Computer Standards & Interfaces 35 (2013) 30–49

Contents lists available at SciVerse ScienceDirect

Computer Standards & Interfaces

journal homepage: www.elsevier.com/locate/csi

H: A component-based specification language for heterogeneous applications☆

J.A. Fernández-Madrigal [a], L. Llopis [b], A. Cruz-Martín [a], C. Galindo [a,*], J. González-Jiménez [a]

[a] System Engineering and Automation Dpt., University of Málaga, Spain
[b] Computer Science and Programming Languages Dpt., University of Málaga, Spain

A R T I C L E I N F O

Article history:
Received 11 July 2011
Received in revised form 5 March 2012
Accepted 26 March 2012
Available online 2 April 2012

Keywords:
Component-Based Software Engineering
Software architectures
Heterogeneous software applications

A B S T R A C T

This paper presents H, a minimalistic specification language for designing heterogeneous software applications, particularly in the realms of robotics and industria, which takes advantage of a Component-Based Software Engineering (CBSE) approach. H copes with some of the most outstanding characteristics of these systems, like diversity at different levels (hardware platforms, programming languages, programmer skills), network distribution, real time and fault-tolerance. The H specification covers the life-cycle of any heterogeneous application. Its development system offers to the designer and/or builder a set of tools for specifying modules, generating code semiautomatically, debugging, maintenance, and a real time analysis of the system.

© 2012 Elsevier B.V. All rights reserved.

1. Introduction

In the last years, a growing trend exists to use off-the-shelf components in the development of industrial, robotics, and embedded software applications [9], since that should lead to shorter time to market and lower prices. In this class of applications, heterogeneity is not only unavoidable, but also a principal characteristic that sets important requirements during development, as discussed recently in [24]. Heterogeneity may appear at different levels: off-the-shelf hardware components (microcontrollers, computers, robots, AGVs, sensors, actuators, etc.); communication networks that add more diversity, especially in industry (different kinds of networks and/or field buses); hardware components that must be programmed, either in general-purpose languages (with or without operating systems), or in specialized ones; etc. Moreover, heterogeneity goes beyond these issues. For example, the programming skills of the people in charge of developing the application vary depending on their areas of expertise, and also the components are subjected to changes as technology evolves.

Component-Based Software Engineering (CBSE) is an emergent area of research where a number of problems related to heterogeneity can be properly addressed ([14,22,16]). Some of these problems are:

- The existence of different quality standards for different components.
- Defective integration of components, mainly due to the lack of sufficiently mature integration frameworks.
- The challenge of obtaining a high degree of reusing without setting excessive requirements on diverse components.
- Dealing with components which functionality evolves over time.
- Dynamic configuration of components.

There has been a number of development frameworks proposed from the CBSE perspective (see Section 2 for a review), and also standards for coping with isolated aspects of heterogeneous systems design, but we have found that the following issues still need more attention:

- Simplification/minimization of the specification of requirements on components in order to facilitate both their validation and integration, in spite of diversity. In this sense, for instance, current standards like CORBA for communicating distributed heterogeneous software are far from simple ([33]).
- Graceful degradation in the application dependability when components satisfy only a subset of their individual requirements, a common situation in heterogeneous applications.
- Integration of components with different degrees of reliability.
- Coping with the evolution of functionality, at different paces in different components, with minimum impact in the design and implementation of the application.
- Facilitating documentation, versioning, and repository management.
- Providing automatic or semi-automatic generation of code for diverse execution platforms.
- Coverage of the main stages of software development life-cycle, especially design, implementation, debugging, and maintenance.

Taking these issues into account, in this paper we present a new development framework that fits in the area of CBSE, called BABEL, aimed to cope with heterogeneity thoroughly, in the sense that it covers some of the previously highlighted issues simultaneously: it is a minimalistic approach to cope with highly heterogeneous

☆ This work has been partly supported by the Spanish Government under research contracts TIN2011-23795, DPI2008-03527 and DPI2011-25483.

* Corresponding author.
E-mail addresses: jafma@ctima.uma.es (J.A. Fernández-Madrigal), luisll@lcc.uma.es (L. Llopis), anacm@ctima.uma.es (A. Cruz-Martín), cipriano@ctima.uma.es (C. Galindo), jgonzalez@ctima.uma.es (J. González-Jiménez).

0920-5489/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.csi.2012.03.003

applications, identifies explicitly and exhaustively the non-portable parts of a design, covers the most important stages of the software lifecycle and encourages and facilitates the use of standards. For space limitations, the paper is focused on one of its core elements: the H specification language, which makes up the *component model* [25] of the framework, although we also provide an overview of the whole development system where H is included. The H language copes well with:

- Heterogeneity in software and hardware components: the former are specified by separating their portable (those that are not tied to any other particular component) and non-portable parts, at two levels of granularity as it will be explained further on. By dividing in this way the specification, heterogeneity is conveniently and explicitly identified from the beginning, enabling its efficient management along the complete life-cycle.
- Distributed applications: a heterogeneous application based on components can be distributed in a network. H allows the programmer to specify in the same framework both networked and not networked applications. Implementations from this specification can use a variety of software for the network: from direct peer-to-peer communications to more sophisticated middleware (e.g.: [50,60]), including standard libraries (e.g.: Berkeley TCP/IP sockets).
- All-scale design: H allows the programmer to specify from very small components (embedded) to large-scale applications.
- Language independence: no constraints are imposed on the pieces of code included into the specification of a software component in H; they just must serve for specifying sequential operations (concurrency is embedded into the H execution model, as explained in the next sections).
- O.S. and hardware independence: a component is not tied to be executed in any particular O.S. or specific hardware unless actually needed.
- Real-time features: H includes facilities for real-time validation and execution. These include a portable system of real-time priorities (that can be mapped to a variety of existing O.S. real-time supports), time measurement, and worst-case execution time (WCET) specifications. These features, as many others, can be restricted to the subset of components that need them, enabling the graceful degradation of the requirements commented before.
- Fault-tolerance features: H also allows the programmer to specify fault-tolerance behaviors when necessary, based on active and passive replication [30].
- Extensibility and reusability: a specification for a component in H can be extended and reused in a clear and well-specified form. This facilitates coping with technology evolution. Object oriented inheritance is used in the language for these purposes.
- Validation enabled: the specification contains enough information to apply some validation and verification techniques at early stages in the design of the application, as we explain further on.

In addition to these features, the H specification language is embedded into the aforementioned BABEL framework, where a number of tools are available in order to cope with other stages of the development life-cycle. These tools include: a visual case tool for the design of components under H, an application for generating implementations semi-automatically from these specifications, a debugger for testing the execution of the application (including its real-time performance), and a web site for maintenance of specifications and implementations.

We have found that the main advantages of the use of the H language and the BABEL framework are drastically reduced development effort (time, mostly) while guaranteeing from very stringent to very relaxed requirements in different parts of the applications. BABEL has been used mainly for robotic software development from longer than a decade in its different versions ([26,13]), serving for integrating heterogeneous and evolving hardware and software components, and allowing a number of very different people (students, engineers, researchers) to focus on practical and theoretical robotic problems and not on software engineering issues, that were satisfactorily solved by the framework. Currently, a well-proven stable version that includes the previous version of H, called Aracne, and some of the tools, is freely available for public download [13]. The maintenance web site was described in detail elsewhere [26] (currently is under a profound update and revision process).

This paper is structured as follows. Section 2 presents other works relative to existing industrial (non-CBSE and CBSE) frameworks. Section 3 offers a general outlook of the main features of H, illustrated with excerpts from real robotic designs. In Section 4 we focus on the time analysis that can be carried out on the specification of an application. Section 5 presents a case study where a heterogeneous domotic system can be developed using H. Finally, Section 6 sets the conclusions of these work, and the future lines of development we plan to follow.

2. Related work

Firstly, we will offer an outlook of some non-CBSE frameworks that have been proposed in the particular fields of manufacturing systems or robotics, which are of interest due to the especial heterogeneity issues that emerge in these applications.

G++ [2] is an object oriented, pattern-based language that helps to design large distributed software systems with concurrent control requirements. The development process with G++ is an evolutionary one that maps the system analysis into a logical design, and then maps that logical design into a physical one that is finally implemented. The implementation needs frameworks or sets of basic classes that support the low-level environment. There is a commercial version of G++ [66] that has been used in the industry. That version works on Java and C++ platforms, and it includes CASE tools for analysis, design, prototyping and code generation, as well as GUI tools for monitoring and control. Like G++, the H language and tools of BABEL take into account most of the software life-cycle; however, BABEL is not restricted to any particular programming language, and it offers a downloadable free version.

Other relevant non-CBSE manufacturing/robotics frameworks are the SEMATECH CIM [63], aimed to the integration of manufacturing execution systems (MES), that concentrates on the shop-floor control in semiconductor factories (some works, e.g., [37], have enlarged its application domain both inside and outside that particular type of manufacturers), and OSEFA, the Open Software Framework for Manufacturing [58,59], which claims that creating the software control of a manufacturing installation should be based on off-the-shelf solutions, which would help to reduce the development effort and cost.

From the component-based point of view, there are a number of well-known general-purpose software component frameworks, like the OMG's standard CORBA [50], COM [44], .NET [43], XPCOM [47] or EJBS [23], which are suited to general application development. In the following we discuss the most interesting approaches (the interested reader can consult [54] for a deep study of fifteen component models): DisCComp ([7,55]) provides a sound formal semantic component model, however its specifications are close to the coding level, separating it from the more abstract H proposal; Fractal ([19,18]) is another platform independent component model that has been implemented for different particular platforms, such as Julia [18], AOKell [61] or FractNet [27], while the BABEL framework presented in this paper provides a finer support for multiple platforms, as it will be explained later on; Palladio ([36,39]) focuses on performance, including analysis techniques, for example response time analysis, while in our proposal the analysis can be performed without an implementation, supporting early design time predictions; finally, KLAPER ([28,29]) is an

Download English Version:

https://daneshyari.com/en/article/453377

Download Persian Version:

https://daneshyari.com/article/453377

Daneshyari.com