Pratisyenlerin Yazılım Mimarisi Bakış Açıları Üzerine Bilgi ve Tecrübelerini Anlamaya Yönelik Bir Anket Çalışması

Mert Ozkaya

Yeditepe University, Atasehir, Istanbul
mozkaya@cse.yeditepe.edu.tr

Özet. Yazılım mimarisi bakış açıları yazılım mimarilerinin değişik bakış açıları cinsinden modüler bir şekilde kısımlara bölünmesini ve her bir kısımın o bakış açısı ile ilgili tasarım kararlarına odaklanmasını hedefler. Bu bildiride, yazılımcıların değişik bakış açıları üzerine olan bilgi ve tecrübelerini anlamaya yönelik farklı endüstrilerde çalışan 20 farklı ülkeden 56 pratisyenin katıldığı bir anket düzenlenmiştir. Anket, pratisyenler tarafından sıklıkla kullanıldığı düşünülen farklı yazılım mimarisi bakış açılarına odaklanmıştır: mantıksal, davranış, eşzamanlılık, fiziksel, ve dağıtım bakış açıları. Anketten çıkan sonuçların bazı ilginç olanları şöyledir: *(i)* mantıksal, davranış, fiziksel, ve dağıtım bakış açıları bir hayli ilgi görürken, eşzamanlılık bakış açısı pratisyenler tarafından fazla ilgi görmemiştir; *(ii)* mantıksal bileşenlerin tasarımında en çok tercih edilen yapısal birim harici arayüzler olurken, en az tercih edilen ise içsel hesaplama birimi olmuştur; *(iii)* en çok tercih edilen mantıksal konektör tipi asenkron event konektörleri olarak belirlenmiştir; *(iv)* kompleks konektör tipleri (adaptör, dağıtıcı, arabulucu gibi) pek ilgi görmemektedir; *(v)* yazılım mimarilerini değişik bakış açılarından modellemede en çok tercih edilen notasyonlar kutucuklar ve çizgiler notasyonu ile doğal diller (İngilizce gibi) olurken, yazılım mimarisi modelleme dilleri ve biçimsel diller pek kullanılmamaktadır; *(vi)* yazılım sistemlerinin farklı bakış açıları kullanarak tasarlamada pratisyenlerin en büyük motivasyonu farklı tasarım kararlarının dokümantasyonu ve iletişimi olarak belirlenmiştir; *(vii)* bileşenlerin etkileşim davranışlarının tasarımı bir hayli ilgi görmektedir; *(viii)* değişik bakış açılarındaki modellerin birbirleri arasındaki ilişkilerinin tasarımının, modelleme notasyonlarının eksikliklerinden dolayı her zaman başarılamadığı gözlemlenmiştir; ve son olarak, *(ix)* pratisyenlerin sistemlerinin davranışsal tasarımlarında en çok önem verdiği kalite özellikleri ölçeklenebilirlik, performans, ve güvenlik olarak gözlemlenirken, dağıtım tasarımları için ise ölçeklenebilirlik ve kullanılabilirlik özelliklerinin öne çıktığı gözlemlenmiştir.

Anahtar Kelimeler. Yazılım Mimarileri, mimari bakış açıları, anket, yazılım modelleme dilleri, pratisyenler

Towards Understanding Practitioners' Knowledge and Experiences on the Software Architecture Viewpoints: A Survey

Mert Ozkaya

Yeditepe University, Atasehir, Istanbul
mozkaya@cse.yeditepe.edu.tr

Abstract. Architecture viewpoints promote separating software architectures into different viewpoints that each address the particular aspect of a software system. This paper discusses a survey conducted among 56 practitioners from 20 different countries who are involved in software development and aims at understanding their knowledge and experience about five important architectural viewpoints – i.e., logical, behaviour, concurrency, physical, and deployment. Some of the interesting survey results are as follows: *(i)* while the logical, behaviour, physical, and deployment viewpoints are widely used, the concurrency viewpoint is not so; *(ii)* the top-preferred structural unit for a logical component is the external interfaces and the least-preferred is the internal computation unit. *(iii)* the top-preferred simple connector type is the asynchronous events; *(iv)* the complex connectors (e.g., adaptor and arbitrator) are not as popular as simple connectors; *(v)* boxes and lines diagram and natural languages (e.g., English) are the top-preferred software modelling notations for each viewpoint considered, while architectural languages, and formal specification languages are never used by many; *(vi)* documenting design decisions and their communication is the main source of motivation for each viewpoint; *(vii)* the specifications of the interaction behaviours of components are highly desired by practitioners; *(viii)* mapping between different viewpoints cannot always be achieved due to the inadequate software modelling notations; and *(ix)* scalability, performance, and security are the top-considered quality properties for the behaviour viewpoint, while scalability and availability are the top-considered ones for the deployment viewpoint.

Keywords: Software architectures, architectural viewpoints, survey, software modelling languages, practitioners

1 Introduction

Software architecture is considered as the blueprint of a software system to be built, in which the low-level details of the system are abstracted and the high-level important aspects that play key roles in meeting the functional and non-functional requirements of the system are focused on [5, 32]. To facilitate the

specifications of software architectures, architectural viewpoints have been proposed, which basically promote the separation of concerns and modularise the software architecture designs into separate perspectives that each addresses the design decisions about a particular aspect [15,34,38]. To model software systems from different architecture viewpoints, practitioners may use different sorts of notations. These notations can simply be boxes and lines diagram or any natural languages (e.g., English) through which practitioners can graphically or textually specify their design decisions and communicate them with others. Practitioners may also use the software modelling languages with concrete syntax and well-defined semantics, which enable the processing of the architectural models for purposes, e.g., analysis, simulation, and code-generation. Indeed, there are different types of languages, including the architectural languages (ALs) with architecture-oriented notation sets (i.e., components and connectors) [23], Business Process Modelling Langages (BPMLs) [39], domain-specific languages (DSLs) [7], formal specification languages (e.g., pi-calculus [26] and CSP [11]) with formally defined semantics, and Unified Modeling Language (UML) [35] and its extensions.

However, the current literature does not really aid in understanding practitioners' perspectives towards different architecture viewpoints. Indeed, many issues, such as the notations preferred by practitioners for different viewpoints, practitioners' expectations from modeling software architectures in different viewpoints, practitioners' main motivations of modeling in different viewpoints, still remain ambiguous. Therefore, in this study, a survey has been conducted among a set of practitioners from different industries to understand their knowledge and experience about the architectural viewpoints. The survey focuses on the five key architectural viewpoints that have been proposed by Taylor et al. [38] and are believed to be highly used by the practitioners who are involved in software development. These are the logical, physical, deployment, behaviour, and concurrency viewpoints. The logical viewpoint deals with the decompositions of software systems into software components and connectors. The physical viewpoint focuses on the physical components in which the logical components will be allocated and the physical communication links. The deployment viewpoint focuses on how the logical components should be mapped into the physical components. The behaviour viewpoint deals with the component and connector behaviours. The component behaviours can be considered either as the interface behaviours or the internal behaviours. The connector behaviours can be considered as the complex interaction mechanisms that control the interactions of the components. Lastly, the concurrency viewpoint deals with the concurrency-related issues of software components such as the synchronous/asynchronous communications, the parallel compositions, the use of threads and processes, and the common concurrency problems (e.g., deadlock and race-condition).

2 Survey Methodology

The survey consists of 38 different questions, which have been determined after several iterations of reviews. Among the 38 questions, just 9 questions use

single-choice answers. 2 of them provide numeric (i.e., integer) answers in which the participants are requested to choose one of the ranges of numbers (e.g., 0-10, 11-50, 51-100, and 100+). The rest of the 7 questions are essentially the yes/no questions. However, to maximise the precision of the survey results, those questions each provide a rating answer, which offers the following options: always (100%), much of the time (≥75%), often (≥50%), sometimes (<50%), and never (0%). The rest of the questions offer multiple-choice answers and allow practitioners to choose as many of the answers as they wish. Note that each multiple-choice answer also include a "free-text" area in which practitioners can type any answers that are not existing in the list of the multiple-choice answer.

The survey has been designed for the software industry and intended for the practitioners who hold software related positions such as software architect, designer, developer, software engineer, programmer, and software project manager. To reach as many practitioners as possible, the social media platforms have been tried initially, including facebook, linkedin, google and yahoo groups, and some popular software development forums (e.g., stack overflow). Also, the existing architectural languages that support the viewpoints considered in the survey have been searched and any scientists with industry background who contributed to the development of those languages have been contacted via e-mail. Moreover, the mailing lists of the widely-known computing societies (i.e., ACM and IEEE) have been used. Lastly, the author's personal contacts working in software industry have been requested to participate in the survey too. So, the survey received 56 different responses from practitioners all over the world.

3 Survey Results

In this part, the answers given to the survey questions are analysed. To facilitate understandability, the answers of the questions are considered in six different sections: one for the profile questions and a separate section for each viewpoint considered (i.e., logical, physical deployment, behaviour, and concurrency). For each section, the answers given to each relevant question are discussed separately.

3.1 Profile of the Practitioners

Practitioners country of work (Q1). The survey attracted practitioners from 20 different countries, given as follows: USA, Colombia, Austria, Latvia, Chile, Indonesia, Ukraine, Australia, Belgium, Finland, France, Germany, India, Portugal, Russia, Spain, Sweden, the Netherlands, Turkey, and UK. The top participating country is USA (22%), which is followed by Turkey (13%), UK (11%), and France (9%).

Practitioners' highest academic degrees and university subjects (Q2 and Q3). 76% of the practitioners hold postgraduate degrees, where 47% hold MSc or MA degrees and 29% hold PhD degrees. 22% of the practitioners hold undergraduate BSc or BS degrees. Lastly, 2% of the practitioners hold college degrees. Many of

the practitioners (63%) have studied the computer science/engineering subject. Concerning the rest of the practitioners, 9% of the practitioners have studied software engineering, 7% studied information technology, 6% studied electrical and electronics engineering, and the other subjects (e.g., physics, maths, history, and management) have been studied by 2-4%.

Practitioners' job positions (Q4). The software architect position is the top-selected one, held by 48% of the practitioners, which is followed by the software developer/programmer (29%) and consultant (29%) positions. The system engineer position (14%) and the high-level manager positions are held by 11-14% of the practitioners.

Practitioners' years of experience on the software architecture modelling (Q5). More than half of the practitioners (52%) have more than 10 years of experience. 21% of the practitioners have 6-10 years of experience, 13% of the practitioners just have 2-5 years of experience, and 9% have less than 2 years of experience. Note that 5% of the practitioners do not have any experience on the architecture modelling at all. These practitioners with no experience are directed to submit the survey form without answering the rest of the survey questions.

Practitioners' work industries (Q6). IT and telecommunications is the top industry that is selected by 47% of the practitioners. This is followed by the finance and accounting (27%), automative and transportation (22%), government (20%), defense/military aviation (16%) and healthcare and biomedical (12%) industries. 8% of the practitioners work in the software outsourcing industry.

Practitioners' software project types (Q7). The top software project type developed by practitioners is the business applications software (63%), which is followed by the web applications (57%) and mobile applications (51%). The systems software, scientific/engineering applications, and safety-critical and mission-critical software have been selected by 27-35% of the practitioners. The cloud applications and telecommunications software are the least selected software project types.

3.2 Logical Viewpoint

The practitioners who model software architectures from the logical viewpoint (Q8). Practitioners frequently model their software architectures from the logical viewpoint. Indeed, 36% of the practitioners always (100%) do so, 31% of the practitioners do so much of the time (>=75%), and 13% do so often (>=50%). Just 12% of the practitioners sometimes (<50%) model their software architectures from the logical viewpoint. 8% of the practitioners never (0%) model their software architectures from the logical viewpoint.

The structural units preferred for the logical components (Q9). As shown in Fig. 1, most of the practitioners (84%) consider components in terms of their external interfaces for sending/receiving services. Half of the practitioners also wish to specify composite components (i.e., the unit of configuration), which

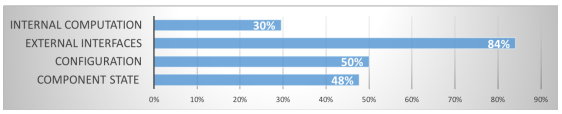

Fig. 1: The structural units preferred by practitioners for the logical components may structurally be composed of some other component and connector instances. Some practitioners (48%) wish to specify the component state descriptions for purposes such as the behaviour specifications (i.e., how the services impact on the state). Lastly, the internal computation unit for specifying the internal behaviours of components has been shown the least interest, selected by 30% of the practitioners.

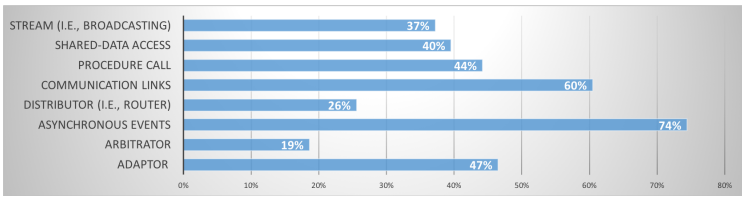

Fig. 2: The types of logical connectors that are preferred by practitioners

The types of logical connectors preferred by practitioners (Q10). In this survey, the connector types proposed by Mehta et al. [24] have been considered. As shown in Fig. 2, the top preferred connector types are the asynchronous event connectors (74%) and simple link connectors (60%). 47% of the practitioners wish to use adaptor connectors, which enable the successful composition of a set of interacting components that are incompatible and cannot communicate otherwise. The shared-data access, procedure call, and stream connectors are preferred by 37-44% of the practitioners. Arbitrators and distributors are rarely used by practitioners (19-26%), where the former represent the connectors for dealing with the quality properties (e.g., schedulability, performance, and security) and the latter represent the connectors for handling the distributed communication issues of distributed components.

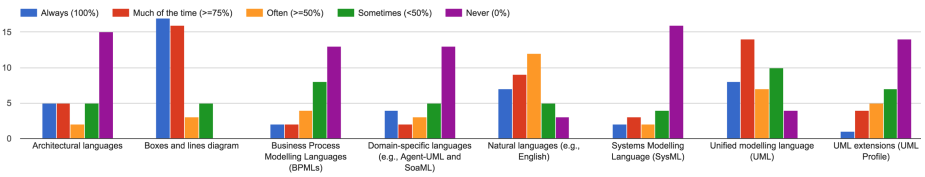

Fig. 3: The software modelling notations used by practitioners for modeling from the logical viewpoint

The software modelling notations used by practitioners for modeling software architectures from the logical viewpoint (Q11, Q12, Q13). As shown in Fig. 3,

the boxes and lines diagram is the top used modelling notation, which is followed by UML and natural languages (e.g., English). Many practitioners never use the BPMLs, DSLs, and UML extensions. Only a few practitioners stated that they use ALs for modeling their software architectures from the logical viewpoint. The ALs they prefer are AADL [9], Acme [10], and Archimate [1]. Besides, practitioners have also indicated to use some other software modelling notations, including pseudocode, formal specification languages (e.g., Z [36], CSP [11], and CCS [25]), Unified Architecture Framework (UAF) [28], and Structured Analysis and Design Technique (SADT) [20].

Practitioners' motivations for modeling software architectures from the logical viewpoint (Q14). Almost all the practitioners are motivated by the abilities of *(i)* documenting and communicating the component structures (91%) and *(ii)* decomposing a large and complex problem into manageable and understandable components (86%). Also, many practitioners (64%) are interested in *(i)* documenting and communicating the types of connectors for the component interactions and *(ii)* analysing the static aspects of systems (e.g., inconsistencies, incompatibilities, and incompleteness). Surprisingly, generating software code from the logical view specifications or the languages' distinguishing features (e.g., large-view management, graphical support, and extensibility) are not among the popular reasons, chosen by 25-30% of the practitioners.

3.3 Behaviour Viewpoint

The practitioners who model software architectures from the behaviour viewpoint (Q15). Half of the practitioners frequently (>=75%) model their software architectures from the behaviour viewpoint (19% chose *always* and 33% chose *much of the time*). 11% of the practitioners often (>=50%) do so and 25% of the practitioners sometimes (<50%) do so. 12% of the practitioners never software architectures from the behaviour viewpoint.

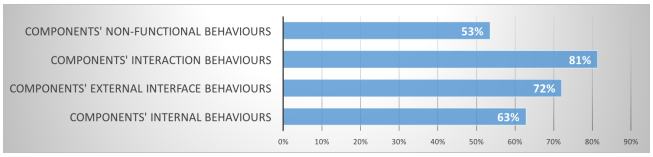

Fig. 4: The component behavioural units preferred by practitioners

The component behavioural units preferred by practitioners (Q16, Q17). As shown in Fig. 4, practitioners are most interested in specifying the interaction behaviours of components (81%) and their external interface behaviours (72%). 63% of the practitioners are interested in specifying the internal behaviours of components. Lastly, 53% of the practitioners are interested in considering the non-functional requirements (e.g., performance, security, and reliability) as part of their component behaviour specifications.

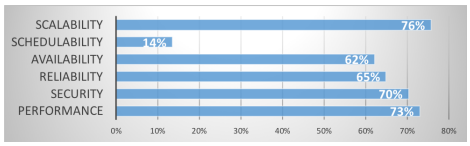

Fig. 5: The non-functional properties that practitioners are interested in

The non-functional properties that practitioners are interested in (Q18). As shown in Fig. 5, scalability, performance, and security are the top-preferred non-functional properties (70-76%), which are followed by the availability and reliability properties (62-65%). However, schedulability is preferred by just 14% of the practitioners.

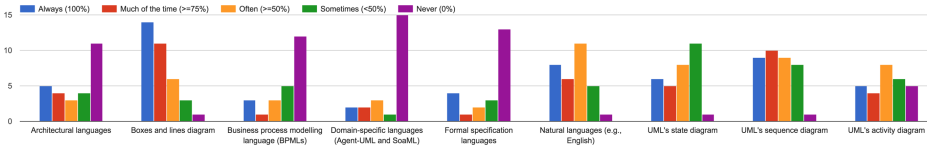

Fig. 6: The software modelling notations used by practitioners for modeling software architectures from the behaviour viewpoint

The software modelling notations used by practitioners for modeling software architectures from the behaviour viewpoint (Q19, Q20). As shown in Fig. 6, the boxes and lines diagram are the top-used notation for specifying the behaviour views of software architectures, which is followed by UML's sequence diagram, UML's state diagram, and the natural languages (e.g., English). Surprisingly, DSLs, BPMLs, formal specification languages, and ALs are the least-preferred modelling notations by practitioners in specifying the behaviour views.

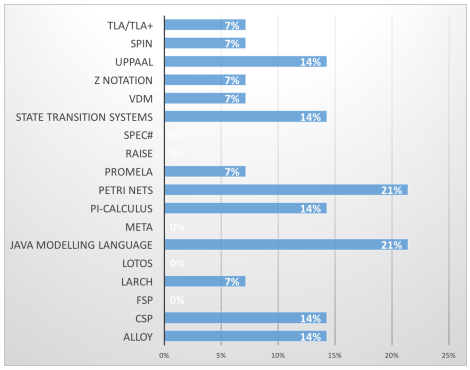

Fig. 7: The formal specification languages used by practitioners for formally modeling software architectures from the behaviour viewpoint

The formal specification languages used by practitioners for formally modeling software architectures from the behaviour viewpoint (Q21). While most of the practitioners never use formal specification languages as revealed in the previous questions, a few of those stated that they use some formal notations. These are Petri nets [27], Java Modelling Language (JML) [4], UPPAAL [16], state transition systems, pi-calculus [26], CSP [11], and Alloy [13].

The architectural languages used by practitioners for modeling software architectures from the behaviour viewpoint (Q22). Only a few number of practitioners stated that they frequently use the ALs for modeling the software architectures from the behaviour viewpoint. So, AADL is practitioners' top choice, followed by the Rapide [18] and Darwin [19] ALs.

Practitioners' motivations for modeling software architectures from the behaviour viewpoint (Q23). Documenting and communicating the high-level system behaviours has been selected by almost all the practitioners (90%). The capabilitiy of making the optimal design decisions about the component behaviours and interactions is also found motivating by more than half of the practitioners (55%). Some practitioners (30-40%) stated that they are motivated by the abilities of *(i)* specifying the non-functional properties for the component behaviours and interactions and *(ii)* utilising from the modelling languages' capabilities (e.g., precision, complex view management, extensibility, etc.). However, most practitioners do not see the generation of software code and the exhaustive (i.e., formal) analysis as the motivating reasons for the behaviour viewpoint modeling.

3.4 Concurrency Viewpoint

The practitioners who use the concurrency viewpoint (Q24) Most of the practitioners do not model their software architectures from the concurrency viewpoint. Indeed, while 38% sometimes (<50%) do so, another 38% never do so really. Note that just 8% of the practitioners always model software architectures from the concurrency viewpoint.

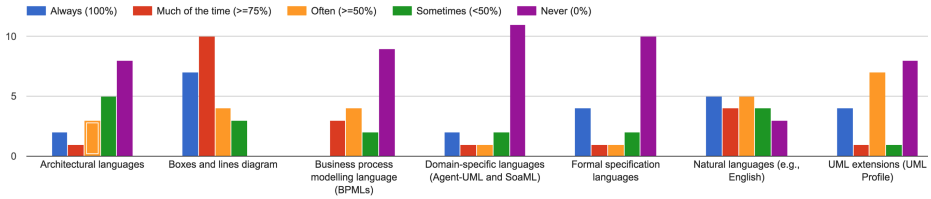

Fig. 8: The software modelling notations used by practitioners for modeling software architectures from the concurrency viewpoint

The software modelling notations used by practitioners for modeling software architectures from the concurrency viewpoint (Q25, Q26). As shown in Fig. 8, the boxes and lines diagram is the top preferred notation, followed by the natural

languages (e.g., English). A few practitioners use their own (in-house) DSLs for modeling software architectures from the concurrency viewpoint. ALs, BPMLs, DSLs, formal specification languages, and UML extensions are rarely used.

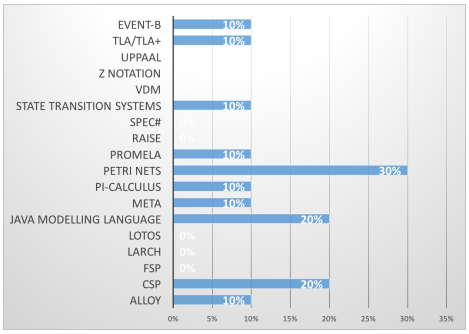

Fig. 9: The formal specification languages used by practitioners for formally modeling software architectures from the concurrency viewpoint

The formal specification languages used by practitioners for formally modeling software architectures from the concurrency viewpoint (Q27). As indicated in the previous question, the formal specification languages are rarely used for the concurrency viewpoint modeling, Fig. 9 shows the formal languages used by a few practitioners. So apparently, Petri nets, JML, and CSP that are shown in Fig. 9 are preferred relatively more than any other languages by the practitioners who use formal languages (20-30%). Some of those practitioners also stated that they use some other formal languages that include Event-B [2], state transition systems, ProMeLa [12], pi-calculus, and Alloy.

The architectural languages used by practitioners for formally modeling software architectures from the concurrency viewpoint (Q28) Like the formal specification languages, a few practitioners stated that they use ALs for modeling software architectures from the concurrency viewpoint. Among the four popular ALs considered for modeling concurrency (Darwin, Rapide, AADL, and Wright), Darwin seems to be the top popular AL among practitioners. Some participants stated that they use their own ALs that they developed (indicated as *other*).

The software modelling notation(s) used for mapping the logical components into the concurrency components (Q29). Most of the practitioners (85%) who model software architectures from the concurrency viewpoint are also interested in mapping their logical components to the concurrency components. Those practitioners mainly prefer the natural languages (e.g., English) and simple boxes and lines diagram. A few practitioners prefer the software modelling languages, such as the ALs, BPMLs, DSLs, and UML's extensions.

Practitioners' motivations for modeling software architectures from the concurrency viewpoint (Q30). Almost all the practitioners (81%) are motivated by the

ability for documenting and communicating the concurrency issues of their software systems. Many practitioners (59%) wish to specify the design decisions for avoiding the concurrency issues such as deadlock, livelock, and race-conditions. Some (30-37%) wish to meet the domain requirements of their concurrent systems. Surprisingly, only a few practitioners (22-26%) find the formal analysis of the concurrent specifications and the mapping between the logical and concurrent components as motivating.

3.5 Physical and Deployment Viewpoints

The practitioners who use the physical and deployment viewpoints (Q31). More than half of the practitioners (52%) frequently model their software architectures from the physical and deployment viewpoints (i.e., *always* or *much of the time*). While 16% of the practitioners often (>=50) do so, 22% of the practitioners sometimes (<50) do so. Lastly, 10% of the practitioners never model software architectures from the physical and deployment viewpoints.

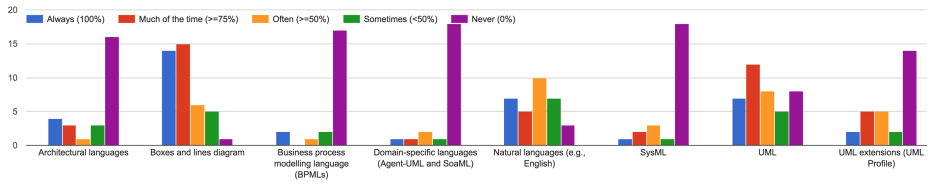

Fig. 10: The software modelling notations used by practitioners for modeling software architectures from the physical and deployment viewpoints

The software modelling notations used by practitioners for modeling software architectures from the physical and deployment viewpoints (Q32, Q33). As shown in Fig. 10, the boxes and lines diagram is the top-preferred notation, which is followed by the natural languages (e.g., English) and UML. The ALs, BPMLs, and DSLs, SysML are never used by most of the practitioners for the physical and deployment viewpoint modeling. Lastly, some of the practitioners stated that they use their own DSLs, and some use the UAF standard [28].

The architectural languages used by practitioners for modeling software architectures from the physical and deployment viewpoints (Q34). AADL is the top-used AL, which is followed by the Archimate language. Abacus [8], Meta-H [22], and East-ADL [6] are the other languages used by practitioners.

The non-functional quality properties that practitioners consider for their deployment viewpoint modeling (Q35). According to the survey results, 84% of the practitioners who model software architectures from the physical & deployment viewpoints also consider documenting the non-functional quality properties that can then be used for analysing the deployment architectures. Among the four non-functional properties considered (i.e., availability, performance, scalability,

and security), the availability (e.g., load balancing, redundancy, and failure situations) and scalability (e.g., the resource capacity for larger systems) properties are shown the greatest interest (79%). 67-69% of the practitioners consider the security (e.g., authentication and secure data transmission) and performance (e.g., the number of CPU and memory estimates) properties. Note that a few practitioners have chosen some other quality properties, i.e., maintainability, developer & operations cooperation, and regulatory compliance.

The software modelling notation(s) that practitioners use for mapping the logical components into the physical components (Q36). The survey results reveal that 80% of the practitioners who model their software architectures from the physical & deployment viewpoints wish to map the logical components into the physical components. The simple boxes and lines diagram (53%) and natural languages (50%) are the top-popular techniques, followed by UML (40%). The rest of the software modelling notations (i.e., ALs, UML's extensions, SysML, BPML, and DSLs) are shown a lack of interest, which are used by 10-15% of practitioners.

The software modelling notation(s) that practitioners use for mapping the concurrent components into the physical components (Q37). According to the survey results, 60% of the practitioners who specify the physical & deployment views are interesting in mapping the concurrent components into the physical components. Practitioners prefer the simple boxes and lines diagram (42%) and natural languages (45%), and UML (34%), as is the case for the mapping between the logical and physical components discussed in Section 3.5.

Practitioners' motivations for modeling software architectures from the physical and deployment viewpoints (Q38). According to the results, most of the practitioners (86%) are motivated by documenting and communicating their software systems' physical components and their physical communications motivates them. 68% of the practitioners are motivated by the ability of analysing the deployment view specifications for quality properties. More than half of the practitioners (54%) are interested in mapping between the logical/concurrency views and physical views. Lastly, the modelling languages with useful capabilities (e.g., graphical support, formal analysis, tool support, etc.) and generating executable software code are shown a lack of interest by practitioners (14-19%).

4 Related Work

The author has previously conducted some similar surveys on practitioners. However, none of them considered the architectural viewpoints from practitioners' perspectives. In [29], the author aimed at learning practitioners' understanding of software architectures. In [31], the author aimed at learning the informal and formal software modeling notations used by practitioners for the specifications of software architectures and practitioners' expectations and motivations for the modeling notations. Likewise, in the rest of this section, some other similar surveys have been discussed, which are not so useful in understanding practitioners' knowledge and experience on different viewpoints.

May [21], for instance, analysed and compared five influential viewpoint models (e.g., Kruchten's 4+1 model [15]) using a comparison framework that May proposed. In his framework, May analysed the viewpoints supported by the viewpoint models regarding their considerations for the architectural structures (e.g., layered, client-server, and abstraction), the types of the stakeholders, and the functional/non-functional concerns. Smolander et al. [14] surveyed among three different companies that work for the telecommunications industry, where one is a system integrator, one is a mobile software company, and the other is a telecommunication service provider. Smolander et al. analysed these three companies so as to determine the viewpoints that are used by these organisations in their software architecture designs. Tang et al. [37] focussed on a number of viewpoint models (including Kruchten's) and analysed their support for the basic requirements of architecture design including the architecture definition, analysis, evolution, the support for specifying design decisions, and the use of repositories. Booch et al. [3] surveyed among a number of enterprise architecture and technical architecture viewpoint models that promote the graphical specifications of the enterprise software architectures. Booch et al. grouped the viewpoint models based on their domain (i.e., company-specific, government-specific, defense-specific, and consulting) and analysed them to learn which viewpoints are supported, the accessibility of any resources for the viewpoint model, and their weakness and strengths relatively to other viewpoint models. Purhonen et al. [33] analysed a set of architecture viewpoints (i.e., structure, behaviour, deployment, and development) regarding their support for the digital signal processing and middleware software architectures. Lastly, Lassing et al. [17] consider four viewpoints, grouped as macro architecture level (i.e., the system level) and micro level (i.e., the internal structure of the system). The macro viewpoints are the context and technical infrastructure viewpoints, while the micro viewpoints are the conceptual and development viewpoints. Lassing et al. analysed these viewpoints for the modifiability of software architectures (i.e., how easily the system functionality can be changed).

5 Discussions and Conclusions

In this study, a survey has been conducted on understanding practitioners' knowledge and experience about the logical, behaviour, concurrency, physical and deployment viewpoints. While the logical, behaviour, physical, and deployment viewpoints are used by many practitioners, the concurrency viewpoint for the specifications of the concurrency issues are shown a lack of interest by practitioners. For each viewpoint considered, practitioners stated that documenting the design decisions and their communications with other stakeholders are their main source of motivation. Also, practitioners prefer the boxes and lines diagram and natural languages (e.g., English) for each viewpoint considered. However, these notations do not offer any concrete syntax and semantics; so, the specifications may not be processed for purposes such as analysis and code generation.

Concerning the logical viewpoint, most practitioners view the logical components structurally as the composition of external interfaces. Practitioners are not so keen to specify the internal computations that are mainly concerned with any internal actions operated on the component state. Practitioners are highly interested in the event connectors, which may receive the asynchronous events generated by a component and send it to all listening components. Complex connectors such as arbitrators and distributors are rarely considered by practitioners. This may be due to the lack of modeling languages that allow for specifying complex connectors. Indeed, the event connectors are highly popular among the architecture modeling languages [30]. Concerning the behaviour viewpoint, most practitioners are interested in specifying the interaction behaviours of the components (i.e., how the component state changes depending on the interface operations performed). Practitioners use UML's state and sequence diagrams for the behaviour specifications. However, other important behaviour modeling notations such as formal languages and some domain-specific languages that could aid in the formal verification of the system behaviours are rarely used. This is probably due to that those languages are found difficult to learn and use compared with UML. Given practitioners' reluctance towards the concurrency viewpoint, practitioners main concerns have been observed to be the software modelling notations that lack in the support for the concurrency issues (e.g., synchronous/asynchronous communications, threads, and processes) and the mapping between the logical and concurrent components. Even if the languages support concurrency, they offer this via a process algebras, which are found as difficult to learn and use. Lastly, concerning the physical and deployment viewpoints, while many practitioners are interested in specifying the scalability and availability of the deployment architectures, there is a strong concern about the languages' tool support for modeling and analysing those properties.

In the future, the survey results are aimed to be used for the proposal of a novel architecture description language that supports the logical, behaviour, concurrency, physical, and deployment viewpoints in a way which addresses the needs of practitioners determined from the survey.

References

1. The Open Group ArchiMate® 1.0 Specification. Technical Standard (Feb 2009)
2. Abrial, J.R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press, New York, NY, USA, 1st edn. (2010)
3. Booch, G., Mitra, T.: A survey of enterprise view models. Tech. Rep. RC25049, IBM Research Report (2010)
4. Chalin, P., Kiniry, J.R., Leavens, G.T., Poll, E.: Beyond assertions: Advanced specification and verification with JML and ESC/Java2. In: de Boer, F.S., Bonsangue, M.M., Graf, S., de Roever, W.P. (eds.) FMCO. Lecture Notes in Computer Science, vol. 4111, pp. 342–363. Springer (2005)
5. Clements, P.C., Garlan, D., Little, R., Nord, R.L., Stafford, J.A.: Documenting software architectures: Views and beyond. In: Clarke, L.A., Dillon, L., Tichy, W.F. (eds.) ICSE. pp. 740–741. IEEE Computer Society (2003)

6. Cuenot, P., Frey, P., Johansson, R., Lönn, H., Reiser, M.O., Servat, D., Tavakoli Kolagari, R., Chen, D.: Developing automotive products using the east-adl2 : an autosar compliant architecture description language. Ingénieurs de l'Automobile (Automobile Engineers) **:793** (2008)
7. van Deursen, A., Klint, P., Visser, J.: Domain-specific languages: An annotated bibliography. SIGPLAN Not. **35**(6), 26–36 (Jun 2000). https://doi.org/10.1145/352029.352035, `http://doi.acm.org/10.1145/352029.352035`
8. Dunsire, K., O'Neill, T., Denford, M., Leaney, J.: The abacus architectural approach to computer-based system and enterprise evolution. In: 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'05). pp. 62–69 (April 2005). https://doi.org/10.1109/ECBS.2005.66
9. Feiler, P.H., Lewis, B.A., Vestal, S.: The SAE architecture analysis & design language (AADL): A standard for engineering performance critical systems. In: IEEE Intl Symp. on Intell. Control. pp. 1206–1211 (Oct 2006)
10. Garlan, D., Monroe, R.T., Wile, D.: Acme: An architecture description interchange language. In: Proceedings of CASCON'97. pp. 169–183. Toronto, Ontario (November 1997)
11. Hoare, C.A.R.: Communicating sequential processes. Commun. ACM **21**(8), 666–677 (1978)
12. Holzmann, G.J.: The SPIN Model Checker - primer and reference manual. Addison-Wesley (2004)
13. Jackson, D.: Alloy: a lightweight object modelling notation. ACM Trans. Softw. Eng. Methodol. **11**(2), 256–290 (2002)
14. Kari Smolander, Kimmo Hoikka, J.I.M.K., Mkel, T.: What is included in software architecture? a case study in three software organizations. In: Proceedings Ninth Annual IEEE International Conference and Workshop on the Engineering of Computer-Based Systems. pp. 131–138 (2002). https://doi.org/10.1109/ECBS.2002.999831
15. Kruchten, P.: The 4+1 view model of architecture. IEEE Software **12**(6), 42–50 (1995). https://doi.org/10.1109/52.469759, `http://dx.doi.org/10.1109/52.469759`
16. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a nutshell. STTT **1**(1–2), 134–152 (1997)
17. Lassing, N.H., Rijsenbrij, D.B.B., van Vliet, J.C.: Viewpoints on modifiability. International Journal of Software Engineering and Knowledge Engineering **11**(4), 453–478 (2001)
18. Luckham, D.C., Kenney, J., Augustin, L., Verra, J., Bryan, D., Mann, W.: Specification and Analysis of System Architecture Using Rapide **21**(4), 336–355 (Apr 1995)
19. Magee, J., Kramer, J.: Dynamic structure in software architectures. ACM SIGSOFT Software Engineering Notes **21**(6), 3–14 (nov 1996)
20. Marca, D.A., McGowan, C.L.: SADT: Structured Analysis and Design Technique. McGraw-Hill, Inc., New York, NY, USA (1987)
21. May, N.: A survey of software architecture viewpoint models. In: Proceedings of the Sixth Australasian Workshop on Software and System Architectures. pp. 13–24 (May 2005)
22. McDuffie, J.H.: Using the architecture description language metah for designing and prototyping an embedded reconfigurable sliding mode flight controller. In: Proceedings of the 21st Digital Avionics Systems Conference. vol. 2, pp. 8B1–1–8B1–17 vol.2 (2002). https://doi.org/10.1109/DASC.2002.1052937

23. Medvidovic, N., Taylor, R.N.: A classification and comparison framework for software architecture description languages. IEEE Trans. Software Eng. **26**(1), 70–93 (2000)
24. Mehta, N.R., Medvidovic, N., Phadke, S.: Towards a taxonomy of software connectors. In: Proceedings of the 22Nd International Conference on Software Engineering. pp. 178–187. ICSE '00, ACM, New York, NY, USA (2000)
25. Milner, R.: A Calculus of Communicating Systems. Springer-Verlag New York, Inc., Secaucus, NJ, USA (1982)
26. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, i. Inf. Comput. **100**(1), 1–40 (1992)
27. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE **77**(4), 541–580 (Apr 1989). https://doi.org/10.1109/5.24143
28. OMG: Unified architecture framework profile (uafp). Specification dtc/17-05-08, OMG (nov 2012), http://www.omg.org/spec/UAF/1.0/Beta2/PDF
29. Ozkaya, M.: What is software architecture to practitioners: A survey. In: Hammoudi, S., Pires, L.F., Selic, B., Desfray, P. (eds.) MODELSWARD 2016 - Proceedings of the 4rd International Conference on Model-Driven Engineering and Software Development, Rome, Italy, 19-21 February, 2016. pp. 677–686. SciTePress (2016). https://doi.org/10.5220/0005826006770686, http://dx.doi.org/10.5220/0005826006770686
30. Ozkaya, M.: Architectural languages' connector support for modeling various component interactions: A review. In: Fujita, H., Herrera-Viedma, E. (eds.) New Trends in Intelligent Software Methodologies, Tools and Techniques - Proceedings of the 17th International Conference SoMeT_18, Granada, Spain, 26-28 September 2018. Frontiers in Artificial Intelligence and Applications, vol. 303, pp. 474–489. IOS Press (2018). https://doi.org/10.3233/978-1-61499-900-3-474, https://doi.org/10.3233/978-1-61499-900-3-474
31. Ozkaya, M.: Do the informal & formal software modeling notations satisfy practitioners for software architecture modeling? Information & Software Technology **95**, 15–33 (2018). https://doi.org/10.1016/j.infsof.2017.10.008, https://doi.org/10.1016/j.infsof.2017.10.008
32. Perry, D.E., Wolf, A.L.: Foundations for the study of software architecture. SIGSOFT Softw. Eng. Notes **17**(4), 40–52 (Oct 1992). https://doi.org/10.1145/141874.141884, http://doi.acm.org/10.1145/141874.141884
33. Purhonen, A., Niemelä, E., Matinlassi, M.: Viewpoints of dsp software and service architectures. J. Syst. Softw. **69**(1-2), 57–73 (Jan 2004)
34. Rozanski, N., Woods, E.: Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives. Addison-Wesley Professional, 2 edn. (2011)
35. Rumbaugh, J., Jacobson, I., Booch, G.: Unified Modeling Language Reference Manual, The (2Nd Edition). Pearson Higher Education (2004)
36. Spivey, J.M.: Z Notation - a reference manual (2. ed.). Prentice Hall International Series in Computer Science, Prentice Hall (1992)
37. Tang, A., Han, J., Chen, P.: A comparative analysis of architecture frameworks. In: 11th Asia-Pacific Software Engineering Conference (APSEC 2004), 30 November - 3 December 2004, Busan, Korea. pp. 640–647. IEEE Computer Society (2004)
38. Taylor, R.N., Medvidovic, N., Dashofy, E.M.: Software Architecture - Foundations, Theory, and Practice. Wiley (2010)
39. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2007)