4. Architecture of a resilience D&C infrastructure.

approach does not follow a centralized coordinator for deployment actions; rather the DMs are independent and use a publish/subscribe middleware to communicate with each other.

In our architecture, we use the GMM to maintain up-to-date group membership information, and to detect failures via a periodic heartbeat monitoring mechanism. The failure detection aspect of GMM relies on two important parameters – *heartbeat period* and *failure monitoring period*. These configurable parameters allows us to control how often each DM asserts its liveliness and how often each DM monitors failure. For a given failure monitoring period, a lower

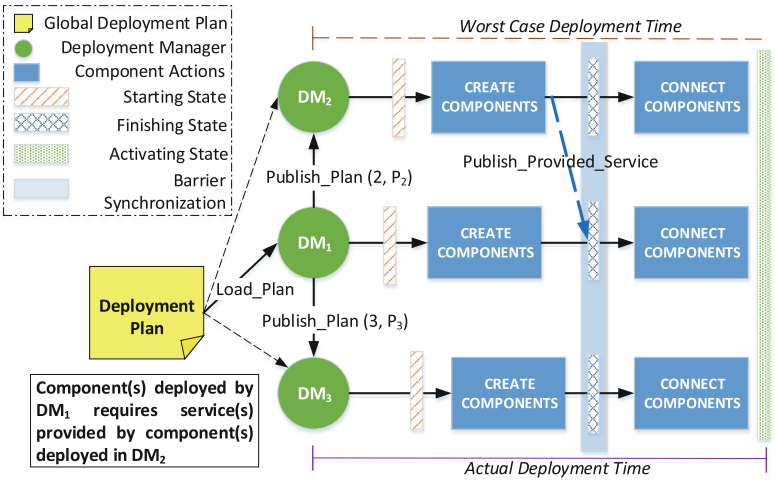

Fig. 5. A three-node deployment and configuration setup

heartbeat period results in higher network traffic but lower failure detection latency, whereas a 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 heartbeat period. Addressing intermittent failures is out of scope for this paper.

Figure 5 shows an event diagram demonstrating a three node deployment process of our new D&C infrastructure. 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. 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*. Two different global deployment plans can be deployed by two different deployment leaders since we do not require a centralized coordinator in our approach.

Deployment and configuration in our scheme is a multi-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 and ACTIVATED stage is where the application components in the deployment plan is active. In the rest of this section, we describe how information in this table is used in our solution to address the key challenges.

Table 1. 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 the previous stage is split into node-specific plans and they are published to the distributed data space using pub/sub middleware
	(2) Node-specific plans published above are received by all DMs and only the ones that are relevant are further split into component server (CS)-specific plans
STARTING	(1) CS-specific plans created in the 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 stage 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

5.2 Addressing Resilient D&C Challenges

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 as 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 [12]. To be more specific, we use the Real-Time Publish Subscribe (RTPS) peer-to-peer discovery mechanism specified by DDS.

Failure Detection Mechanism: To detect the loss of existing members, we need a failure detection mechanism that detects different kinds of failures. In our architecture this functionality is provided by the GMM. The GMM residing on each node uses a 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 kinds of infrastructure failures.

Resolving Challenge 2 (Leader Election): Leader election is required in order to tolerate cluster leader failure. We do this by implementing a rank-based leader election algorithm. Each DM is assigned a unique numeric rank value and this information is published by each DM as part of its heartbeat. Initially the DM with the least rank will be picked as the cluster leader. If the cluster leader fails, each of the other DMs in the cluster will check their group membership table and determine if it is the new leader. Since, we associate a unique rank with each DM, only one DM will be elected as the new leader.

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.

Resolving Challenge 4 (D&C State Preservation): In our current implementation, once a deployment plan is split into node-specific deployment plans, all of the DMs receive the node-specific deployment plans. Although any further action on a node-specific deployment plan is only taken by a DM if that plan belongs to the node in which the DM is deployed, all DMs store each and every node-specific deployment plans in its memory. This ensures that deployment-related information is replicated throughout a cluster thereby preventing single point of failure. However, this approach is vulnerable to DM process failures since deployment information is stored in memory. To resolve this issue, we are cur-

rently working on extending our solution to use a persistent backend distributed database to store deployment information.

6 Experimental Results

This section presents results to demonstrate the autonomous resilience capabilities of our D&C infrastructure. We show how our resilient D&C infrastructure adapts applications as well as itself after encountering a node failure during deployment-time, and runtime.

6.1 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 1 GB 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 Sects. 6.2 and 6.3 is a simple two-component client-server experiment presented earlier in Fig. 4. 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 s and failure monitoring period set to 5 s.

6.2 Node Failure During Deployment-Time

Figure 6 presents a time sequence graph of how our D&C infrastructure adapts itself to tolerate failures during deployment-time. As can be seen, 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.

6.3 Node Failure During Application Run-Time

Figure 7 presents a 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

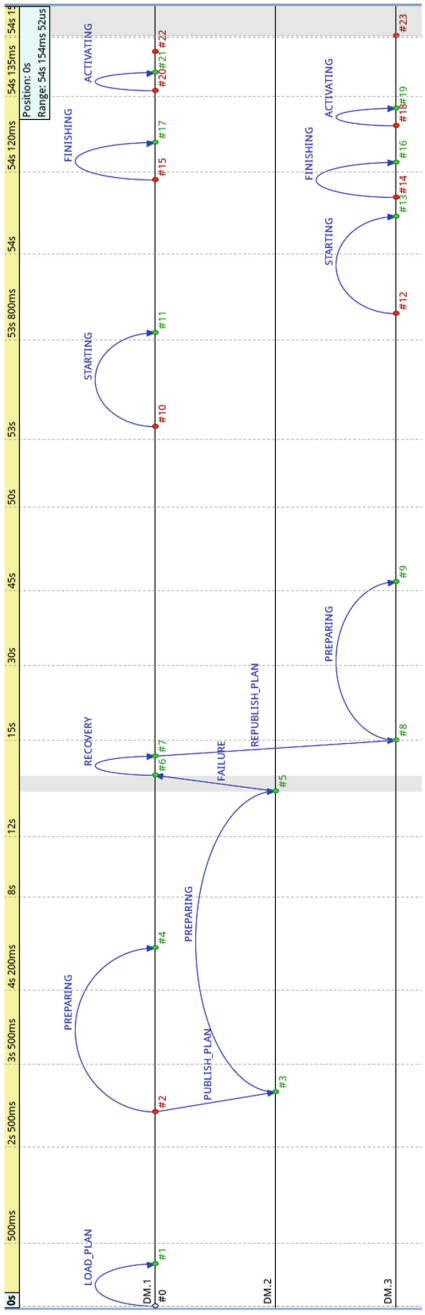

Fig. 6. Node failure during application deployment time.

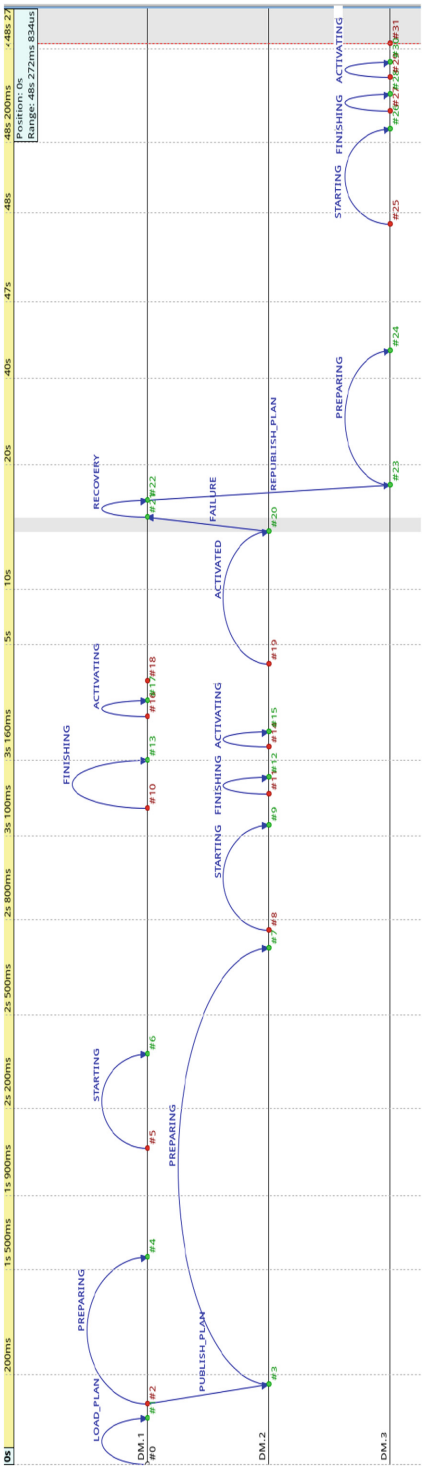

Fig. 7. Node failure during application run-time

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 similar to the way 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.

7 Conclusions and Future Work

This paper described a resilient Deployment and Configuration (D&C) infrastructure for highly dynamic and remote CPS. This dynamic and remote nature calls for autonomous resilience in such systems. The D&C infrastructure is the right artifact to architect such a solution as these systems are commonly built using Component-Based Software Engineering (CBSE) approach using appropriate component models and their corresponding D&C infrastructure. However, existing D&C infrastructures do not meet the requirements essential to facilitate autonomous resilience. As such, in this paper we presented a novel D&C infrastructure that is resilient and capable of reconfiguring existing applications.

The work presented in this paper incurs a few limitations: (1) As mentioned in Sect. 5.2, our current implementation for D&C state preservation is sufficient but not ideal. In our on-going research effort [18], we look into using a distributed database to store relevant D&C state resulting in a stateless D&C infrastructure. We plan to add similar concept to extend the work presented in this paper. (2) The D&C infrastructure presented in this paper performs reconfiguration without any smartness,*i.e.*, we randomly decide where a component should be migrated. However, this is not sufficient for systems that can host multiple applications. We require the D&C infrastructure to utilize available system information to make a more educated decision on how a system should be reconfigured. Again, some initial work towards achieving such an infrastructure has been presented as part of our on-going research effort [18].

Acknowledgment. This work was supported by the DARPA System F6 Program under contract NNA11AC08C, USAF/AFRL under Cooperative Agreement FA8750-13-2-0050, and Siemens Corporate Technology. 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 the sponsors.

References

1. Akkerman, A., Totok, A.A., Karamcheti, V.: Infrastructure for automatic dynamic deployment of J2EE applications in distributed environments. In: Dearle, A., Savani, R. (eds.) CD 2005. LNCS, vol. 3798, pp. 17–32. Springer, Heidelberg (2005)
2. Arshad, N., Heimbigner, D., Wolf, A.L.: Deployment and dynamic reconfiguration planning for distributed software systems. In: Proceedings of 15th IEEE International Conference on Tools with Artificial Intelligence, pp. 39–46. IEEE (2003)
3. Bruneton, E., Coupaye, T., Leclercq, M., Quéma, V., Stefani, J.B.: The fractal component model and its support in java. Softw. Pract. Exp. **36**(11–12), 1257–1284 (2006)
4. Bures, T., Gerostathopoulos, I., Hnetynka, P., Keznikl, J., Kit, M., Plasil, F.: Deeco: an ensemble-based component system. In: Proceedings of the 16th International ACM Sigsoft Symposium on Component-Based Software Engineering, pp. 81–90. ACM (2013)
5. Caron, E., Chouhan, P.K., Dail, H.: Godiet: a deployment tool for distributed middleware on grid'5000. Ph.D. thesis, INRIA (2006)
6. Dubey, A., Emfinger, W., Gokhale, A., Karsai, G., Otte, W., Parsons, J., Szabo, C., Coglio, A., Smith, E., Bose, P.: A software platform for fractionated spacecraft. In: Proceedings of the IEEE Aerospace Conference 2012, pp. 1–20. IEEE, Big Sky, MT, USA, March 2012
7. Flissi, A., Dubus, J., Dolet, N., Merle, P.: Deploying on the grid with deployware. In: 8th IEEE International Symposium on Cluster Computing and the Grid, CCGRID 2008, pp. 177–184. IEEE (2008)
8. Heineman, G.T., Councill, W.T. (eds.): Component-based Software Engineering: Putting the Pieces Together. Addison-Wesley Longman Publishing Co., Inc., Boston (2001)
9. Hennicker, Rolf, Klarl, Annabelle: Foundations for ensemble modeling – the Helena approach. In: Iida, Shusaku, Meseguer, José, Ogata, Kazuhiro (eds.) Specification, Algebra, and Software. LNCS, vol. 8373, pp. 359–381. Springer, Heidelberg (2014)
10. Lee, E.A.: Cyber physical systems: design challenges. In: 2008 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC), pp. 363–369. IEEE (2008)
11. Mahadevan, N., Dubey, A., Karsai, G.: Application of software health management techniques. In: Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, pp. 1–10. ACM (2011)
12. Object Management Group: Data Distribution Service for Real-time Systems Specification, 1.0 edn., March 2003
13. Object Management Group: Light Weight CORBA Component Model Revised Submission, OMG Document realtime/03-05-05 edn., May 2003
14. Object Management Group: The Common Object Request Broker: Architecture and Specification Version 3.1, Part 3: CORBA Component Model, OMG Document formal/2008-01-08 edn., January 2008
15. OMG: Deployment and Configuration Final Adopted Specification. http://www.omg.org/members/cgi-bin/doc?ptc/03-07-08.pdf
16. Otte, W.R., Dubey, A., Pradhan, S., Patil, P., Gokhale, A., Karsai, G., Willemsen, J.: 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 2013), Paderborn, Germany, June 2013

17. Otte, W., Gokhale, A., Schmidt, D.: 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, pp. 21–30. ACM (2011)
18. Pradhan, S., Dubey, A., Levendovszky, T., Kumar, P.S., Emfinger, W.A., Balasubramanian, D., Otte, W., Karsai, G.: Achieving resilience in distributed software systems via self-reconfiguration. J. Syst. Softw. (2016)
19. Pradhan, S., Gokhale, A., Otte, W., Karsai, G.: 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 2012). IEEE, San Juan, Puerto Rico, USA, December 2012
20. Salehie, M., Tahvildari, L.: Self-adaptive software: landscape and research challenges. ACM Trans. Auton. Adapt. Syst. (TAAS) **4**(2), 14 (2009)