Key Considerations for a Resilient and Autonomous Deployment and Configuration Infrastructure for Cyber-Physical Systems

Subhav Pradhan, William Otte, Abhishek Dubey, Aniruddha Gokhale and Gabor Karsai
Institute for Software Integrated Systems
Dept. of Electrical Engineering and Computer Science
Vanderbilt University, Nashville, TN 37212, USA
Email: {pradhasm,wotte,dabhishe,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. The distributed and remote nature of these systems often necessitates the use of Deployment and Configuration (D&C) services to manage the lifecycle of software applications. Fluctuating resources, volatile cluster membership and changing environmental conditions require resilient D&C services. However, the dynamic nature of the system often precludes human intervention during the D&C activities, which motivates the need 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 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 makes two contributions towards addressing these needed. First, we identify the key challenges in achieving such a self-adaptive D&C infrastructure. Second, we present our ideas on resolving these challenges and realizing a self-adaptive D&C infrastructure.**

***Keywords*-self-adaptation, resilience, deployment and configuration, cyber-physical.**

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 wherein 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, these systems must be resilient to failures and other anomalies.

A resilient system can adapt to both internal and external anomalies by modifying its normal behavior while still remaining functional. Since human intervention is extremely limited in case of the dynamic distributed systems we consider, 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. This paper describes the key considerations in engineering such autonomic resilient systems.

Specifically, we focus on describing how the deployment and configuration capabilities (D&C) of a CPS can be made self-adaptive to realize resilient CPS. A D&C infrastructure is responsible for managing an application's lifecycle within a system, which includes initial deployment and configuration of the application as well as run-time modifications including mitigation and recovery from various kinds of failures. However, since the D&C infrastructure can also incur faults, it itself should be resilient to failures through self adaptation. These requirements have been identified in our previous work [2], however, to our knowledge 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* [3], these solutions are not self-adaptive.

Following are the primary contributions of this paper:

- We identify the key challenges in achieving a self-adaptive D&C infrastructure for highly dynamic CPSs, and
- We briefly describe a self-adaptive D&C infrastructure that addresses the above-mentioned challenges.

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 a brief architectural description of a self-adaptive D&C infrastructure capable of handling aforementioned challenges; and finally, Section V 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 [4]. As highlighted in this roadmap, we implement all the steps in the collect/analyze/decide/act loop.

In this section we compare our work 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** [5] and **GoDIET** [6] are general-purpose deployment frameworks targeted towards deploying large-scale, hierarchically composed, Fractal [7] component model-based applications in a grid environment. However, both of these deployment frameworks lack autonomous resilience since neither of them supports application adaptation nor self-adaptation.

The Object Management Group (OMG) has standardized the Deployment and Configuration (D&C) specification [8]. Our prior work on the Deployment And Configuration Engine (**DAnCE**) [9], [10] describes a concrete realization of the OMG D&C specification for the Lightweight CORBA Component Model (LwCCM) [11]. **LE-DAnCE** [10] and **F6 DeploymentManager** [12] 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 [11] whereas the F6 Deployment Manager does the same for components based on F6-COM component model [13]. 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 D&C infrastructure adaptation.

B. Dynamic Re-configuration

A significant amount of research has been conducted in the field of dynamic reconfiguration for component-based applications. In [14], 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 [15] 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 [16]–[19] all support application adaptation but not resilience of the adaptation engine itself. In [17], the authors present a self-managing solution which is component-based and non-intrusive since it decouples application specifics from autonomic specific software artifacts. It supports self-optimization and self-healing. Another work presented in [18], supports dynamic reconfiguration of applications based on J2EE components. In [19], the authors present a framework that supports multiple extensible reconfiguration algorithms for run-time adaptation of component-based applications.

Finally, in [20], 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 and Challenges

This section describes the problem at hand by first presenting the target system model and then the problem of self-adaptation in context of the D&C infrastructure. Finally, we pose the key considerations in engineering resilient self-adaptive D&C for CPS.

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

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 system composed of multiple deployment entities, called *Deployment Managers (DM)*, with one DM residing on each node.

B. Problem Statement

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.

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 1.

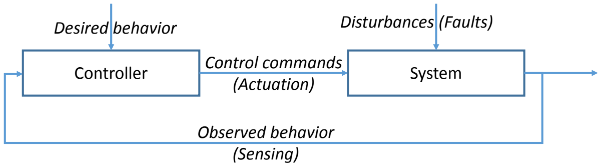

Figure 1. Self-adaptive System as a Control System

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.

In the system discussed in the paper, a disturbance could be an infrastructure fault or an application fault. Observed behavior could be 'node is responding to pings' (indicating there is no infrastructure failure), or 'application is functioning' (indicating there is no application failure). The controller logic can simply detect if the opposite of the above behaviors is observed (i.e. 'node is not responding to pings' or 'application stopped responding').

The actuation commands include the necessary actions to facilitate 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.

C. Key Considerations and Challenges

To correctly provide self-adaptive D&C services to a CPS cluster, the D&C infrastructure must resolve a number of challenges; these challenges are described below:

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, 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.

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.[4] However, a node that hosts the DM acting as the cluster leader can fail at

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

[2] Although we use the component model described in [11], 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

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

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.

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.

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 those portions of the distributed application that were 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,[5] (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 particularly important for a newly elected cluster leader.

IV. Self-adaptive D&C Infrastructure

Figure 2 shows the overall approach of our solution. Infrastructure failures are **detected** using the *Group Membership Monitor* (GMM) described in Section IV-A. Application failure detection is outside the scope of this paper, however, we refer readers to our earlier work [21] 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 describing how our approach is addressing the key challenges highlighted in Section III-C.

A. An Architecture for Self-adaptive D&C

Figure 3 presents the architecture of our self-adaptive 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.

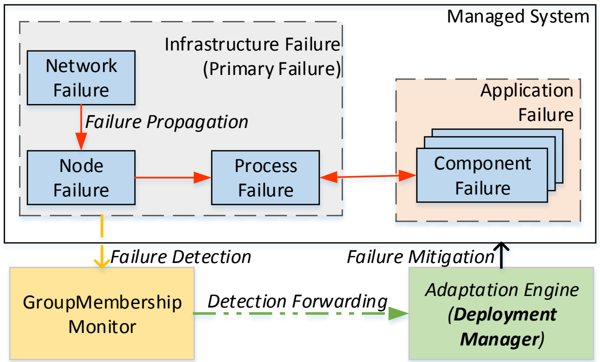

Figure 2. Anomaly Detection and Handling for Self-adaptive D&C.

Each DM, if required, spawns one or more Component Servers (CSs). These CSs are processes that are responsible for managing lifecycle of application components. Note that our 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.

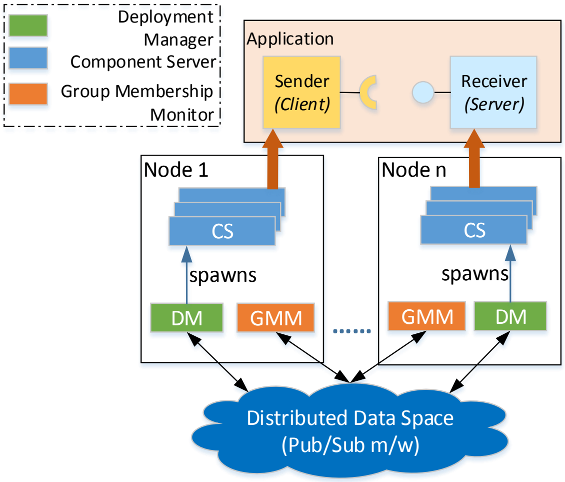

Figure 3. Self-adaptive D&C architecture

In our architecture, we use GMM for two reasons: (1) maintaining up-to-date group membership information, and (2) detecting 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 parameters are configurable. Configuring the heartbeat period allows us to control how often each DM assert their liveliness, whereas configuring the failure monitoring period allows us to control how often

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

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 heartbeat period.

Handling intermittent failures is out of scope for this paper, however, the above-mentioned tunable parameters can be configured to provide a rudimentary support for handling intermittent failures. For example, a setting of low heartbeat period and comparatively higher failure monitoring period results in a larger monitoring window and therefore can arguably reduce occurrence of intermittent failures. Setting these values will depend on the system and the environment in which it is deployed.

Figure 4 shows an event diagram demonstrating a three node 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 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* [6] for that particular deployment plan. 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 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. In the rest of this section, we describe how information in this table is used in our solution to address the key challenges.

B. Addressing Self-adaptive D&C Challenges

We now discuss how our architecture resolves the key challenges identified in Section III-C.

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.

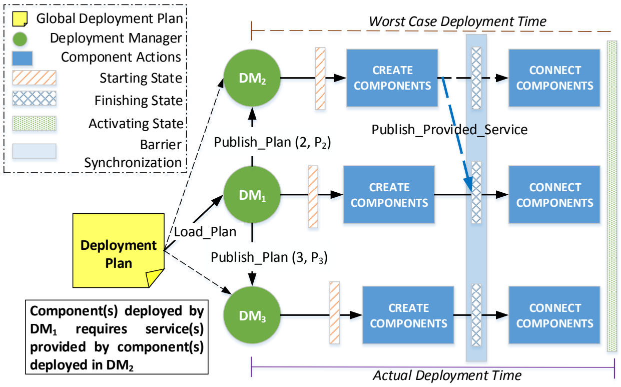

Figure 4. A Three-node Deployment and Configuration Setup

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 [22]. To be more specific, we use the Real-Time Publish Subscribe (RTPS) peer-to-peer discovery mechanism supported by OpenDDS.

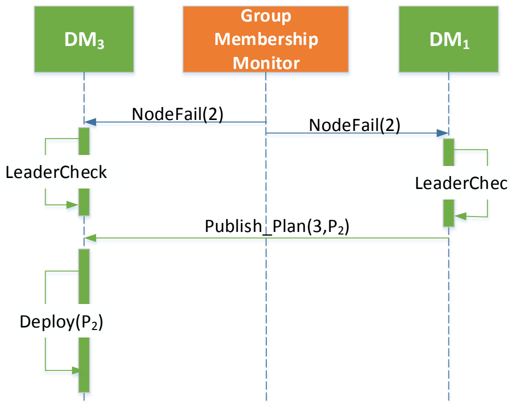

Figure 5. Failure Detection Mechanisms

Failure Detection Mechanism (Figure 5): 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

[6] 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*.

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.

different classes of infrastructure failures.

2) 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 unique rank with each DM, only one DM will be elected as the new leader.

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 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 participate in the 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.

Our future work is seeking 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 components' state information. This will also be part of our future work.

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 lead 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.

V. Conclusions and Future Work

This paper described a self-adaptive Deployment and Configuration (D&C) infrastructure for highly dynamic CPS. The 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 approach to self-adaptive D&C.

The work presented in this paper incurs a few limitations: (1) the failure detection mechanism presented in Section IV-B1 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* [23] as a mechanism to achieve distributed consensus before taking any failure mitigation actions; (2) as mentioned in Section IV-B4, 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; and (3) empirically validating the architecture on a number of large-scale CPS will also be a focus of our future work.

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

Acknowledgment

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.

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] 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.

[3] 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.

[4] 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.

[5] 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.

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

[7] 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.

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

[9] 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.

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

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

[19] 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.

[20] 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.

[21] 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.

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

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