Journal of Logical and Algebraic Methods in Programming 84 (2015) 67–91

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp

Integrating deployment architectures and resource consumption in timed object-oriented models [☆]

Einar Broch Johnsen [*], Rudolf Schlatte, S. Lizeth Tapia Tarifa

Department of Informatics, University of Oslo, Norway

ARTICLE INFO

Article history:
Received 31 July 2012
Received in revised form 25 November 2013
Accepted 2 July 2014
Available online 21 July 2014

Keywords:
Deployment architecture
Resource management
Object orientation
Formal methods
Performance
Real-Time ABS

ABSTRACT

Software today is often developed for many deployment scenarios; the software may be adapted to sequential, concurrent, distributed, and even virtualized architectures. Since software performance can vary significantly depending on the target architecture, design decisions need to address which features to include and what performance to expect for different architectures. To make use of formal methods for these design decisions, system models need to range over deployment scenarios. For this purpose, it is desirable to lift aspects of low-level deployment to the abstraction level of the modeling language. This paper proposes an integration of deployment architectures in the Real-Time ABS language, with restrictions on processing resources. Real-Time ABS is a timed, abstract and behavioral specification language with a formal semantics and a Java-like syntax, that targets concurrent, distributed and object-oriented systems. A separation of concerns between execution cost at the object level and execution capacity at the deployment level makes it easy to compare the timing and performance of different deployment scenarios already during modeling. The language and associated simulation tool is demonstrated on examples and its semantics is formalized.

© 2014 Elsevier Inc. All rights reserved.

1. Introduction

Software is increasingly often developed as a range of systems. Different versions of a software may provide different functionality and advanced features, depending on the target users. A development method which attempts to systematize this software variability is product line engineering [1]; in a product line, different versions of a software (i.e., the products) may be instantiated with different features. An example is software for cell phones. Products for different cell phones and service subscriptions are produced by selecting among functional features such as call forwarding, answering machine, text messaging, etc. However, the selection of features in a product may be restricted by the hardware capacity of the different targeted cell phones. In addition to their functional variability, software systems need to adapt to different *deployment architectures*. For example, *operating systems* adapt to specific hardware and even to different numbers of available cores; *virtualized applications* are deployed on a varying number of (virtual) servers; and *services on the cloud* may need to dynamically adapt to the underlying cloud infrastructure and to changing load scenarios. This kind of adaptability raises new challenges for the modeling and analysis of component-based applications [2]. To apply formal methods to the design of such systems, it is interesting to lift aspects of low-level deployment concerns to the abstraction level of the modeling language.

☆ This work was done in the context of the EU project FP7-610582 *ENVISAGE: Engineering Virtualized Services* (http://www.envisage-project.eu).
* Corresponding author.
E-mail addresses: einarj@ifi.uio.no (E.B. Johnsen), rudi@ifi.uio.no (R. Schlatte), sltarifa@ifi.uio.no (S.L. Tapia Tarifa).

http://dx.doi.org/10.1016/j.jlamp.2014.07.001
2352-2208/© 2014 Elsevier Inc. All rights reserved.

The motivation for the work presented in this paper is to apply performance analysis to formal object-oriented models in which objects are deployed on resource-constrained deployment architectures. The idea underlying our approach is to make a separation of concerns between the *cost* of performing a computation and the available resource *capacity* of the deployment architecture, rather than to assume that this relationship is fixed in terms of, e.g., specified execution times. Although a range of resources could be considered, this paper focuses on processing capacity. In our approach, the underlying deployment architecture of the targeted system forms an integral part of the system model, but defaults are provided which allow the modeler to ignore architectural design decisions when desirable. The separation of concerns between cost and capacity allows the performance of a model to be compared for a range of deployment choices. By comparing deployment choices many interesting questions concerning performance can be addressed during the system design phase, for example:

- How will the response time of my system improve if I double the number of servers?
- How do fluctuations in client traffic influence the performance of my system on a given deployment architecture?
- Can I better control the performance of my system by means of application-specific load balancing?

Our approach is based on ABS [3], a modeling language for distributed concurrent object groups akin to concurrent objects (e.g., [4–6]), Actors (e.g., [7,8]), and Erlang processes [9]. Concurrent object groups communicate by asynchronous method calls and futures [6]. ABS is an executable imperative language which allows modeling abstractions; for example, functions and algebraic data types can be used to abstract from imperative data structures while retaining an overall object-oriented design. ABS has a formal semantics in an SOS style [10] as well as a tool suite to support the development and analysis of models [11]. The core of this tool suite is an editor in Eclipse with a compiler and a language interpreter executing on Maude [12], a platform for programs written in rewriting logic [13]. The syntax of ABS and its semantics for sequential object-oriented programs are sufficiently similar to industrial programming languages (specifically, Java) that a moderately experienced software engineer can start using it with a reasonably small learning effort.

To model object-oriented applications in resource-constrained deployment architectures, we extend ABS with *deployment components*. Deployment components were originally proposed by the authors in [14]. Deployment components capture the execution capacity of a location in the deployment architecture, on which a number of concurrent objects are deployed. Deployment components are parametric in the amount of concurrent execution capacity they allow within a time interval. This allows us to analyze how the execution capacity of a deployment component influences the performance of objects executing on the deployment component. The authors also extended this approach to support dynamic resource reallocation [15] and object mobility [16]. This paper improves and combines results from [14–16] by, first, giving a *unified* presentation of this work; second, adapting our approach to a dense *real-time* model whereas the previous papers used discrete time; third, refining the cost model of the previous papers [14,15] from fixed costs to the *flexible user-defined cost expressions* introduced in [16]; and fourth, refining the semantics to directly handle slow computations which require several time intervals. To validate and compare the concurrent behavior of models under restricted concurrency assumptions, we use the tool suite for Real-Time ABS.

Paper overview. Section 2 presents the Real-Time ABS modeling language and Section 3 extends Real-Time ABS with a deployment layer to capture deployment architectures and resource consumption. Sections 4, 5, and 6 highlight different aspects of the modeling language through examples and show how the Real-Time ABS tool can be used to obtain insights into the deployment aspects of the models. Section 7 formalizes the modeling language in terms of an operational semantics. Section 8 discusses related and future work, and Section 9 concludes the paper.

2. Modeling timed behavior in Real-Time ABS

ABS is an executable object-oriented modeling language which combines functional and imperative programming styles to develop high-level executable models. ABS targets the modeling of distributed systems by means of concurrent object groups that internally support interleaved concurrency. Concurrent object groups execute in parallel and communicate through asynchronous method calls. A concurrent object group has at most one active process at any time and a queue of suspended processes waiting to execute on an object in the group. This makes it very easy to combine active and reactive behavior in the concurrent object groups, based on a cooperative scheduling [3] of processes which stem from method activations. Objects in ABS are dynamically created from classes typed by interface; i.e., there is no explicit notion of hiding as the object state is always encapsulated behind interfaces which offer methods to the environment.

Inside an object, internal computation is captured in a simple functional language based on user-defined algebraic data types and functions. Thus, the modeler may abstract from many details of the low-level imperative implementations of data structures, and still maintain an overall object-oriented design which is close to the target system. A schematic view of the modeling layers of ABS is given in Fig. 1; this section presents the functional and imperative layers, the deployment layer is discussed in Section 3.

At a high level of abstraction, concurrent object groups typically consist of a single concurrent object; other objects may be introduced into a group as required to give some of the algebraic data structures an explicit imperative representation when this is natural in a model. To simplify the presentation in this paper, we aim at high-level models and only consider concurrent objects (i.e., the groups will always consist of single concurrent objects). We make use of ABS *annotations* (a general mechanism to add meta-data to statements) to express timing and deployment aspects in our models. This pa-

دریافت فوری ← متن کامل مقاله

ISIArticles
مرجع مقالات تخصصی ایران

✓ امکان دانلود نسخه تمام متن مقالات انگلیسی
✓ امکان دانلود نسخه ترجمه شده مقالات
✓ پذیرش سفارش ترجمه تخصصی
✓ امکان جستجو در آرشیو جامعی از صدها موضوع و هزاران مقاله
✓ امکان دانلود رایگان ۲ صفحه اول هر مقاله
✓ امکان پرداخت اینترنتی با کلیه کارت های عضو شتاب
✓ دانلود فوری مقاله پس از پرداخت آنلاین
✓ پشتیبانی کامل خرید با بهره مندی از سیستم هوشمند رهگیری سفارشات