Journal of Systems Architecture 60 (2014) 770–781

Contents lists available at ScienceDirect

Journal of Systems Architecture

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

An architectural approach with separation of concerns to address extra-functional requirements in the development of embedded real-time software systems

Marco Panunzio *, Tullio Vardanega

Department of Mathematics, University of Padova, via Trieste 63, 35121 Padova, Italy

ARTICLE INFO

Article history:
Received 16 February 2012
Received in revised form 22 March 2014
Accepted 9 June 2014
Available online 19 June 2014

Keywords:
Embedded real-time systems
Extra-functional properties
Software architecture
Component-based software engineering
Separation of concerns

ABSTRACT

A large proportion of the requirements on embedded real-time systems stems from the extra-functional dimensions of time and space determinism, dependability, safety and security, and it is addressed at the software level. The adoption of a sound software architecture provides crucial aid in conveniently apportioning the relevant development concerns. This paper takes a software-centered interpretation of the ISO 42010 notion of architecture, enhancing it with a component model that attributes separate concerns to distinct design views. The component boundary becomes the border between functional and extra-functional concerns. The latter are treated as decorations placed on the outside of components, satisfied by implementation artifacts separate from and composable with the implementation of the component internals. The approach was evaluated by industrial users from several domains, with remarkably positive results.

1. Introduction

Embedded real-time systems in general are characterized by two distinctive features: (1) they are resource-constrained, since most often only scarce processing power and memory space are available; and (2) the growing incidence of requirements that address concerns over and above system functionality, in the dimensions of time and space determinism, dependability, safety and, increasingly, security [1].

In this work we collectively refer to those requirements as *extra-functional*, to signify that, while they do not concur to the functional activity of the system, they crucially contribute to the ultimate quality of the system. We also use the term *property* for a feature of an implementation item that provably meets a requirement placed on the specification of that feature. We therefore have extra-functional requirements and extra-functional properties. In this paper we refer to the latter by the shorthand EFP.

Understanding, providing and asserting EFP has an increasingly large and costly effort footprint on the development process in a variety of application domains. The embedded real-time software systems industry therefore seeks ways to accommodate attention for EFP in their otherwise consolidated development practices without breaking the integrity of the overall process.

The central tenet of this work is that the adoption of a sound software architecture helps achieve a clear-cut and composable apportionment of development concerns. With that, EFP can be addressed aside from, yet in coordination with, the functional dimension, in keeping with the established principle of separation of concerns [2].

Interestingly, once the step is taken to put the software architecture at the center of the development strategy, product line concerns can be accommodated by elevating that notion to the stipulation of a *reference* software architecture. We draw from [3] that the reference software architecture is a common and agreed architectural framework capable of addressing all the relevant industrial needs, providing a recurrent solution to the development of a certain class of software systems. To that we add the capability of operating as a single and consistent basis for addressing EFP.

This paper reports on the lessons learned in pursuit of that vision in a large and encompassing research program that progressed along two parallel and complementary lines. One line of the research took place as part of an initiative launched by the European Space Agency (ESA) targeting the definition and realiza-

* Corresponding author. Tel.: +39 0498271359.
E-mail addresses: panunzio@math.unipd.it (M. Panunzio), tullio.vardanega@math.unipd.it (T. Vardanega).

http://dx.doi.org/10.1016/j.sysarc.2014.06.001
1383-7621/

tion of a reference software architecture that should steer the development of on-board software for satellites across all of its software supply chain, using the component-based approach described in [4,5]. That initiative started from the capture of all product-line needs by the domain stakeholders and concluded with their validation mapping to the chosen reference software architecture. The other line of research occurred within the ARTEMIS JU CHESS project[1] ("Composition with Guarantees for High-integrity Embedded Software Components Assembly" 2009–2012), which aimed at the realization of a model-based component-oriented approach for the development of embedded real-time software systems for telecom, space, and railway applications [6]. All industrial parties from both actions subscribed from the outset to the principle of addressing EFP separately from the functional dimension. In doing so, they were witness to the unifying power of the software architecture concept and to the evidence that EFP were indeed addressed at a distinct level of abstraction as well as at a distinct step in the development process, in overlay to the functional specification of the software.

The remainder of the paper is organized as follows. Section 2 recalls the essential aspects captured in the concept of software architecture and argues why that concept is useful to address concerns in dimensions other than functional. That discussion proceeds into an original interpretation of the notion of reference software architecture and of its founding principles. Section 3 relates the work presented in this paper with state-of-the-art approaches that address similar goals. Section 4 describes the essential details of the proposed approach, with special focus on the way it assists the specification and assures the fulfilment of the extra-functional properties expected of the system. Section 5 presents an instantiation of the reference software architecture to a variety of industrial domains, and discusses how it meets the stakeholders' needs captured as part of the research effort. Section 6 discusses four extensive industrial case studies on which the approach presented in this paper was applied and successively evaluated. Section 7 draws some conclusions and outlines future work.

2. The role and potential of the software architecture

2.1. Common understanding

At odds with its intrinsic centrality, the software engineering practice harbors an exceedingly informal and liberal interpretation of the concept of software architecture. Most practitioners regard software architecture as a synonym to software design: reference [7] collects a large set of community definitions that portray the confusion. In actual fact, the software architecture is a much larger scope than that, with ample bearing on the principles that guide the design and evolution of the software system. The definition given in IEEE 1471, later promoted to ISO 42010 [8] clarifies, to our satisfaction, that an architecture is "the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution". When applied to software systems, the definition of software architecture captures well the following concerns:

- *Software decomposition*: the organization of the software in terms of parts, so that every individual part has its own architectural cohesiveness (it addresses a single well-defined part of the problem), and the interactions between parts are minimized so as to reduce unnecessary dependencies (i.e., coupling), hence reducing incidental complexity of understanding, verification and validation, operation and maintenance;
- *Externally visible attributes of software "components"*: those attributes represent features or needs that are specific to individual components yet can influence other parts of the software or determine properties of the whole. The other attributes, if any, shall remain as (externally invisible) internal details and will not be used for the overall reasoning at the level of the software architecture;
- *Relationship between software "components"*: how components relate to one another in providing services and fulfilling needs;
- *Extra-functional concerns*: the abstraction level at which extra-functional concerns are to be addressed;
- *External interfaces*: the way the software interacts with the external environment (e.g., by commanding sensors or actuators, or serving external interrupts);
- *Principles for the development and evolution of the software*: the software design process that fixes the rules for development, maintenance and evolution, and dictates the supported form of software reuse;
- *The rules in place to warrant the consistent relationship between all of the above concerns*: a methodology capable of encompassing and consistently harmonizing all the aspects listed above; additionally, the methodology shall provide criteria to determine whether a software part can be included in the system, as it conforms to the principles sustained by the architecture, or it shall be rejected, as its inclusion would break system integrity.

A reference software architecture prescribes the concrete form of the software architectures that shape the specific systems for which it was originally developed. The reference software architecture can thus be regarded as a sort of "generic" software architecture that prescribes the founding principles, the underlying methodology and the architectural practices recognized by the domain stakeholders as the baseline solution to the construction of a certain class of software systems in that domain. Symmetrically, the software architecture of one particular system for a given domain can then be regarded as an "instantiation" to the specific system needs of the reference software architecture for that domain.

2.2. Narrowing to our context

The reference software architecture chosen for the ESA initiative – and later reflected in the whole span of investigation covered in this work – has a number of distinctive features, which are best introduced here before proceeding further.

First of all, the reference software architecture includes a companion component-oriented development methodology. Software systems in the industrial domains that adopt that methodology are therefore developed by defining (or reusing) components and by creating assemblies of them. The interpretation of components varies with the specific goals of the approach they serve. Numerous definitions for them indeed exist [9–11]. Suffice it to say for now that a component is the unit of design and of encapsulation of our approach: we return to this definition later in this paper.

Secondly, the reference software architecture was formulated so as to support the principle of separation of concerns. This is a long-known but much neglected practice first advocated by Dijkstra in [2], which strives to part the various aspects of software design and implementation so as to enable separate reasoning and focused specification for each of them. Separation of concerns is applied to the component model that rests at the very core of the reference software architecture. We consequently pursue it at a level of abstraction much higher than programming, as done

[1] http://www.chess-project.org/.