The Journal of Systems and Software 83 (2010) 937–958

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Ambient-PRISMA: Ambients in mobile aspect-oriented software architecture

Nour Ali [a,*], Isidro Ramos [b,1], Carlos Solís [a,2]

[a] Lero – The Irish Software, Engineering Research Centre, University of Limerick, Limerick, Ireland
[b] Department of Information Systems and Computation, Universidad Politecnica de Valencia, Camino de Vera s/n, E-46022 Valencia, Spain

ARTICLE INFO

Article history:
Received 30 November 2008
Received in revised form 23 November 2009
Accepted 8 December 2009
Available online 16 December 2009

Keywords:
Aspect-oriented software architectures
Mobility
Middleware
Distribution
Model driven engineering
Ambients

ABSTRACT

This work presents an approach called Ambient-PRISMA for modelling and developing distributed and mobile applications. Ambient-PRISMA enriches an aspect-oriented software architectural approach called PRISMA with the ambient concept from Ambient Calculus. Ambients are introduced in PRISMA as specialized kinds of connectors that offer mobility services to architectural elements (components and connectors) and are able to coordinate a boundary, which models the notion of location. Mobility of architectural elements is supported by reconfiguring the software architecture. This paper presents a metamodel that introduces ambients to design aspect-oriented software architectural models for mobile systems. The design of models is performed using an Aspect-Oriented Architecture Description Language. A middleware called Ambient-PRISMANET which maps the metamodel to .NET technology and supports the distributed runtime environment needed for executing mobile applications is also presented. In addition, a CASE Tool which allows users to specify the aspect-oriented architectural models in a graphical way and generate .NET code is provided. In this way, we explain how Ambient-PRISMA follows Model Driven Engineering. An example of an auction system is used throughout the article to illustrate the work.

1. Introduction

In the last few decades, the information society has undergone important changes which have increased the complexity of software development. New technologies have become part of our daily life and the Internet has been established as a framework for global knowledge. The software, the devices (PCs, laptops, PDAs, smart phones, etc) and people involved in current business processes are usually distributed, and mobile. As a result, the structure of software systems has become more complex due to the fact that these systems need to take into account new requirements such as adaptability, and security.

Software architecture is a discipline that focuses on the design and specification of overall system structure (Shaw and Garlan, 1996; Bass et al., 2003). It is considered to be the bridge between the requirements and implementation phases of the software life-cycle. The software architecture of a system describes its structure in terms of components (computational units), connectors (coordination units), and configurations (connection of components and connectors) (Medvidovic and Taylor, 2000). Reconfiguration is the change of the system structure. A kind of reconfiguration that can be found in distributed software systems is mobility. For example, Carzaniga et al. (1997) define code mobility as the capability to reconfigure dynamically, at runtime, the binding between the software components of the application and their physical location. Thus, code mobility causes the structure of a distributed system to change by creating and removing bindings (or connections) between software components and physical locations.

Aspect-Oriented Software Development (AOSD) (Filman et al., 2004) is also a technique that reduces complexity by increasing reusability, flexibility, and maintainability across the software development process. AOSD modularizes crosscutting concerns in separate entities called aspects (Kiczales et al., 2001). Distribution and mobility have been identified as crosscutting concerns; separating them in aspects increases their reusability, and decreases maintenance costs (Lobato et al., 2004; Lopes, 1997; Soares and Borba, 2002). For example, Lopes compared the code of nine case studies implemented in plain Java with the code obtained in the DJ aspect-oriented approach (Lopes, 1997). The results obtained were that in four of the case studies the number of lines achieved in plain Java were the same as the ones achieved in DJ, and five of the case studies the number of lines achieved in DJ were reduced in comparison with the plain Java code. The results demonstrate that distributed applications can benefit from AOSD as the lines of code can be reduced and the distribution concerns are well localized for maintenance tasks.

Research works which integrate AOSD and software architecture have been proposed (Chitchyan et al., 2005; Cuesta et al.,

* Corresponding author. Tel.: +35 3 61 233799.
E-mail addresses: Nour.Ali@lero.ie (N. Ali), iramos@dsic.upv.es (I. Ramos), Carlos.Solis@lero.ie (C. Solís).
[1] Tel.: +34 96 387 73 50.
[2] Tel.: +35 3 61 233799.

0164-1212/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2009.12.009

2005). Introducing aspects in software architecture improves the modularization of crosscutting concerns that cannot be reflected in software architecture concepts (such as components, or connectors) (Batista et al., 2006)). This facilitates the evolution of software systems (Tarr et al., 1999)), and conserves the traceability of aspects across the software life-cycle (Baniassad et al., 2006).

Currently, one of the existing challenges is to allow code generation from software architecture. Model Driven Engineering (MDE) (Schmidt, 2006) (also known as Model Driven Development) is a software development approach based on transformations between models. MDE proposes the development of software by using models that describe a system in a technology-independent way. These models can be transformed into technology dependent models, which describe a system based on a specific technology. Two main approaches exist for supporting MDE through tools: the Model Driven Architecture (MDA) proposed by the Object Management Group (OMG) (Object Management Group, 2003), and the Software Factories proposed by Microsoft (Greenfield et al., 2004).

This paper presents an approach for developing distributed and mobile applications called Ambient-PRISMA. Ambient-PRISMA enriches an aspect-oriented software architecture approach called PRISMA with the ambient concept inspired from Ambient Calculus (AC) (Cardelli and Gordon, 1998; Cardelli, 1998). An ambient is a bounded place where computation happens such as a phone, a PC, a folder or a network. AC provides an explicit and abstract way for modelling locations, as well as a realistic mobility model that supports mobile computation (mobile code) and mobile computing (mobile devices) in a unique way.

AMBIENT-PRISMA introduces ambients as new kinds of architectural elements which define a bounded place where other architectural elements reside and are coordinated with the exterior of the boundary. Ambients can be hierarchically organized, conforming to a tree structure that is used to model distributed systems hierarchies. Architectural elements (including ambients) can move by entering and exiting ambients. Thus, mobility causes software architecture to reconfigure which involves removing mobile architectural elements from their originating ambients and adding them to their destination ambients, as well as creating and deleting connections among architectural elements. The functionality (behaviour) of an ambient is defined through mobility, coordination and distribution aspects.

Ambient-PRISMA follows MDE. It provides a metamodel for designing aspect-oriented software architectural models of mobile systems in a technology-independent way, a middleware called Ambient-PRISMANET, and a CASE Tool. The Ambient-PRISMANET middleware maps the metamodel to .NET technology and supports the distributed runtime environment needed to execute mobile applications. It also provides reconfiguration mechanisms needed for supporting mobility. The CASE Tool allows users to specify the aspect-oriented architectural models in a graphical way and generate .NET code of mobile applications.

The structure of the paper is as follows: Section 2 gives an overview of the PRISMA approach. Section 3 explains an Auction System example used to illustrate the work presented in this paper. Section 4 explains the characteristics of AMBIENT-PRISMA. Section 5 defines the AMBIENT-PRISMA metamodel basing on the PRISMA one. Section 6 shows how AMBIENT-PRISMA software architectural models can be defined. Section 7 introduces the AMBIENT-PRISMA-NET middleware, which permits the execution of AMBIENT-PRISMA models. Section 8 presents the AMBIENT-PRISMA Case tool that is used for modelling, and executing distributed and mobile applications by means of Model Driven Engineering techniques. Section 9 presents related work in aspect-oriented software architectures and mobility. Finally, conclusions and future work are presented in Section 10.

2. PRISMA

PRISMA (Pérez et al., 2008) is an approach that includes a metamodel (Pérez et al., 2005), an Aspect-Oriented Architecture Description Language (Pérez et al., 2006), and a tool for developing software systems from their aspect-oriented software architectures. PRISMA integrates the AOSD and the Component-Based Software Development (CBSD) (Szyperski, 2002) for defining software architectures. PRISMA uses AOSD to separate crosscutting concerns (safety, coordination, etc.) of architectures in aspects.

In PRISMA, there are two kinds of architectural elements: components and connectors. A component captures the logic of software systems and does not act as a coordinator. A connector acts as a coordinator among other architectural elements. An architectural element can be seen from an external and an internal view. In the external view (also known as black box view (Miles and Hamilton, 2006), see Fig. 1), an architectural element encapsulates its functionality and publishes a set of services that are offered to the rest of the architectural elements. These services are grouped into interfaces that are published through ports.

In the internal view (also known as white box view (Miles and Hamilton, 2006)), an architectural element is seen as a prism, where each side is an aspect. In this way, architectural elements are represented as a set of aspects and weaving relationships among them (see Fig. 1). An aspect defines the properties and behaviour of an architectural element from a concrete crosscutting concern. The behaviour defined in aspects specifies how and when services are executed. These services can be part of interfaces. In PRISMA, the difference between a component and a connector is that a component cannot have a coordination aspect, and a connector cannot have a functional aspect but must have a coordination

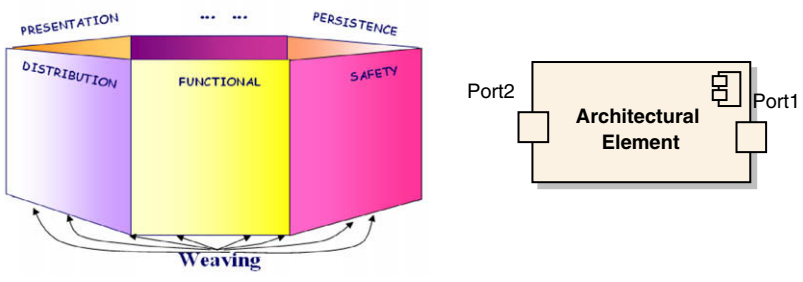

Fig. 1. A PRISMA architectural element.