Towards a Resilient Deployment and Configuration Infrastructure for Fractionated Spacecraft

Subhav Pradhan, William R. Otte, Abhishek Dubey, Aniruddha Gokhale, and Gabor Karsai*
* Institute for Software-Integrated Systems, Vanderbilt University,
Nashville, TN 37235, USA
Email:{pradhasm,wotte,dabhishe,gokhale,gabor}@isis.vanderbilt.edu

***Abstract*—Fractionated spacecraft are clusters of small, independent modules that interact wirelessly to realize the functionality of a traditional monolithic spacecraft. System F6 (F6 stands for Future, Fast, Flexible, Fractionated, Free-Flying spacecraft) is a DARPA program for fractionated spacecraft. Software applications in F6 are implemented in the context of the F6 Information Architecture Platform (IAP), which provides component-based abstractions for composing distributed applications. The lifecycle of these distributed applications must be managed autonomously by a deployment and configuration (D&C) infrastructure, which can redeploy and reconfigure the running applications in response to faults and other anomalies that may occur during system operation. Addressing these D&C requirements is hard due to the significant fluctuation in resource availabilities, constraints on resources, and safety and security concerns. This paper presents the key architectural ideas that are required in realizing such a D&C infrastructure.**

***Index Terms*—D&C for CPS, reconfiguration, failures.**

I. Introduction

System F6 [1] (F6 stands for Future, Fast, Flexible, Fractionated, Free-Flying spacecraft) is a highly resource-constrained, dynamic system which consists of a cluster of satellites forming the 'fractionated spacecraft', which provides a distributed computing platform in space. Each satellite module communicates wirelessly with others in the same cluster and with resources located on the ground. F6 clusters may host multiple applications, which share available resources to accomplish their mission objectives through collaboration. It is a distributed cyber-physical system (CPS) because the cyber infrastructure that is responsible to support mission critical applications must be aware of all physical constraints and physical dynamics when operating a computing cluster platform in space.

The F6 Information Architecture Platform (IAP) provides infrastructure to build and manage applications that are created using software components with well-defined interfaces. These components conform to a novel component model called F6COM [2], which allows developers to design various software components that can operate in the real-time embedded environment of a fractionated spacecraft.

Managing the lifecycle of an application on the F6 Platform is the responsibility of the deployment and configuration (D&C) infrastructure. The F6 D&C infrastructure is based on the OMG Deployment and Configuration for Component-based Application specification [3], [4]. A typical F6 application will consist of several interconnected components. Deployment and configuration of these applications in F6 is specified in an XML document called a deployment plan that captures component configuration and interconnection. This plan is generated from a modeling tool. The D&C infrastructure consists of a set of privileged system processes called Deployment Managers (DM); one per F6 satellite node.

One of the key features required for creating a resilient platform is the ability to provide redeployment and reconfiguration for F6 applications to bring the system back to stability in the event of one or more failures in any part of the system. Moreover, since F6 is a highly resource-constrained and dynamic system, all these additional requirements are expected to provide efficient and predictable performance [5].

Therefore, the primary contribution of this paper is to present an ongoing research activity which extends our current D&C infrastructure for F6 system to support reconfiguration of deployed software elements in response to various faults that can occur in a highly resource-constrained dynamic system, such as F6. The rest of the paper is organized as follows: Section II describes related research; Section III describes the current D&C infrastructure (F6DM); Section IV describes our current work to extend the F6 DM to provide component reconfiguration capabilities; and finally Section V provides concluding remarks alluding to future work.

II. Related Work

Due to the growing complexity of embedded systems, adaptive D&C infrastructures are becoming increasingly important. In [6], [7], the authors present a novel middleware that supports reconfiguration of Distributed Real-Time and Embedded (DRE) systems, and a way for using Model-Driven Engineering (MDE) to create multiple possible configurations at design time such that the system can be reconfigured from one configuration to another based on events depicted using state diagrams. The actual transition from one configuration to another is performed via model to model (M2M) transformations. While this work shares our vision, a major limitation of this work is that it supports only a finite number of predefined reconfigurations. For this approach to be effective, a user needs to pre-define all possible failure scenarios and their reconfigurations during design-time, which is hard and tedious.

Other MDE-based approaches devoted to embedded system designs such as Ocarina [8], ModES [9], and [10] allow development of real-time embedded systems but these approaches

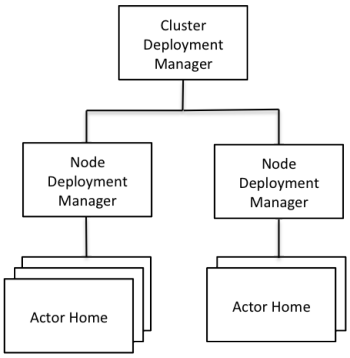

Fig. 1. Simplified F6 D&C Architecture

do not support reconfiguration. Ocarina is a framework that allows users to create AADL (Architecture Analysis and Design Language) models based on which it can perform validation and generate application code. Similarly, ModES defines a set of meta-models allowing users to create models of application, platform to indicate available hardware/software resources and mappings between the two. Finally, [10] allows users to specify generic transformation rules to generate platform-specific models from application model and target platform model. Once a platform-specific model is generated, the framework generates executable code. Thus, based on transformation rules, a user can generate different platform-specific models for a given application.

III. F6DM Architecture and Design

The F6IAP consists of a specific operating system, the F6 Operating System (F6OS), which provides all necessary abstractions to manage the resources available to all applications. A *thread* is the smallest unit of execution in F6OS and an *actor* (equivalent to a process in typical operating systems) is a collection of threads that share a common address space. F6OS supports two kinds of actors - (1) Application actors, and (2) Platform actors. Application actors are user-defined actors that represent applications. These application actors consists of various user-defined components which can be dynamically installed and removed. Unlike application actors, platform actors extend F6OS by providing long running and privileged services that are essential for management of the F6 cluster and regular application actors. The F6 Deployment Manager (F6DM) is one of the platform actors that resides on every node.

The F6DM is an important part of the F6 D&C infrastructure and is implemented as a platform actor that is responsible for deployment and configuration of applications throughout the F6 system. The F6DM achieves its functionality by performing the following three roles as shown in Figure 1:

- Cluster Deployment Manager: At this level the F6DM operates to coordinate deployment activities amongst node-level F6DM instances.
- Node Deployment Manager: At this level the F6DM operates as a creator and manager of one or more actors on a single node.
- Actor Home: This is the lowest level at which the F6DM operates as a component server responsible for managing lifecycles of various components. It is responsible for - (a) creating and configuring components, and (b) managing the lifecycle of the components based on the commands received from the Node Deployment Manager.

The deployment and configuration in System F6 is a four-phase process that is initiated by the Operations Manager (another platform actor that manages spacecraft system operations), typically coordinating with the Cluster Deployment Manager, but may be initiated by communicating with any reachable Deployment Manager instance. Following are the four phases of application deployment:

- Prepare Plan: This phase is the initial part of the deployment process during which all nodes involved in a deployment are provided with actor binaries and their deployment plan.
- Start Launch: This is the second phase of application deployment. In this phase all components that are part of the deployment plan provided in the first phase will be deployed into their respected physical nodes, but are not yet connected or running.
- Finish Launch: In this phase of application deployment, all connections between components instantiated in the second phase are created and made ready for activation.
- Start: During this phase, all components and actors are placed into active status and the application begins execution.

IV. Reconfiguration Capabilities in the D&C Infrastructure

Reconfiguration is of utmost importance in systems such as F6 since it is required to maintain system functionality which might be affected by various failures. To add reconfiguration capability to our existing D&C framework we are taking inspiration from our previous work [11] in the field of Software Health Management. *software health management* (SHM) is an extension of classical software fault tolerance that borrows ideas from system health management [12]. SHM works by detecting, isolating and adaptively mitigating the effects of faults, and is an active area of research [13], [14], [15].

There are two different approaches to specify the mitigation scenarios: explicit encoding of mitigation scenarios [15], or implicit encoding of mitigation scenarios.[1] Explicitly defining all possible sequences of fault events and the corresponding reconfiguration actions is both cumbersome and error-prone, especially for large systems. Implicit encoding [16], [11], is centered around a *functional model*, created at design-time, that describes the *required functionality of the system in terms of the component configurations* required to provide that functionality. The functional requirements of a system are expressed by describing a collection of trees, where the root of each tree is a system goal and the children are the basic

[1]Detection and isolation of faults is out of scope of Deployment Manager and hence this paper.

functions that are required to achieve that goal. A functional requirement in this tree can be completely dependent on another functional requirement or can be separately mapped to a set of components that are required to achieve that function. This mapping is called functional allocation. This approach encodes the possible design space of valid configurations succinctly, which can then be searched at runtime — subject to state and other specified constraints — to determine a new valid configuration.

A function at any level of the functional requirements' directed acyclic graph can depend on other child functions and can depend upon the availability of a set of components at that level. The set of components related to a function can be hierarchically organized into groups. These are:

1) ALT Group: **Exactly 1 out of N** components from a given group are required to achieve the X function. We write this as $X \rightarrow \texttt{EXACTLY}(1, Comp1, Comp2, ...CompN)$, where comp$i$ is the i^{th} component of the group.
2) M-of-N group: **At least M out of N** components from a given group are required to achieve the X function. We write it as $X \rightarrow \texttt{ATLEAST}(M)(Comp1, Comp2, ...CompN)$
3) AND Group: **All** components from a given group are required to achieve the X function. We write it as $X \rightarrow \texttt{ALL}(Comp1, Comp2, ...CompN)$.

In [16], we showed that the set of expressions based on the functional requirements tree, the function allocations and the explicit and implicit component operational requirements describe a Boolean search space, where the state of each component and group is described by a Boolean literal. Additionally, all function allocation requirements can be encoded as boolean expressions. For example $\texttt{EXACTLY}(1, Comp1, Comp2)$ can be encoded as $(Comp1 \wedge \neg Comp2) \vee (Comp2 \wedge \neg Comp1)$.

Thereafter, we could possibly use a Boolean satisfiability (SAT) solver as follows: (a) Translating the functional requirement, function allocation and component operational requirements into a set of clauses in Conjunctive Normal Form (CNF), (b) Setting constraints on the states of components that are known to be faulty (determination of a fault component is the task of a fault diagnoser); (c) Encoding the currents state of active components as assumptions, (d) and finally, invoking the SAT solver to search for a satisfying solution.

Even though the constraint problem can be very large depending on the states of all the components in the system, there is a good reason to believe that this approach of creating boolean satisfiability problem and solving it using SAT solver is efficient since modern off-the-shelf SAT solvers are capable of solving problems containing thousands of variables and tens of thousands of clauses.

Currently, we consider the following kinds of failures in the system (collection of nodes):

- Component failure: Actors are composed of components. In this system, we consider components to be units of fault containment, which can be replaced as a whole.

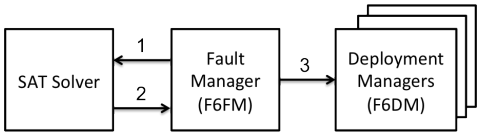

Fig. 2. Reconfiguration Process Involving F6FM, F6DM and a SAT Solver

- Actor failure: Represents a failure of an actor which can also be an Actor Home. By default, failure of an actor means that all components that reside in that actor also fail.
- Node failure: Represents failure of the entire physical computing host. This results in failure of all actors and therefore all components hosted on the faulty node.

Detection of all three failures in the F6 software layer is the responsibility of the Fault Manager (F6FM) which is another platform actor that runs on every node to provide fault detection and diagnosis services. Since fault detection and diagnosis is not part of this paper, we will not be discussing F6FM in more detail. Figure 2 shows the targeted architecture to achieve resilient D&C infrastructure in F6.

Once the F6FM detects a fault, it sends this information to the SAT solver with the current system state. The SAT solver, in turn, will provide F6FM with a solution which is used to deduce the changes in state of different reconfigurable entities (components, actors) in the system. These configuration changes deduced from the solution provided by the SAT solver will be translated to appropriate reconfiguration commands (listed below) and sent to related F6DMs (which can be a Cluster Deployment Manager in case of physical node failures, Node Deployment Manager in case of Actor Home failures or Actor Home in case of component and application actor failure). Upon receiving reconfiguration commands from the F6FM, the F6DM will invoke those commands on different entities that needs to be reconfigured. Following is the list of fault mitigation commands/actions that can be issues for a faulty component:

- RESET: If there is a component failure, the F6DM can issue this command to the associated Actor Home which will re-initialize the affected component.
- FAILOVER: This command instructs the Actor Home to failover to one of the replicas of affected component. The replica component can be in a different node in which case the Actor Home needs to communicate with the corresponding Actor Home via F6DM. During replication we should also be cautious of a component's state. It is not in the scope of this paper to discuss replication in detail but our previous work [17] describes existing work on active and passive component replication. These techniques will be used in the F6 system.
- MOVE[2]: This command instructs the Actor Home to migrate a particular component to another location. Components could be relocated to a different partition in the same node or it can be relocated to a completely different

[2]the mapping of move command to the SAT problem is yet to be defined

physical node. During migration we have to be aware of component connections. A component in F6 system can have two kinds of connections:(1) Pub/Sub connection, and (2) facet/receptacle connection. In case of Pub/Sub connection, we do not need to worry about maintaining existing connection as the participating components are not physically connected; communication happens via the underlying middleware. However, components that are connected via facet-receptacle connection we need to make sure that this connection still exists once the components have been relocated.

- STOP: This command instructs the Actor Home to stop a component changing its state from active to passive.
- START: This command instructs the Actor Home to start a component changing its state from passive to active.
- REWIRE: This command instructs the Actor Home to rewire the existing facet-receptacle connection that a component has.

In our case, the component model we use (F6COM), supports four different component states - initial state, active state, passive state, and inactive state [2]. However, as part of reconfiguration, the only component states that we are concerned with are (a) active state, and (b) inactive state. The goal of the satisfiability problem is to find a set of components that can be turned to ACTIVE (set to 1) or turned to INACTIVE (set to 0), given the current set of assumptions for active and faulty components. If no solution is found, the assumptions on the known state of components is removed. If this still does help then we issue the RESET command. However, there might be cases where the component cannot be reset. In these cases, the appropriate solution would be to issue the FAILOVER command which makes sure that a replica of the affected component is activated. In this way, the provided solution is translated to a set of reconfiguration commands.

In case of physical node failure, the Cluster Deployment Manager will need to invoke separate action to determine elements affected by the node failure, based on existing deployment plan information, and make sure each element fails over to one of the its existing replicas.

V. Conclusions and Future Work

This paper discussed our approach towards achieving a resilient Deployment and Configuration (D&C) infrastructure for a highly resource-constrained and dynamic system. The solution proposed in this paper is to extend our existing D&C infrastructure and provide it with reconfiguration capabilities. This allows for redeployment and reconfiguration of software elements in response to faults and other anomalies that may occur during system operation. In future we would like to further extend our D&C infrastructure to itself become fault tolerant as in the current system it can be a single point of failure.

Acknowledgments: This work was supported by the DARPA System F6 Program under contract NNA11AC08C. 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. The authors thank Graham O'Neil and Olin Sibert of Oxford Systems and all the team members of our project for their invaluable input and contributions to this effort.

References

[1] A. Dubey, W. Emfinger, A. Gokhale, G. Karsai, and W. O. et al., "A Software Platform for Fractionated Spacecraft," in *Proceedings of the IEEE Aerospace Conference, 2012*. Big Sky, MT, USA: IEEE, Mar. 2012, pp. 1–20.

[2] 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 *To Appear in the Proceedings of the IEEE International Symposium on Object-oriented Real-time Distributed Computing (ISORC '13)*, Paderborn, Germany, Jun. 2013.

[3] *Deployment and Configuration of Component-based Distributed Applications, v4.0*, Document formal/2006-04-02 ed., OMG, Apr. 2006.

[4] 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.

[5] 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.

[6] F. Krichen, B. Zalila, M. Jmaiel, and B. Hamid, "A middleware for reconfigurable distributed real-time embedded systems," *Software Engineering Research, Management and Applications 2012*, pp. 81–96, 2012.

[7] F. Krichen, A. Ghorbel, B. Hamid, and B. Zalila, "An mde-based approach for reconfigurable dre systems," in *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2012 IEEE 21st International Workshop on*. IEEE, 2012, pp. 78–83.

[8] J. Hugues, B. Zalila, L. Pautet, and F. Kordon, "From the prototype to the final embedded system using the ocarina aadl tool suite," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, no. 4, p. 42, 2008.

[9] F. do Nascimento, M. Oliveira, and F. Wagner, "Modes: Embedded systems design methodology and tools based on mde," in *Model-Based Methodologies for Pervasive and Embedded Software, 2007. MOMPES'07. Fourth International Workshop on*. IEEE, 2007, pp. 67–76.

[10] W. Chehade, A. Radermacher, F. Terrier, B. Selic, and S. Gérard, "A model-driven framework for the development of portable real-time embedded systems," in *Engineering of Complex Computer Systems (ICECCS), 2011 16th IEEE International Conference on*. IEEE, 2011, pp. 45–54.

[11] N. Mahadevan, A. Dubey, D. Balasubramanian, and G. Karsai, "Model-based deliberative failure mitigation in software component assemblies," in *SEAMS*, 2013, in review.

[12] A. Srivastava and J. Schumann, "The Case for Software Health Management," in *Fourth IEEE International Conference on Space Mission Challenges for Information Technology, 2011. SMC-IT 2011.*, August 2011, pp. 3–9.

[13] L. Pike, A. Goodloe, R. Morisset, and S. Niller, "Copilot: A hard real-time runtime monitor," in *Runtime Verification*. Springer, 2010, pp. 345–359.

[14] J. Schumann, A. Srivastava, and O. Mengshoel, "Who guards the guardians??toward v&v of health management software," in *Runtime Verification*. Springer, 2010, pp. 399–404.

[15] N. Mahadevan, A. Dubey, and G. Karsai, "Application of software health management techniques," in *SEAMS*, 2011, pp. 1–10.

[16] A. Dubey, N. Mahadevan, and G. Karsai, "A deliberative reasoner for model-based software health management," in *The Eighth International Conference on Autonomic and Autonomous Systems*, 2012, pp. 86–92.

[17] F. Wolf, J. Balasubramanian, S. Tambe, A. Gokhale, and D. Schmidt, "Supporting component-based failover units in middleware for distributed real-time and embedded systems," *Journal of Systems Architecture*, vol. 57, no. 6, pp. 597–613, 2011.