Science of Computer Programming 106 (2015) 3–29

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

DREMS ML: A wide spectrum architecture design language for distributed computing platforms[☆]

Daniel Balasubramanian, Abhishek Dubey, William Otte, Tihamer Levendovszky, Aniruddha Gokhale, Pranav Kumar, William Emfinger, Gabor Karsai[*]

Institute for Software-Integrated Systems, Dept. of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN 37212, USA

A R T I C L E   I N F O

Article history:
Received 9 April 2013
Received in revised form 2 April 2015
Accepted 3 April 2015
Available online 11 April 2015

Keywords:
Architecture description language
Model-driven development
Fractionated spacecraft

A B S T R A C T

Complex sensing, processing and control applications running on distributed platforms are difficult to design, develop, analyze, integrate, deploy and operate, especially if resource constraints, fault tolerance and security issues are to be addressed. While technology exists today for engineering distributed, real-time component-based applications, many problems remain unsolved by existing tools. Model-driven development techniques are powerful, but there are very few existing and complete tool chains that offer an end-to-end solution to developers, from design to deployment. There is a need for an integrated model-driven development environment that addresses all phases of application lifecycle including design, development, verification, analysis, integration, deployment, operation and maintenance, with supporting automation in every phase. Arguably, a centerpiece of such a model-driven environment is the modeling language. To that end, this paper presents a wide-spectrum architecture design language called DREMS ML that itself is an integrated collection of individual domain-specific sub-languages. We claim that the language promotes "correct-by-construction" software development and integration by supporting each individual phase of the application lifecycle. Using a case study, we demonstrate how the design of DREMS ML impacts the development of embedded systems.

1. Introduction

Today's large-scale distributed real-time embedded systems are enormously complex and there is an ever-increasing need for engineering tools to support their design, development, integration, deployment, operation, and maintenance. Often these systems are running on a mobile distributed platform with wireless connectivity. Such platforms are expected to host many different applications running side-by-side, possibly in different security domains. The platform is highly

☆ 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 reflect the views of DARPA. The authors thank Olin Sibert and Graham O'Neil, and all the team members of our project for their invaluable input and contributions to this effort.

* Corresponding author.
E-mail addresses: daniel@isis.vanderbilt.edu (D. Balasubramanian), dabhishe@isis.vanderbilt.edu (A. Dubey), wotte@isis.vanderbilt.edu (W. Otte), tihamer@isis.vanderbilt.edu (T. Levendovszky), gokhale@isis.vanderbilt.edu (A. Gokhale), pkumar@isis.vanderbilt.edu (P. Kumar), emfinger@isis.vanderbilt.edu (W. Emfinger), gabor@isis.vanderbilt.edu (G. Karsai).

http://dx.doi.org/10.1016/j.scico.2015.04.002
0167-6423/© 2015 Elsevier B.V. All rights reserved.

resource-constrained thus resource management is an issue. Applications are often mission-critical and the system is expected to provide some guaranteed level of service in all situations, hence fault tolerance is a requirement. As an example one can consider a swarm of UAVs that act as a sensor network with in-network processing (while also flying in formation) or a fractionated spacecraft.

The DARPA System F6 program[1] was concerned with developing a cluster of small, independent spacecraft modules that interact wirelessly to maintain coordinated flight and support the functions usually performed by a monolithic satellite. This hardware platform is considered a reusable resource, a 'global space commons', on which various distributed software applications can be deployed and executed such that they share cluster resources. For a variety of reasons, many challenges arise in hosting these applications so that they can deliver services with the expected level of quality. For instance, it is expected that multiple organizations and users whose diverse software applications have varying demands for computational and communication resources are to be supported on this distributed hardware which can experience highly fluctuating connectivity. Moreover, the success of every mission depends on the capability of autonomous fault management, the delivery of desired real-time services and the availability of separate domains and levels of security.

To support these needs, we have developed an architecture called Distributed Real-Time Managed System (DREMS) [1], and its core, the Information Architecture Platform (*IAP*) for F6 that comprises (i) a novel operating system, (ii) a middleware layer that supports different communication-interaction patterns including request-response and publish/subscribe, and (iii) a component model that is used to develop DREMS applications [2,3]. Note that an entire F6 cluster of satellites is considered an open (hardware) platform whose services are available through the software platform (IAP) to applications developed by various customers. In this sense it is similar to a cloud-computing platform that follows a 'Platform as a Service' model.

The architecture provides first class support for multiple levels of security (MLS) that is enforced by the operating system kernel, the core ingredient of the Trusted Computing Base (TCB). Multi-level fault management is also supported by the IAP, with different functions, such as detection, mitigation, recovery distributed across the architecture.

Despite the elaborate and elegant runtime architecture to support DREMS applications, developers face a number of complex inherent and accidental challenges in constructing their applications. The inherent challenges pervade all phases of the application lifecycle, including design, development, deployment, resource scheduling, security provisioning, verification, determining the right testing strategies, runtime resource and fault management, and dealing with evolution in requirements and maintenance. The accidental challenges stem from the mundane and error prone activities of composing application components, providing the glue code to interact with the middleware capabilities, deploying the applications on the resources of the cluster, configuring the resources according to the partition schedule, provisioning monitoring and fault detection capabilities, testing, and dealing with all these complexities when iterating over the development cycle due to changes in requirements [4].

Clearly, there is a need for tools that application developers can use to handle all these challenges. Although a variety of tools that handle individual aspects of the problem space exist, such a disparate and disconnected tooling capability is not desired for a number of reasons. First, every different tool implies a learning curve on the part of developers and having to deal with the vagaries of individual tools. Second, the developers are now responsible for connecting these disparate tools into a tool chain requiring a number of transformations from the output of one tool to another. While these challenges are predominantly accidental, a more serious challenge stems from the fact that most of the existing tools do not provide domain-specific architectural reasoning capabilities desired for the IAP.

Architectural description languages, such as the Architecture Analysis and Design Language (AADL) [5,6], OMG SysML [7,8], and OMG MARTE profile for UML [9], are geared towards addressing the problem of disconnected and disparate tooling. Architecture description languages enable the proper decomposition of the system into manageable parts with well-defined interfaces (and thus contracts) between them, which ease system integration problems. Overall, an architectural model defined in these languages helps to capture in a single place details about the system's requirements, architecture and implementation details. A significant advantage for developers is that they can generate a variety of artifacts: analytical models to conduct timing, reliability, security, performance, etc. analysis from a single source. When language capabilities are offered in the context of a model-driven engineering process, particularly in the form of model-integrated computing (MIC) [10], application developers can validate their system using domain-specific artifacts that promote the realization of systems that are "correct-by-construction", and utilize the model transformation capabilities of MIC that can automate most of the mundane, accidental complexities faced by developers.

This paper presents DREMS modeling language (ML), which is an architecture description language (ADL) and its associated tooling. DREMS ML is a "wide spectrum" ADL because it provides a single source for DREMS application developers and the system integrator to address all the inherent and accidental challenges described above in realizing DREMS applications. Our recent publications describing DREMS ML [11] have focused on showing how it helps developers use the underlying component abstractions, configure the components and deploy applications that are composed of interacting components. Moreover, the focus of these papers was to describe the language in terms of its support for reusability, property configuration at various levels and integration with textual languages. The key topics we cover in this paper are the following:

[1] F6 stands for 'Future, Fast, Flexible, Fractionated, Free-Flying spacecraft'.