Information and Software Technology xxx (xxxx) xxx–xxx

Contents lists available at ScienceDirect

Information and Software Technology

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

Do the informal & formal software modeling notations satisfy practitioners for software architecture modeling?

Mert Ozkaya

Altinbas University, Department of Computer Engineering, Istanbul TURKEY

ARTICLE INFO

Keywords:
A survey
Software architectures
Informal/formal semantics
ADLs
UML
Formal specification languages

ABSTRACT

Context: Software architectures can be modeled using semantically informal (i.e., ambiguous) or formal (i.e., mathematically precise) software modeling notations.
Objective: In this paper, 115 different practitioners from 28 different countries who work in companies that perform software development have been surveyed. The goal is to understand practitioners' knowledge and experience about informal and formal modeling notations in specifying software architectures.
Method: The survey consists of 35 different questions, divided into three parts, i.e., the participant profile, the informal modeling, and formal modeling. The results of the survey lead to many interesting findings:
Results: (1) Informal software modeling notations (especially UML) are practitioners' top choice in specifying software architectures (94%). (2) Despite many informal languages, some practitioners (40%) insist using informal ad-hoc techniques (i.e., simple boxes/lines and natural languages) to specify complex design decisions (e.g., structure, behaviour, and interaction). (3) Practitioners using informal notations are impressed by their low learning-curve (79%), visuality (62%), and general-purpose scope (66%). (4) Practitioners still criticise informal notations too, essentially for the lack of support for complex design decisions and their exhaustive analysis. (5) While formal modeling notations bridge this gap mentioned in step-4, many practitioners (70%) rarely use formal notations due essentially to the high-learning curve, the lack of knowledge among stakeholders, and the lack of popularity in industry. (6) Among the considered formal notations (i.e., process algebras, high-level formal specification languages, and architecture description languages (ADLs)), process algebras are never used and ADLs are the least used formal languages are ADLs (i.e., 12% frequently use ADLs). (7) Practitioners complain about ADLs' weak tool support (38%) and their lack of tutorials/guidance/etc (33%).
Conclusion: The survey results will let the community realise the advantages and disadvantages of informal and formal modeling notations for software architecture modeling from practitioners' viewpoint.

1. Introduction

Software architectures have been proposed in the nineties as a way of specifying important design decisions for software systems and their reasoning. As Garlan and Shaw proposed in their seminal work [1], several complex design decisions, including structural (i.e., the components and connectors), behavioural (the component behaviours), interaction (the interaction protocols), non-functional property (e.g., reliability and performance), and concurrency (i.e., synchronisation and data access) decisions can be specified and reasoned about at the early stages of software design.

1.1. Software architecture modeling

Design decisions made for a software system to be built are documented as an architectural model, which can be communicated among different stakeholders, analysed for design errors, and improved to better meet the software requirements [2]. Architecture modeling is performed via the modeling notations that allow for textually or visually representing the architectural models at various levels of abstractions. Architectural modeling notations can be considered as simple as any natural language or simple boxes/lines or any modeling languages with syntax and semantics (e.g., ADLs [3]). Modeling notations can be distinguished in various aspects, such as the textual/visual specifications of software architectures, semantical basis, general-purpose/domain-specific scope, language features (e.g., extensibility, multiple views, etc.), and tool support for different operations (e.g., analysis and code generation). In this study, architectural modeling notations are considered based on their semantical basis, which can be defined either formally or informally. The software modeling notations with formal semantics (aka formal modeling notations) are defined mathematically using formal methods [4]. Formal modeling notations

E-mail address: mert.ozkaya@kemerburgaz.edu.tr.

https://doi.org/10.1016/j.infsof.2017.10.008
Received 16 May 2017; Received in revised form 30 September 2017; Accepted 12 October 2017

enable precision, which promotes the detailed and exact specifications of software architectures. Also, formal architectural models can be analysed exhaustively using the formal techniques such as model checking and theorem proving, which are based on mathematical proofs. The software modeling notations without formally defined semantics are called as informal modeling notations in this paper. The semantics of informal modeling notations are either defined informally in plain English or semi-formally. Lacking in precision, informal notations may not always enable specifying the details of software architecture models and thus the resulting models may be ambiguous that can be interpreted by different stakeholders differently.

1.1.1. Informal modeling notations

Informal architecture modeling notations can be considered in three groups, i.e., *(i)* simple boxes/lines, *(ii)* natural languages (e.g., English), and *(iii)* modeling languages. Natural languages (e.g., English) and simple boxes/lines allow for specifying any complex design decisions without having to stick to any notations. Note that the former is textual and the latter is graphical. However, the specifications in natural languages and simple boxes/lines may not be executed via analysis tools for, e.g., syntax/semantics checking, behaviour analysis or code generation.

Unlike natural languages and simple boxes/lines techniques, informal software modeling languages offer concrete notations that have well-defined syntax and semi-formal semantics (if any). UML [5] is one of the most well-known informal modeling languages, which offers visual diagram types for specifying several important design decisions including structural, behavioural, interaction, and deployment. UML has inspired many other informal modeling languages (e.g., SysML [6], Secure-UML [7], and Agent-UML [8]), which either extend UML's notation for meeting new requirements or adapt UML for particular domains. Besides UML and its derivatives, there are architecture description languages (ADLs) that offer architecture-oriented notation sets consisting of architectural elements such as components, ports, connectors, and connections [3,9]. The literature includes tens of different ADLs that vary in many aspects, such as tool support, language features (e.g., extensibility and multiple views), language definition (e.g., textual/visual notation sets or formal/informal semantics)[1]. Some other informal notations include the domain-specific languages (e.g., Liszt [10] for PDE solvers and NDL [11] for device derivers), Business Process Modeling Languages (BPMLs) [12], etc.

1.1.2. Formal modeling notations

Formal modeling notations can be examined in two parts, i.e., *(i)* formal specification languages and *(ii)* formal ADLs. Formal specification languages offer notations for precisely specifying complex design decisions, which can then be formally (i.e., exhaustive) analysed against safety and liveness properties. Some formal languages are also supported with the formal analysis tools that exhaustively check for non-functional properties. The most common formal specification languages are process algebras [13], which allow for specifying systems in terms of processes that interact with each other concurrently (e.g., CSP [14], FSP [15], and pi-calculus [16]). There are also high-level formal specification languages that practitioners can more easily learn and use for specifying their software systems. Some of them offer general-purpose scope (e.g., ProMeLa [17], LOTOS [18], and Alloy [19]); some offer domain-specific scopes (e.g., UPPAAL's input language [20] for real-time systems); and, some are platform-specific (e.g., Java Modeling Language (JML) [21] for Java and Spec# [22] for C#).

The formal ADLs are considered in this survey as the ADLs that offer formally precise architecture-oriented notation sets. Formal ADLs are examined here in three groups, i.e., early general purpose ADLs, domain-specific ADLs, and extensible ADLs. General-purpose ADLs (e.g., Wright [23], Darwin [24], and Rapide [25]) offer high-level notation sets for specifying any types of systems and mainly focus on the formal analysis of architectural specifications. Domain-specific ADLs (e.g., AADL [26] for embedded systems) offer relatively low-level (i.e., more detailed) notation sets for the domain of interest. They essentially tend to focus on formally analysing the low-level specifications for non-functional properties (e.g., AADL's tools for performance analysis) and generating low-level software code (AADL's tool for *C* code). Lastly, extensible ADLs' notation sets (e.g., e.g., Acme [27] and XADL [28]) allow practitioners to extend a generic language with several features such as formal semantics, formal analysis, complex connectors, and tool support.

1.2. Paper motivation and goal

While there are too many informal and formal software modeling notations existing today, it is not possible to understand *(i)* practitioners' knowledge and experience about informal and formal modeling notations and *(ii)* what motivate/demotivate practitioners for using either type of the notations. The existing analytical studies on software architectures and modeling languages rather focus on different issues including the definition of software architecture, practitioners' needs and concerns about software architecture, ADLs' support for several different features (e.g., expressiveness, tool support, analysis, usability, realisability), or practitioners' expectations from ADLs.

Therefore, in this study, a survey has been conducted, which focuses on understanding the practitioners' level of knowledge and experience and their views about the informal and formal software architecture modeling notations. With the survey results discussed in this paper, the goal is to enlighten the software architecture community about the level of tendency towards informal and formal modeling notations and their motivating/demotivating features for practitioners. It will also be possible to learn about the popular informal and formal software modeling notations that practitioners highly prefer for software architecture specifications or the notations that are rarely used.

1.3. Paper structure

In the rest of the paper, the similar surveys in the literature are discussed firstly. Next, the survey's research questions are introduced, which is followed by the explanations of the survey design, execution, and sampling respectively. Then, the survey results are discussed statistically in three parts, i.e., the participants profile, informal architecture modeling notations, and formal modeling notations respectively. Lastly, the discussion and conclusion are given.

2. Related work

According to the up-to-date research of literature, there are more than 120 different known architecture modeling notations[2], which can be categorised based on their semantics definitions as discussed in Sections 1.1.1 and 1.1.2 (i.e., formal or informal). Unsurprisingly, UML is by far the top-used informal modeling language by practitioners [29], which inspired many other notations, e.g., Systems Modeling Language (SysML) [6] for system engineering requirements, Agent-UML [8] for the specifications of multi-agent software systems, and Secure-UML [7] for the specifications of non-functional properties and their analysis. Some modeling notations even formalise UML for the precise specifications of UML architectures and their formal analysis. UMLsec [30] is one of the most prominent UML extension that formalises UML for the

[1] The following link leads to the up-to-date list of ADLs and those indicated to have no formally defined semantics are the informal ADLs: https://sites.google.com/site/ozkayamert1/als.

[2] See the analysis of the architecture modeling notations in https://sites.google.com/site/ozkayamert1/als.