An Agent and Pattern-Oriented Approach to Data Visualization

A

Chung-Yeung Pang
https://orcid.org/0000-0002-7925-4454
Seveco AG, Switzerland

Severin K. Y. Pang
Cognitive Solutions and Innovation AG, Switzerland

INTRODUCTION

The standard style of software development is to analyze the problem, develop a design to solve the underlying problem, and implement the design into an application. As a result, the application is strongly tied to the problem to be solved. With new requirements, the application must be extended to solve the new problem. This way of software development is fine in many cases. However, applications built in this style may not work well when dealing with big data, which contains a wide variety of data types and structures.

In order to be able to handle a large amount of data, the software system must be flexible and agile. This chapter uses an example to show that following traditional standard programming would result in rigid and complex software components. An approach is presented that combines generic programming, pattern-oriented programming, and agent-oriented programming to build flexible and agile software systems. The approach is used to build a data visualization system that can handle a large variety of data. The visualization system is part of a web and mobile geography application. This application has to process thousands of physical and human geography topics with hundreds of data sources from the internet. The focus of this chapter is the general technical approach to building a flexible, scalable, and agile visualization system that can handle this amount of data. Code segments are used for illustration. The programming language used in the presentation is Dart with the Flutter framework. Flutter is a new product from Google and is widely used for web and mobile app development. The Dart language is similar to Javascript. Anyone with programming skills in C, C++, Java or Javascript can easily follow the code segments.

In this chapter, techniques are presented on how software components and patterns can change their behavior depending on the data context through reflection. The main motivation of aspect-oriented programming, the separation of concerns, is also discussed. In order to be able to pursue this approach, a paradigm shift in programming is required. This shift is also presented in this chapter.

The content of this chapter first deals with the background. This is followed by a section on generic programming, a section on patterns and frameworks, and a section on data, context, and agents. At the end, a conclusion is drawn, along with a section on the consequences and applicability of the development approach and a section on future research and direction.

DOI: 10.4018/978-1-7998-9220-5.ch074

Background

This section provides background information about the programming paradigm and the agile development process, the agent, and big data.

Programming, Paradigms, and Agile Development Process

In the early days of software history, programmers tended to develop their programs without documentation in an ad-hoc style. The programs are usually not structured and organized. One result of this programming style was the software crisis of the 1960s, 1970s and 1980s (Software Crisis, 2010).

Structural programming (Jackson, 1975) was introduced in the 1970s to combat spaghetti code resulting from the ad hoc style of programming. Much emphasis has been placed on how a program is well structured. However, it failed to handle the complexities of enterprise applications. At the end of the 1980s, object-oriented (OO) programming began to spread. Software scientists claim that OO languages were designed for programming on a large scale (Wegner, 1989). The OO paradigm with polymorphism and inheritance was a solution to resolve and control the complexity of enterprise applications.

Despite its promises, the OO paradigm was not the ultimate destination of the programming paradigm's journey. Thereafter, various programming paradigms were proposed with great promises. Much research has been carried out on these paradigms. These include aspect-oriented programming, pattern-oriented programming and agent-oriented programming.

Aspect-oriented programming deals with the separation of concerns. For example, technical infrastructure code and business logic code should not be mixed in one software component. AspectJ (AspectJ, 2021) is a practical aspect-oriented extension of the Java programming language. This chapter also takes this concept into account.

We live in a world full of patterns. Our habits are nothing more than repetitively following a series of behavior patterns. The publication of the book by Gamma et al. (1995) made software developers aware of patterns that we use all the time and that should be used in software development. A pattern is defined as the solution to a problem in a specific context. Pattern-oriented programming is a programming approach to identify the problem to be solved and to look for the pattern that provides the solution to the problem in the given context. Patterns in the context of this chapter differ from those of Gamma et al. (1995) and other authors (Weiss, 2003; Hohpe, & Woolfe, 2004). They are not abstractions of a software design. This chapter uses the properties of a pattern proposed by Pang (2020). A pattern can provide a solution through a set of code, composing other patterns together, getting resources, and activating a set of actions to handle a problem in the requests.

Agent-oriented programming (AOP) was introduced by Shoham within his Artificial Intelligence studies in 1990 (Shoham, 1990). His definition of AOP is the following:

AOP can be viewed as a specialization of object-oriented programming. The state of an agent consists of components called beliefs, choices, capabilities, commitments, and possibly others; for this reason, the state of an agent is called its mental state. The mental state of agents is captured formally in an extension of standard epistemic logics: beside temporalizing the knowledge and belief operators, AOP introduces operators for commitment, choice and capability. Agents are controlled by agent programs, which include primitives for communicating with other agents. In the spirit of speech-act theory, each communication primitives is of a certain type: informing, requesting, offering, and so on.