What Industry needs from Architectural Languages: A Survey

Ivano Malavolta, Patricia Lago, Henry Muccini, Patrizio Pelliccione, Antony Tang

Abstract—Many times we are faced with the proliferation of definitions, concepts, languages and tools in certain (research) topics. But often there is a gap between what is provided by existing technologies, and what is needed by their users. The strengths, limitations and needs of the available technologies can be dubious.

The same applies to software architectures, and specifically to languages designed to represent architectural models. Tens of different architectural languages have been introduced by the research and industrial communities in the last two decades. However, it is unclear if they fulfill the user's perceived needs in architectural description. As a way to plan for next generation languages for architectural description, this study analyzes practitioners' perceived strengths, limitations and needs associated to existing languages for software architecture modeling in industry. We run a survey by interviewing 48 practitioners from 40 different IT companies in 15 countries. Each participant is asked to fill in a questionnaire of 51 questions. By analyzing the data collected through this study, we have concluded that (a) whilst practitioners are generally satisfied with the design capabilities provided by the languages they use, they are dissatisfied with the architectural language analysis features and their abilities to define extra-functional properties; (b) architectural languages used in practice mostly originate from industrial development instead of from academic research; (c) more formality and better usability are required of an architectural language.

Index Terms—Software Architecture, Architecture Description Languages, ADL, Survey, Empirical Study

1 Introduction

1.1 The Problem

In their seminal paper on software architecture [1], Dewayne Perry and Alexander Wolf foresee the 90s as the decade of software architecture, and justify the need of a (software) architecting discipline on the conjecture that the slow progress in the development of software systems is due to the excessive focus on development and the limited focus on architecting. Twenty years later, we recognize the impact that software architecture has been having on both academic and industrial worlds. Software architectures are nowadays used for different purposes, including documenting and communicating design decisions and architectural solutions [2], driving analysis techniques (like testing, model and consistency checking, performance analysis [3], [4], [5], [6]), for code generation purposes in model-driven engineering [7], [8], for product line engineering [9], for risks and cost estimation [10], [11], and many more.

One principal issue is the proliferation of languages for software architectures (SA) description without a clear understanding of their merits and limitations. Tens of architectural languages (ALs)[1] can be found today, each characterized by slightly different conceptual architectural elements, different syntax or semantics. They focus on a generic or a specific operational domain; some do not provide design analysis whilst others support different analysis techniques. As observed in [12], one of the reasons for the accumulation of so many architectural languages is the need to satisfy different *stakeholder concerns*: a language has to adequately capture design decisions judged fundamental by the system's stakeholders. Stakeholder concerns are various, ever evolving and adapting to changing system requirements. Hence it is difficult to capture all such concerns with a single, narrowly focused notation. Instead of defining a unique language for software architecture specification, we must accept the existence of domain specific languages for SA modeling, allowing an AL to address specific types of stakeholder concerns. The adoption of UML for modeling architectures (e.g., [13], [14]) did not converge into a standard and uniquely identified notation for SA modeling: a number of UML profiles and extensions have been proposed to enhance the modeling for different concerns, thus further proliferates new architectural languages.

In summary, it is clear that an ideal and general purpose AL is not likely to exist [15], [14], [16], [12]. Rather, architectural languages must be able to focus on **"what is needed"** by the stakeholders involved in the architecting process. Whilst much research has been conducted on ALs, there is a lack of understanding

- *I. Malavolta, H. Muccini, and P. Pelliccione are with the Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica, Università dell'Aquila, Italy. E-mail: [ivano.malavolta,henry.muccini,patrizio.pelliccione]@univaq.it*
- *P. Lago is with the Department of Computer Science, VU University Amsterdam, the Netherlands. E-mail: p.lago@vu.nl*
- *A. Tang is with the Swinburne University of Technology, Australia. E-mail: ATang@swin.edu.au*

1. Hereafter, we use the term architectural language, or AL, to refer to any form of expression used for architecture description. We use the AL term for the sake of clarity, since in the last decades several different definitions of the ADL term have been proposed.

Digital Object Indentifier 10.1109/TSE.2012.74

on practitioners' architectural description needs, and on what they consider to be useful, useless or missing in current ALs.

1.2 Related Studies

A few studies have analyzed **practitioners' needs** and the usefulness/uselessness of ALs as perceived in industrial practice.

Medvidovic et al. [12] identified three main *needs* (called lampposts) required to provide a more complete treatment of and a broader perspective on ALs. They are the *technology*, *domain*, and *business* needs. The technology needs focus on concerns surrounding recurring technical challenges of engineering software systems; the domain lamppost illuminates concerns driven and informed by the knowledge of a specific application domain; the business lamppost focuses on capturing and exploiting knowledge of the business context of a given development effort. In the WICSA 2005 working session report on architecture description languages in practice [17], Woods and Hilliard identified five *deficiencies* that hinder the widespread adoption of contemporary ALs, and specifically:

- ALs often make restrictive assumptions (like imposing a particular architectural model); those assumptions might be inappropriate,
- ALs are single view oriented, while a multi-view approach is required,
- few ALs are supported by satisfactory tools,
- most ALs are general purpose and so lack domain specific entities important to architects,
- ALs are not adopted in industrial settings. Overall, the mismatch between the way architects work and the support provided by ALs is recognized as the source of most of those problems.

A similar study, specific to the information systems domain, has been conducted in [18], in order to identify the *needs* that an architect would have with respect to ALs. They are:

- better support for multiple views, including functional, information, concurrency, and deployment,
- direct support for describing domain-specific concerns,
- satisfactory tool support,
- support for incremental adoption,
- support for reusing architectural models.

In [19] Pandey attempted to identify what went wrong with the ALs that did not become popular beyond their place of origin. He identified a set of *strengths* and *weaknesses* of current ALs (including UML). Architectural languages *lacks* are also discussed in [20]. This work identified six main *lacks*, that are:

- lack of adequate support for multiple viewpoints,
- lack of features for documenting decisions, commitments, obligations, and freedoms,
- lack of adequate support to architectural constraints,
- imposed implicit model of computation,
- lack of quantification mechanisms,
- lack of adequate to support architectural styles and patterns.

An analysis of AL features is proposed in [21], where Clements defined three categories of features (system-oriented, language-oriented, and process-oriented) and interviewed the proposers of eight ALs in order to collect information about the AL applicability, scope, support for variability, validation, analysis, and extensibility.

While the studies reported above have revealed interesting discussion points, the findings are mostly based on authors' subjective interpretations and personal experience. They do not have a broad base on the practitioners' understanding of what is useful or not, and are outdated.

This paper aims at filling this gap. We interviewed practitioners in the IT domain, and collected qualitative data through a questionnaire. The collected data shows the *practitioners' needs*, their degree of *satisfaction* and *dissatisfaction* with existing ALs, the *limitations* they envision on their day-to-day work, and what is perceived to be *useful* or *useless*. By presenting and discussing these results we are the first to provide an overview of the industry needs for researching and developing the next generation of ALs.

1.3 This Study

To the best of our knowledge, this paper presents the first investigation into industrial use of ALs. Our **goal** is to better understand practitioners' perceived strengths, limitations and needs associated to using ALs for software architecture modeling in the industry. The results of this study will (hopefully) lead to the development of new theories and languages for next generation ALs.

In this research, we directly contacted industrial experts for an interview. We selected participants from industries who may have used different types of ALs in production, including formal, semi-formal and informal notations. We also asked selected participants to nominate additional experts in their network (following a "snowball sampling" approach [22]). Eventually, 48 practitioners from 40 different IT companies in 15 countries participated to the survey. Specifically, 23 participants filled in a questionnaire, and 25 participants were interviewed based on the same questionnaire. The questionnaire was made of 51 questions, mostly with open answers. It was designed with two research questions in mind:

- RQ1: What are the architectural description needs of practitioners?
- RQ2: What features typically supported by existing ALs are useful (or not useful) for the software industry?

RQ1 explores the needs practitioners have in describing architectures in general. RQ2 investigates a list of features that are systematically extracted from existing

Findings and Results	Implications
F1) All top used ALs have been originated in industry	**I1)** ALs resulting from academic research seem not to have directly reached industry. This suggests that academic ALs do not fulfill industrial needs, even if they might have inspired the *industrial* ALs in some ways. It points to the necessity of understanding industrial requirements. Researchers who aim to develop an AL for practical use should understand such requirements.
F2) ALs should combine features supporting both communication and disciplined development. We call this *introvert versus extrovert nature of architects role*	**I2)** Major academic research efforts have been dedicated so far to formal and domain-specific ALs, while industry turns to informal ALs for communication purposes (considered as a primary need in practice). This finding emphasizes the need to further improve ALs in this direction: ALs need to be simple and intuitive enough to communicate the right message to the stakeholders involved in the architecting phase, but they shall also enable formality so to drive analysis and other automatic tasks.
F3) Organizations (even in domains involving critical systems) prefer semi-formal and generic ALs than formal- and domain-specific ones like ADLs	**I3)** This result might reveal a misconception that ADLs are crucial in specific domains like safety-critical systems. If such ALs do not effectively support communication among stakeholders in a simple way, such ALs are neglected and replaced by less formal ones.
F4) Extensibility is needed	**I4)** Practitioners need better support for extending a language and extensions are seldom reused across projects. This draws open research directions in modular and extensible ALs.
F5) Tool support is fine but should provide features to support collaboration and more flexible architecting.	**I5)** Practitioners, mostly, do not require technical features for verification, code generation or the like. They do require generic features, integrating configuration management or supporting collaboration, informal free-hand sketching, combining text and graphics. Research is needed to investigate these features that better help architects and developers in their everyday job, in their brainstorming meetings and collaborating in a team.
F6) Heavyweight and complex ALs often deter practitioners	**I6)** A good combination of features fulfilling practitioners' needs is crucial for adoption, and closing the gap between industry and academia.
F7) Selecting an AL is not driven by specific system/-domain characteristics, but depends on the presence of competencies, tool- and community support	**I7)** This result partly discredits a belief commonly shared in the research community, namely that an AL is selected based on the system to be realized or its application domain.
F8) Most important requirements of an AL are firstly design, followed by communication (also said to be the most unsatisfactory) followed by analysis. Code generation is not often required. Link to requirements (elicitation and specification) is important as well	**I8)** These results help prioritize research in ALs.
F9) Practitioners ranked the features provided by the (existing) ALs that they use (both past and future projects)	**I9)** The ranking (given in Table 11) provides an overview of the levels of satisfaction the practitioners have with their ALs. This is a further source for reflection.

TABLE 1
Main findings and implications of our study

ALs. We designed a questionnaire and associated interview guide, including a set of open questions, to address RQ1. We also asked participants to rank a list of features to address RQ2. In designing the study we pay special attention in selecting interviewees that represent small, middle, and large organizations and involved in software architecture description.

Our analysis produced two overall findings that are summarized in the first two rows of Table 1, and a number of more detailed results associated with clusters of questions; they are summarized in the remainder of Table 1. Those findings bring two general messages for future research in-the-field: they are the implications I1 and I2 that are summarized in the first two rows on the column *Implications* of Table 1.

It is important to note that this study depicts the current adoption of ALs by practitioners. The study does not specifically outline the *influences* that past research had on the developments of ALs. For example, we note that AL research of the 90's influenced the definition of UML 2.0, which includes notions of architectural components, connectors, ports, substructure, port bindings, etc. that were first introduced in early ALs such as Acme [23]. Similarly, AADL [24] was influenced by both Meta-H [25] and Acme [23]. This should be taken into account while reading the outcomes of the survey.

A comprehensive view of the ALs *adoption* requires to consider the entire adoption curve for ALs. Redwine and Riddle [26] proposed a model that describes the characteristics of maturing software technologies. They reviewed a number of software technologies, they found that it typically takes 15-20 years for a technology to be ready for popularization, and they identified six phases: (a) Basic research; (b) Concept formulation; (c) Development and extension; (d) Internal Enhancement and exploration; (e) External Enhancement and exploration; (f) Popularization.

Mary Shaw [27] elaborated the model proposed by Redwine and Riddle [26] with the aim to assess the progress of the software architecture area toward maturity. This work, published in 2001, identifies software architectures into the phase of *Development and extension*

and that *Enhancement and exploration* are beginning in earnest. Today we are still into the *Enhancement and exploration* phases and the phase of *Popularization* is beginning.

This survey reflects the point of view of practitioners that could help the adoption of ALs. The outcomes of our survey challenge recurring and well established beliefs of what architectural description tools should provide. At the same time, our findings provide inspiration and reflection for future research in architectural languages to better match the demands from industry. As such, outcomes can be seen as potential guidelines for envisioning the next generation of ALs to be used to better support architects in their everyday job.

1.4 Paper Outline

The paper is organized as follows. After discussing related work and clarifying our terminology around the notion of architectural languages (in Section 2), Section 3 explains our research methodology in designing the survey, followed by the description of the survey participants and survey outcomes (in Sections 4 and 5, respectively), then discussed in Section 6. Limitations and threats to validity are discussed in Section 7, while Section 8 concludes this paper.

2 Architectural Languages

Many languages have been proposed for specifying and analyzing software architectures (e.g., [16], [28], [29]). They have been referred to as architecture description languages, as canonical architectural languages, as architecture definition languages, and so on. These terms, however, have different meanings in different contexts. In order to avoid terminology misinterpretation, throughout this paper we use the term Architectural Language (or AL) to refer to *any form of expression used for architecture description, being it formal, informal, box-and-line, model-based*. We instead use the term Architecture Description Language (or ADL) to refer to a subset of ALs, and specifically to all those languages explicitly referred to as ADLs by their inventors (without entering on the merit of such a terminology). We refer to industry ad-hoc ALs to mean all those ALs that have been originated by industries and standardization bodies in the context of a specific project. We refer to UML and UML-based notations to identify UML and the UML profiles defined to model software architectures.

Classically, ALs have been classified into three broad categories: box-and-line informal drawings, formal architecture description language, and UML-based languages. Box-and-line have been for a long time the only means for describing SAs. While providing useful documentation, the level of informality limited the usefulness of the architecture description[30], [1]. Basically, what was needed at that time was a more rigorous way for describing SAs.

As a result, a thread of research on formal ALs has been carried out, leading to the definition of tens of different ALs. Several formal ALs have been proposed, each characterized by different conceptual architectural elements, different syntax or semantics, focusing on a specific operational domain, or only suitable for different analysis techniques (e.g., [24], [31], [32], [33]). However, these efforts have not seen the desired adoption by industrial practice. Some reasons for this lack of industry adoption have been analyzed in [17], [18], [19], [20], [21]: formal ALs have been rarely integrated in the software life-cycle, they are seldom supported by mature tools, scarcely documented, focusing on very specific needs, and leaving no space for extensions enabling the addition of new features.

As a way to overcome some of those limitations, UML has been indicated as a possible successor of existing ADLs. Many proposals have been presented to use and extend UML 1.x to model software architectures (e.g., [34], [13], [14], [35], [36], [37], [38], [39]), and then many others for extending UML 2.x. While UML 2.x has introduced many new concepts to make it more suitable for architectural description, still many extensions have been proposed to cater for specific concerns [40], [41], [42], [43].

ALs have been also historically classified into two generations [12]. A "first generation" going from 1990 to 2000 had the main purpose to design an ideal AL [44] to support components and connectors specification and their overall interconnection [1], [44], as well as composition, abstraction, reusability, configuration, heterogeneity, and analysis [45]. In the "second generation" of ALs, going from 2000 up to now, new requirements emerged to deal with the increasing complexity of modern systems [46], [29]. They provide modeling support for *configuration management*, *distribution*, and *product lines*. Structural specifications have been integrated with behavioral ones [47], [48] with the introduction of many formalisms such as pre- and post-conditions, process algebras, statecharts, POSets, CSP, π-calculus, and others [16].

3 Research Methodology

The main purpose of this study is to explore the language features that are needed by practitioners to describe software architectures.

This study is organized in four phases. In Phase 1 we identify existing ALs for finding practitioners to be interviewed. We want to reach those industrial software architects who have been exposed to architecture description languages. In Phase 2 we identify the target population we want to involve in our survey, i.e., the communities that can provide the information we seek in our study. In Phase 3 we design and carry out the survey to elicit information from the participants. In Phase 4 we analyze the data we obtained from the survey.

3.1 Phase 1 - ALs Identification

The first phase of our study is to define the set of ALs we intend to study. The two steps performed to select ALs were:

Definition of a preliminary set of ALs. In this step, we talked to software architecture researchers and experts and searched papers published in software architecture conferences, journals, and events. As a result of this phase we identified 57 ALs[2].

Systematic search. In this step, we defined *Google Scholar*[3] and *Microsoft Academic Search*[4] queries to augment the list of notations in step (1). Google Scholar and Microsoft Academic Search search both the Internet and scientific digital libraries. We explicitly included non-scientific publications to identify industry ALs. Listing 1 shows an example of Microsoft Academic Search query. We searched for documents published after 1991 because it is well-known that the concept of ADL did not exist before that year. This query performed on the *Computer Science* domain returned a list of 102 references[5].

```
1 ADL architecture description language year>=1991
```

Listing 1. Example of Microsoft Academic Search query

The query used in Google Scholar produced 10,800 hits. Listing 2 shows a refinement of the query that produced a list of 2,730 hits[6].

```
2 ADL "architecture description language"
```

Listing 2. Example of Google Scholar query

In this search process we discovered that another community uses the term ADL to refer to languages for specifying the architecture for retargetable compilation [49]. As this community is clearly out of our scope, we refined the search query excluding results that contain names of approaches that fall in this other domain, such as *EXPRESSION ADL*, *Valen C*, etc. The query is shown in Listing 3. The execution of the query resulted in 1,930 hits.

```
3 ADL "architecture description language" -"EXPRESSION ADL" -"
    ↪Valen C" -MIMOLA -nML -ARC -Axys -RADL -Tensilica -
    ↪MDES -TDL -Flexware -HMDES -PRMDL -Maril -CSDL
```

Listing 3. Refined Google Scholar query

The list of 57 ALs produced in step (1) is enriched by the 120 ALs found in step (2) [7]. This output is used in Phase 2 to build the community of practitioners target of our survey.

3.2 Phase 2 - Planning the Survey

The community of practitioners is built by following a process composed of three steps as shown in Figure 1.

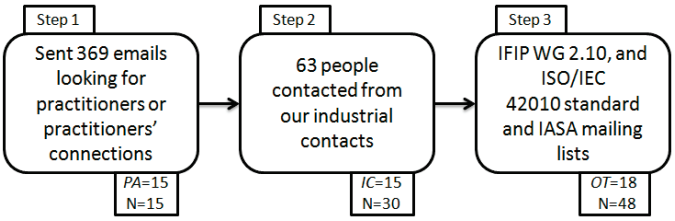

Fig. 1. Planning the survey: selection of the population

Step 1: we extracted the names of the authors from the scientific papers, white papers, and technical reports that had been identified in Phase 1. We contacted these authors by email with the aims to (a) extract useful information about the ALs and (b) to build a community of practitioners.

First of all we removed from the list of ALs those that do not have tool support since these ALs cannot be used in industrial settings. We obtained this information about the lack of tool support directly from the papers or from email responses by AL proposers. Some AL proposers explained that building a tool is their future work. Out of 120 ALs, we excluded (a) 19 ALs that do not have any tool support; (b) 2 ALs (going back to the early 90s) have been named in some papers but we could not find any further description anywhere; (c) the research papers of 2 ALs were in Chinese. Consequently, we considered the remaining 97 ALs[8].

By considering the paper authors of these 97 ALs, three different groups of people emerge: (i) industrial experts who wrote a relevant paper, (ii) academic researchers who wrote a relevant paper with some industrial validation, and (iii) academic researchers who wrote a relevant paper with no industrial validation. We contacted these three different groups of people asking them for their involvements in our study.

(i) *Industrial experts who wrote a relevant paper:* we found 33 papers containing industrial experts as authors. These papers describe 23 different ALs. We directly contacted industrial experts asking if they are available for an interview. We asked also a reference to contact any other practitioner potentially interested in this study. We call these invitees *paper authors*, or PAs.

(ii) *Academic researchers who wrote a relevant paper with some industrial validation:* we found 13 papers that show an industrial validation of an AL or of an approach that makes use of an AL. These papers describe 11 different ALs (two of them are also used and described in papers containing industrial experts as authors). We contacted the authors of these papers asking for a reference contact within the company in which the approach presented in the paper has been validated. Our general intent is to interview industrial partners of the authors to gain a better insight on the usage of ALs. We asked also for any further industrial contact potentially interested in

2. The list of ALs is shown at http://goo.gl/e6vda. Initial ALs are shown in light blue.

3. Google Scholar website: http://scholar.google.com.

4. Microsoft Academic Search website: http://academic.research.microsoft.com.

5. Query was performed on July 4, 2011.

6. The queries were performed on July 4, 2011.

7. The complete list of ADLs can be found at: http://goo.gl/e6vda.

8. The 97 ALs can be found at http://goo.gl/e6vda marked with white rows.

this study.
(iii) *Academic researchers who wrote a relevant paper with no industrial validation:* we found 68 papers that have no industrial validation. These papers cover 65 ALs; however two of them are also taken into account by papers with industry authors. We contacted these authors to ask if they can provide further information about the validation of the AL considered in the papers. We also asked to interview their industrial partners and any further industrial contact potentially interested in this study.

In summary, we sent 369 emails to these three groups and we obtained the following results:

- 110 persons replied;
- 77 persons did not reply directly, but they were in CC (Carbon Copy) to emails replied by their co-authors, then they are aware.
- 59 email addresses do not exist anymore;
- 123 persons never replied;

Among the 110 answers, 15 persons indicated their willingness to be interviewed or to fill in the questionnaire. All of these 15 practitioners belong to the PA group. (see Step 1 of Figure 1).

Step 2: in addition to these people we contacted 63 people from our industrial contacts and 15 of them were available to be interviewed or to fill in the questionnaire. We call IC - Industrial Contact - this set of practitioners (see Step 2 of Figure 1).

Step 3: we recruited 18 people from both members and guests of the IFIP Working Group 2.10[9], and members of the following mailing lists: ISO/IEC/IEEE 42010 standard[10], and IASA (An Association for all IT Architects)[11]. We call OT - Other - this set of practitioners (see Step 3 of Figure 1). Step 3 resulted in a population of N=48 industrial contacts.

Summary: to every contact we proposed either to be interviewed by phone or to fill in the questionnaire (as a second choice).

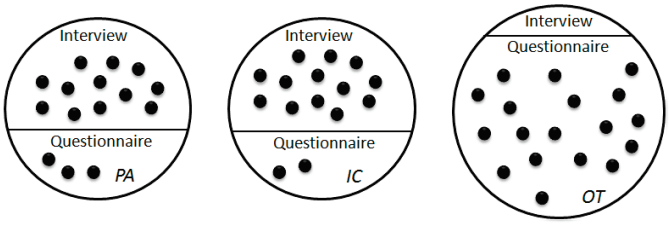

Fig. 2. Population involvement

As shown in Figure 2, 12 people of group PA decided to be interviewed, while the remaining 3 people preferred to fill in the questionnaire. Of group IC, 13 people chose to be interviewed and 2 decided to fill in the questionnaire. All 18 practitioners of group OT filled in the questionnaire. In summary, 48 practitioners participated in the survey, 25 participants were interviewed and 23 participants filled in the questionnaire.

3.3 Phase 3 - Designing the survey

The main purposes of our study are to understand which and how ALs are used in the software industry, why some ALs are not used in practice, and what are the missing AL features according to practitioner' needs. In this survey, we ask participants 51 questions: 42 are open questions and the rest are guiding questions. The open questions allow them to freely discuss their individual experiences. The questionnaire and the interview are structured identically. There are seven sections (a-g) and its flow is shown in Figure 3. Each block in Figure 3, except for eliciting company and personal information, is labeled with the research questions it aims to address (see Section 1.3). The seven questionnaire sections are[12]:

Introduction: We explain the purpose of our study, which is to *investigate the use of notations and languages for architecture descriptions (called Architectural Languages or ALs) in practice*. We disclose the investigators privacy and other administrative information. We provide general information about the study and the definitions of the terminology that we use. These definitions include software architecture and architecture description based on the ISO/IEC/IEEE 42010 standard. A Software Architecture is defined as *"the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution (ISO/IEC/IEEE 42010)"*. An architecture description language is defined as *"a form of expression used for architecture description (ISO/IEC/IEEE 42010). It may be a formal language (like Acme, Darwin, AADL), a UML-based notation, as well as any other means you may have used to describe a software architecture"*. It is important to notice that, while the questionnaire used the term ADL as defined above, this paper uses a more organized terminology, as already introduced in Section 2.

Company info (a.1-a.3): In this section, we ask the participants about their experiences and their organizations. We ask for the size of their organization, the number of architects within the organization, and if their organization recognizes software architect as a distinct professional figure.

Personal info (b.1-b.5): For personal information, we ask how long a participant has been with the organization and in software development, the current role, work activities and the type of projects in which a participant is involved. This information allows us to understand the working environment of the participant and to analyze demographical information about the participants.

Generic questions on the current use of software architecture descriptions (c.1-c.5): We expect that there are different ways to describe a software architecture,

9. http://www.ifip.org/bulletin/bulltcs/memtc02.htm#wg210
10. http://www.iso-architecture.org/ieee-1471/
11. http://www.iasaglobal.org/iasa/Chile.asp

12. The full questionnaire may be found at http://www.di.univaq.it/malavolta/files/ADLsurveyQuestionnaire.pdf.

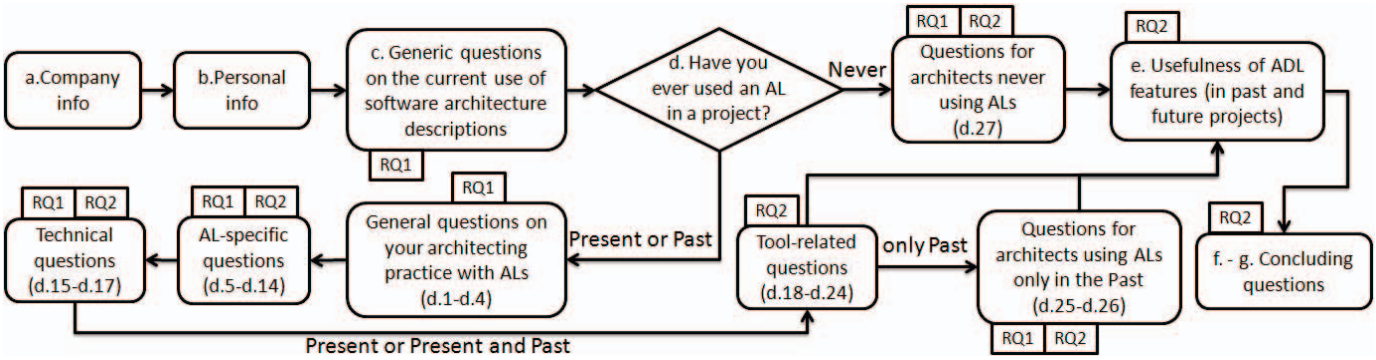

Fig. 3. Interview and questionnaire flow

including a mix of formal, semi-formal and informal notations based on different circumstances. In this section, we ask participants how they document their architecture and also which kind of architectural information is important for them. We firstly ask participants what are the three most important information that matter to them with respect to describing software architecture. Then we ask them whether they need to analyze their architecture model semantically and whether they need to generate code from their models. We also ask if there is any need to use a formal notation in the documentation of their systems. We ask the participants if they use semi-formal languages (such as UML) as an organizational standard to document their software architecture. Participants are asked to list all the notations they use within the organization. We then ask how many people in the organization use ALs, and if UML is used in some way within the organization.

AL Usage (d.1-d.27) In this part of the survey, we ask practitioners if they have used an AL in the past, if they are currently using an AL or if they never used an AL. Then, we try to understand which ALs are used, and how ALs are used in the software development life-cycle (e.g., for analysis, code generation, etc.). We then ask the participants about the criteria for choosing their ALs and AL limitations. We question participants if they model architectural view(s), and in case which views they use.

In order to understand whether an AL is sufficiently useful, we ask participants if they customize their ALs to suit their needs. We also ask whether they use a verification engine to verify architectural models, and if they use their AL to capture and reuse architectural styles and patterns. The use of tool is essential for using an AL properly or effectively. To this respect, we ask participants how satisfied they are with the AL tools they are using, which features are missing from a tool and what is unnecessary in the tool they use. Finally, we ask what kind of representation (visual, textual, sketches) is preferred.

Present/Past - for participants who use ALs in the past or are currently using ALs.

1) *General questions on your architecting practice with ALs (d.1-d.4):* We try to understand which ALs they use or used and how they use or used the ALs.
2) *AL-specific questions (d.5-d.14):* Here we ask participants to focus on one project they have been involved in where using an AL. Then we ask specific questions to understand details of their experience.
3) *Technical questions (d.15-d.17):* We ask technical questions about used ALs, focusing on a single experience.
4) *Tool-related questions (d.18-d.24):* We ask questions that are related to AL tool support.
5) Only Past - *Questions for architects using ALs only in the Past (d.25-d.26):* we want to understand why participants do not use ALs.

Never - for participants who never use ALs (d.27) We ask questions to understand why they never used ALs.

Usefulness of AL features in past and future projects (e): Based on personal experience of the participants, we want to know how important are some of the generic features of ALs. As shown in Table 2, we have listed twenty generic features as emerged from the conclusions of [50], [12], [51] and based on our experience.

Tool support	Support for collaborative architecting
Life-cycle wide support	UML support
Support for iterative architecting	Versioning
Extensibility	Code reverse/forward engineering
Customization	Textual syntax
Analysis	Graphical syntax
Support for a library of components, patterns, etc.	Tree-based syntax
Support for multiple architectural views	Sketch-based syntax
Interoperability with other ALs	Support for the alignment of SA description with the implemented system
Architectural styles support	Well-defined semantics

TABLE 2
Generic features of ALs

Then we asked the participants to rank them. The rank is in a 5-point Likert scale, ranging from *definitely not useful* to *definitely useful*.

Concluding questions: In this section we seek any additional information that participants would like to add. We ask two open questions. Firstly, what are the additional features that a participant may wish to see in an AL. Secondly, if the participants would add any comments on this study. These questions aim at catching any issues that are important to a participant but are not asked in the questionnaire.

3.4 Phase 4 - Analyzing the Data

To understand practitioners' perceived strengths, limitations and needs about existing industrial software architecture modeling languages, we analyzed all the data gathered from the respondents. The analysis resulted in findings concerning AL usage, the role and duties of the software architect in the organization, and the usefulness of various AL features. The findings are described in Section 5. In this section we describe how we organized and analyzed the data we obtained from the respondents.

We organized the various activities we carried on in this phase of our study into five steps, namely: collection of responses, data organization, vertical analysis & coding, horizontal analysis, and results & findings discovery. Each activity focuses on a specific aspect of data analysis and creates a building block towards the final discovery of the relevant findings. In the following we will give a high-level description of each activity.

Collection of responses. This initial step consists of the collection of all respondent's answers. Here we (i) collected all the answers provided by respondents of the online questionnaire as a set of spreadsheets, and (ii) obtained all the recordings of the phone interviews. The outcome of this step consists of a set of spreadsheets and audio recordings.

Data organization. The main goal of this step is to have a homogeneous representation of the collected data, from the online questionnaire and the phone interview. We firstly created a spreadsheet (called $data_{sp}$) with a structure fitting in the contents of both instruments; this task was straightforward since we followed the structure of the online questionnaire when making the phone interviews. Basically, each column in $data_{sp}$ represents a question in the online questionnaire, and each row represents all the answers given by a specific respondent. We then populate $data_{sp}$ with the answers from the online questionnaire and the transcribed recordings of the phone interviews. The transcription of the recordings is word-for-word and has not been edited in any way.

Vertical analysis & coding. In this step we analyzed the responses given to each question to find trends and common perceptions about the specific topic raised by the question.

At this point we had to distinguish between closed-ended questions and open-ended ones. For closed-ended questions such as *a.3* or *e*, we statistically analyzed the number of occurrences of a given response to draw observations [52], [53], [54]. For open-ended questions such as *d.1*, *d.2*, *d.9*, we interpreted the answers and filtered the responses into categories. Each category is composed of two pieces of information: a unique identifier to be statistical analyzed and a textual description for defining the meaning of the category. Categories have been attached to "chunks" of responses of varying size: words, phrases, sentences, paragraphs, etc. Categories came either from our research questions, concepts within the software architecture area, key concepts that the respondents brought into the study, etc. The set of categories evolved as additional responses were analyzed.

We used *coding* technique for qualitative data analysis [52]. Our main goal is to encode qualitative data such as responses to the open-ended questions so that quantitative and standard statistical data analyses can be applied. In order to mitigate possible biases, the coding procedure and its subsequent analysis have been performed by two researchers; then, the obtained categories have been checked by a third researcher. The outputs of this step are: (i) the categorization of responses; (ii) a description of the trends from the statistical analyses. The results are reported in Tables 3, 4, 5, 6, 7, 8, 9).

Horizontal analysis. In this step investigated the data to explore possible relations across related questions. We *cross-tabulate* and group the responses, and make comparisons between two or more nominal variables [54]. We have two goals in mind:

1) *check consistency*: we cross-analyzed different questions in order to make a sanity test of the given responses. This analysis helped us in (i) spotting issues in question *d* regarding the terminology used in the questionnaire (see Section 7) and to mitigate any issues accordingly, (ii) checking that all the other questions were answered coherently;
2) *verify hypotheses*: while designing the questionnaire we kept track of interesting relations that may exist between the responses of different answers. In this context, we used cross-tabulation as a planned strategy for evaluating the actual existence of those relations. This part of our analysis brought us into interesting findings, such as *F2*, *F3*, and *F4* in Table 1.

Results & findings discovery. In this step we collected the results of all the statistical analyses, we extracted key findings and we organized them into a mind map. Then, we set up a series of brain storming meetings in which we discussed the points of the mind map and possible unforeseen relations of the results. These meetings have been carried out iteratively for both the vertical and horizontal analyses to consolidate the results and the categories defined in the previous analysis steps.

4 Overview of Survey Participants

The participants of this survey are from 40 different companies over 15 countries. Companies participating in

the study are representative of both small-medium companies (52%) and large organizations, the latter counting 1.000-4.999 employees (21%) and 5.000 employees or above (27%).

The community of architects in such organizations ranges from 1 (freelance consultants) to 5000 members (large international organizations), with an average of 143 members. The role of an *architect* is understood in broad terms, as *an employee that is responsible for architecting task's*. In 58% of the participating organizations do recognize 'software architect' as a distinct professional figure, while in 42% of the organizations software architects are only de-facto roles.

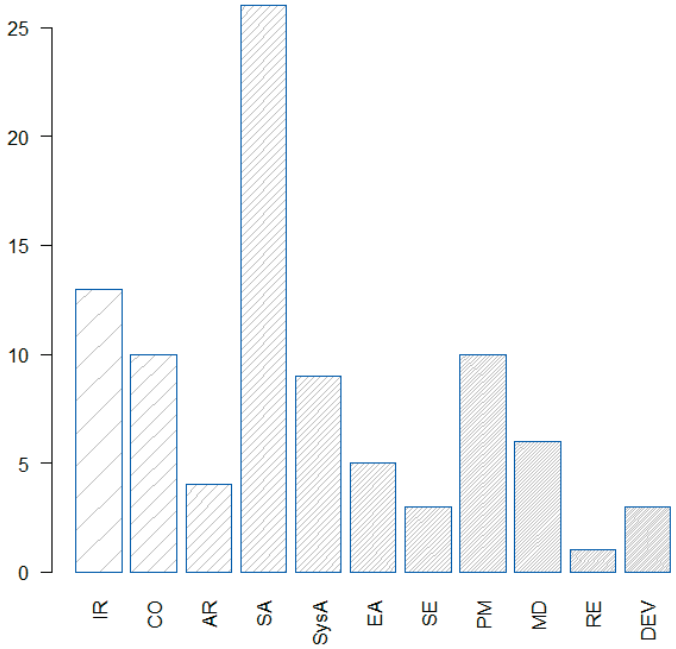

Fig. 4. Current role of participants, where: **IR** = Industrial researcher, **CO** = Consultant, **AR** = Academic researcher, **SA** = Software architect, **SysA** = System architect, **EA** = Enterprise architect, **SE** = Software engineer, **PM** = Project manager, **MD** = Modeler/designer, **RE** = Requirements engineer, **DEV** = Developer.

By analyzing the data represented in Figure 4, we see that about 54% of the respondents declared to be software architects, 19% are system architects and 10% enterprise architects, for a total of 30 out of the 48 participants. Other roles have been mentioned as reported in the Figure. Of these, all academic researchers either moved to academia after long-lasting industrial career, or do research part-time. Many indicated multiple roles, such as developer and software architect, or software architect and consultant.

When asked about the industrial activities they perform, the respondents, which are all industrial people, indicated to be involved in software production (87.5%), research (29%) and training (12.5%) activities. Figure 5 further illustrates the activities overlap. Obviously, participants often have more than one role (for this reason the sum of the percentages is more than 100%). Respondents have experience in their current role from between 6 months (with 8 years of software development experience) to 20 years, with an average of about 7 years of experience.

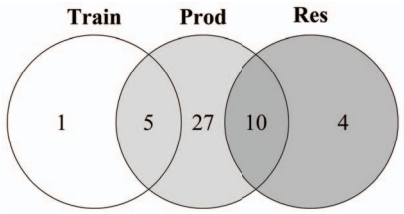

Fig. 5. Participants role (all participants are industrial), where: **Prod** = people involved in production, **Res** = people involved in research, and **Train** = people involved in training activities.

Participants of this study worked with different kinds of systems. We grouped the kinds of systems based on their critical nature as an indication of their (potential) need for more or less formal ALs. We defined two main groups of systems:

- *Critical*: they are characterized by the perceived high severity of consequences that system failures may cause[13]. They mainly have hard/strict timing and/or reliability constraints.
- *Non-critical*: they are non-critical systems, and mainly have soft timing or reliability constraints.

According to the above distinction, *critical systems* may pertain to domains like automotive, avionics, defense and homeland security, industrial automation, business information, finance; whereas, *non-critical systems* may concern media and entertainment, education, mail and messaging, project management, and information systems in general.

The majority of the participants (about 62%) was working exclusively on critical systems, while about 38% of the participants was working on non-critical systems. It is important to note also that 14% of the participants were working on both critical and non-critical systems. Projects in which participants are involved have a duration that spans from 6 months to 9 years, and the projects involve team members spanning from 6 to 10000 people.

5 Main Findings and Results

This section reports on the main results we extracted from the collected data[14]. Following the questionnaire structure described in Section 3.3, this section describes the outcomes related to the five sections from *c* to *g*. Instead of providing a question-by-question reasoning, we analyze clusters of questions, as described in Figure 6.

13. Definition provided by the steering committee of ISARCS, the International Symposium on Architecting Critical Systems, http://www.isarcs.org/

14. The raw data on which this article is based can be found at http://goo.gl/XpiKi

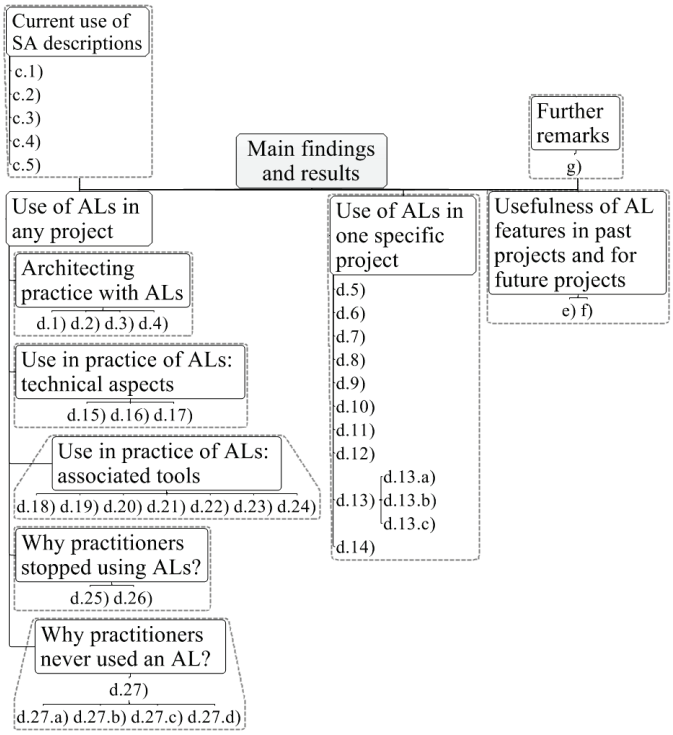

Fig. 6. Clusters of questions

We identified two main findings (bringing an overall message to the reader) and a list of detailed results coming from the analysis of each cluster of questions (bringing reflection on specific needs for AL features). The two main findings are summarized here below and discussed in Section 6.

1) *Extrovert versus introvert nature of architects role:* ALs need to be simple and intuitive enough to communicate the right message to the stakeholders involved in the architecting phase (extrovert), but they shall also enable formality so to drive analysis and other automatic tasks (introvert).
2) *All top used ALs have been originated in industry:* The three top-ranked ALs used in industry (AADL [24], ArchiMate [55], and industry ad-hoc notations) sum up to 41% among all the ALs. This lets us conclude that, based on our study population, the top used ALs in industry are those introduced by industrial researchers and practitioners.

The following sections report on the analysis of each cluster of questions in Figure 6. For each cluster we include a Table summarizing the results of each question, as well as the discussion of the cluster. Whenever appropriate the discussion refers back to the two findings described above.

5.1 Current use of SA descriptions

Table 3 reports the answers to questions of the group *"c. Generic questions on the current use of software architecture*

c.1)	**Question:** What are the three most important information that concern you during architecture design?
	Answers: Respondents identify the characteristics of the system to be developed (72.3%), the requirements coming from the context of the system being developed with possible external constraints (such as time, budget, stakeholders expectations, business goals, state of the art, state of legacy systems, other systems to integrate with) (61.7%), and the definition of functional and extra-functional requirements (36.2%).
c.2)	**Question:** Do you have the need to analyze your architecture description semantically or to generate code? Are there any needs for documenting your architecture design using a formal notation?
	Answers: 63% of respondents have the need to analyze architecture descriptions semantically. The kind of analyses that the subjects named in the answers are: (i) data flow analysis, (ii) run-time dependencies analysis, (iii) performance, (iv) scalability, (v) security, (vi) requirements analysis (from either informal specifications or UML models), (vii) simulation, (viii) finding design flaws, (ix) change impact analysis, (x) cost/value analysis. The main reasons why some other subjects do not feel the need for semantic analysis or code generation (37% of participants) are: (i) architecture descriptions are too abstract, and their formal description may become too complex; (ii) architecture descriptions are used only for documentation and communication; (iii) analyses are done manually, i.e., manual reviews.
c.3)	**Question:** Do you use standard notations (for example UML, Visio stencils, a formal language, etc.) for describing architecture design in your organization? If the answer is yes, please list the standard notations you use. If the answer is no, please describe if this could be a limitation for you and how you specify architectural descriptions of the system-of-interest.
	Answers: In general, 90% of the respondents indicated that they do use organization-wise standard notations, while 10% do not. In particular, we analyzed the level of formality of the standard notations, and we found that about 21% of the respondents' organizations use formal notations (including AADL, SDL languages, Simulink), 93% use semi-formal notations (like UML and company-specific UML profiles, SysML, ArchiMate), and 38% use informal ones (i.e., box and lines notations eventually in MS Word/PowerPoint, and Visio stencils). We also analyzed the standard notations according to their domain specificity. We defined two groups: generic, meaning any notation not bound to a specific domain, like UML as is; and domain-specific, including domain-specific languages (e.g., AADL) and domain-specific UML profiles (e.g., SysML, MARTE). We found that 88% of the organizations use generic notations, while 44% use domain-specific notations. Clearly, some organizations (33%) use them both, not necessarily in the same project.
c.4)	**Question:** To your knowledge, how many people in your organization use ADLs?
	Answers: 31.5% declare that the number of between 1 - 9 people use ALs in their organization; 22.9% of respondents say that nobody use ALs in their organization; 22.9% of organizations have between 10-99 people using ALs; 14.9% of organizations have 100-above people using ALs.
c.5)	**Question:** UML can be considered as a de-facto industry standard for documenting software architectures. How do you relate your work to UML? Do you use it in some way?
	Answers: Respondents say that UML is mainly used for: (i) describing implementation details of the system, (ii) describing the architecture from a high-level point of view, (iii) documenting and detailing specific parts of the system, (iv) documentation purposes, (v) modeling the allocation of logical elements to deployment elements, (vi) tracing requirements (just one case), (vii) dividing the system into components, and (viii) describing use cases. 95% of the respondents use structural diagrams, 61.9% use behavioral diagrams, and finally, 57.1% make use of both structural and behavioral diagrams.

TABLE 3
Questions on the current use of SA descriptions.

descriptions". In the results, we observe that by far organizations mostly use semi-formal and generic notations. These results go to the cost of formal- and domain-specific notations like ADLs. Apparently, organizations opt more for notations that are not specialized for their specific application domain. This is surprising as most organizations in our survey work with critical systems and we expect that they are more likely to use such formal/domain-specific notations.

5.2 Use of ALs in any project

5.2.1 Architecting practice with ALs

This cluster of questions (see Table 4) raises one of the main findings described in Section 5: top used ALs have been produced by industry, i.e., industrial adoption of ALs produced by academia is minimal. Moreover, we can notice that UML is largely (86%) used in industry to represent software architectures.

d.1)	**Question:** Which ADLs did or do you use? If you use(d) many, please list them.
	Answers: 86% of the respondents' organizations make use of UML or a UML profile. 9.3% make use of ad-hoc or in-house languages. Apart from ad-hoc languages, the most used ALs are AADL [24] (16.3%), ArchiMate [55] (11.63%), Rapide [56] (6.98%), and EAST-ADL [57] (4.65%). Moreover, only 6/48 respondents use ADLs exclusively, 17 mix ADL & UML, and 20 UML exclusively.
d.2)	**Question:** In general, when you use an ADL on a project, what is your workflow? For example, brain storming on whiteboard/paper, then translation into the ADL; describe the SA directly in the ADL tool; etc. . .
	Answers: ALs are usually used iteratively, each iteration aiming to refining the current architecture description. Also, when the use of an AL is coupled with brainstorming activities, the obtained AL specification is proposed to all participants of brainstorming sessions and refined. It is important to emphasize that only one respondent declares to start from requirements; all other respondents take design as input.
d.3)	**Question:** How do you check that the system-of-interest conforms to the architecture description?
	Answers: While 20 (out of 31, ≈65%) respondents use informal reviews to check architecture compliance, there is a fair distribution of inputs used to do the checking (from informal documentation, to formal executable models, to test cases). Similarly, checking activities concern any type of software life cycle phase (backward for requirements compliance and forward for code inspection).
d.4)	**Question:** Do you use specific architectural styles while designing a software architecture? If the answer is yes, please name which styles, how their constraints are enforced on architecture descriptions, and if the used ADL and its tool support the style. If the answer is no, please describe why you do not consider architectural styles.
	Answers: About 62% of the respondents declare to use architectural styles (21 respondents). Among the others declaring not to use an architectural style, 2 respondents explain that they use either external patterns or reference architectures, and 3 respondents declare to use an AL that imposes certain elements and enforces them into a particular paradigm. Overall, about 76% of the respondents declare to use either an architectural style, or certain patterns and AL-imposed constraints. It is interesting to observe that both respondents using and not using architectural styles explain that there must be the freedom to flexibly choose the most appropriate style for the specific project/system rather than adopt a pre-defined style for the domain. This flexibility seems to generally hold also when ALs enforce architectural styles (in which cases the language should be customizable to the needs).

TABLE 4
General questions on your architecting practice with ALs.

5.2.2 Use in practice of ALs – technical aspects

Answers to this cluster of questions (see Table 5) highlight some technical aspects about the use of ALs

d.15)	**Question:** If the question makes sense in your context, have you ever extended (or customized) the ADL in your projects? If the answer is yes, please describe what (and how) you extend the ADL. If the answer is no, describe why you never extended or customized the ADL.
	Answers: About 68% of the respondents extended the ALs they used by adding new views (about 48%) or constraints (13%), or both. Some respondents further explained that the extension in some cases is just informal (17%), in some others is for analysis purposes and concerns both the language and its supporting tool (26%). Pre-existing extension mechanisms are used in 35% of those cases where respondents answered 'yes' to d.15; when pre-existing extension mechanisms are used, new high-value language features (i.e., new features that may evolve the language towards the whole company needs, and not just simply language decorations) are introduced. About 32% of the respondents used ALs 'as is' with no extensions. Reasons mentioned are: because there was no perceived need (55%) or because there were insufficient resources to do so (18%).
d.16)	**Question:** Do you use facilities to interact with verification engines? If the answer is yes, please describe which facilities you use and how you use them. If the answer is no, please describe why you are not using those facilities.
	Answers: About 76% of the respondents do not use facilities to interact with verification engines. Most of the participants seem not to need it (69%). Some people link this to the use of high-level architecture descriptions that would not be suitable in any case also because it is not evident the return on investment; other respondents have no knowledge about verification engines. Some participants (15%) motivate this to the lack of suitable tools. 24% of the respondents (8 respondents) that do use verification engines carry out verification to check for performance or various dependability properties. Some carry out model or property checking. Most make use of engines external to the SA modeling environment (62,5%) and some refer to engines that are part of the modeling environment (37,5%). In fact, using one or the other (or both) seems to be due to what is available on the market rather than on preferences: no opinion was evident from the answers. By looking at what verification engines are used for, two specific properties emerge, namely performance and dependability (including reliability, availability, safety, and security).
d.17)	**Question:** How do you relate your approach for architecting to the concept of reuse? For example, specify if you use a repository of components definitions, styles, patterns, which concepts you reuse more, or how often you reuse architectural elements.
	Answers: Reuse is of the essence in software development. Nonetheless, to our surprise about 38% of the respondents declare either not to focus on reuse during their architecting activities and about 12% declare to do some kind of informal, unorganized reuse (e.g., copy-and-paste). Relatively few respondents (about 68%) do apply explicit reuse mechanisms in their practice: most of them (29%) reuse components definitions/realizations; others mention mechanisms such as product line engineering methods (6%), reuse of architecture descriptions (21%), architectural patterns (9%), design decisions and other architectural knowledge (15% - sometimes with the help of wiki's).

TABLE 5
Technical questions

d.18)	**Question:** How satisfied are you with ADL tools you use? Why?
	Answers: Levels of satisfaction vary: on the positive side 46% declare to be satisfied, 9% very satisfied (a total of 55%). On the negative side, 21% are dissatisfied and 12% are very dissatisfied (for a total of 33%). About 12% are neutral. Many respondents declare that tools provide insufficient support for graphical representation or they are too generic and not aligned with their specific domain/process. For many UML users, tools are too complex and heavy. For other respondents, however, verification-related tools are satisfactory even if they need building tool-specific usage skills. Users of informal tools (Visio, etc.) are generally satisfied. In some cases, tools are positively perceived while the supported languages are criticized. Interestingly, both satisfied and dissatisfied respondents ground their (positive or negative) motivation on three main classes of features: extensibility, fitness to the system to be modeled, and graphical drawing capabilities. On the positive side, these three classes naturally emerge as relevant criteria driving tool development and evaluation.
d.19)	**Question:** Do the tools miss some facility that you consider important for your activities? Which ones?
	Answers: 29 respondents filled this question. Of these, only 4 respondents (14%) have all tool facilities they need in their architecting practice. The remaining 25 respondents (86%) require tool features that are missing (see Figure 7), the top-most being analysis and simulation support (24%), better visualization (17%), and usability (17%). About 21% of the respondents consider the integration of the tool with other development environments/tools (here included documentation generation, and link to requirements and implementation) a missing facility. About 17% of the respondents say that current tools lack facilities that allow designers to specify the system from different viewpoints and to generate new views from an existing system.
d.20)	**Question:** Do the tools provide some facility that you consider unnecessary? Which ones?
	Answers: About 68% of the respondents find all tool features necessary. Seven respondents (8%) indicate being overloaded with features they do not personally use and would like to filter them out, or be prompted with features incrementally on need. We could identify the following types of unnecessary features: too much/too formal semantics (not needed to communicate architectural decisions with the client); too large scope (i.e., trying to support too much makes the tool too generic and hence it fails in representing domain-specific aspects), code generation and project management.
d.21)	**Question:** Do you use any facility for working in collaborative environments? If the answer is yes, please describe which facilities you use and how many people in average collaborate on the software architecture description. If the answer is no, please describe why you are not using those facilities.
	Answers: Out of the 48 respondents, 16 mentioned that they use knowledge sharing platforms for managing and sharing information. These tools include Microsoft SharePoint, wiki pages, etc. Since the respondents were non-specific about what type of information they share, we assume that the information is about general software development. Eight respondents specifically mentioned that they use version control system for managing software and programs. These tools include Topcased, CVS/SVN, Eclipse etc. Six respondents mentioned that they use collaboration tools to manage their architecture models, some of the tools mentioned are IBM Rational Jazz, SAVI etc. Three respondents said they did not use any collaboration tools, and fifteen respondents did not respond to this question. For those respondents who discuss the management of architecture models, some have described that small number of writers work on a part of the architecture model, but many readers share it. Collaboration on an architecture model ranges from a few people to a few hundred people. From the responses, we derive three observations: (a) collaborative working on architecture description has not been a major concern to architects, there has been no major issues reported by the respondents; (b) most respondents did not respond to this question. For those who responded that they use a collaborative tool, they said they use it for collaboration on software development, on top of architecture description; (c) a number of respondents commented that the use of collaborative tool is limited, they either do not consider an issue, or their practices are immature to use a collaboration tool, or they simply use wikis and web pages to share architecture descriptions.
d.22)	**Question:** Do you use versioning to store and keep track of architecture descriptions? How do you use it?
	Answers: It does not come as a surprise that about 82% of the respondents use some version control features to manage architecture descriptions. About 18% do not use any version control for this purpose, and seem not to need it either. Version control practices are mostly tool supported (53%), but about 18% use manual techniques (like file naming conventions). Two respondents use version control built in the AL tool itself (this being Enterprise Architect and ArchiMate, respectively) and one respondent uses wiki versioning features.
d.23)	**Question:** Which kind of representation (concrete syntax) of ADLs do you use?
	Answers: About 9% of the respondents use textual AL representations only, while 18% exclusively use visual ones; about 62% of the respondents use them both. Moreover, next to visual or textual representations, about 38% also use tree-based representations, and 32% use free-hand, informal ones.
d.24)	**Question:** What is your opinion on the usefulness of the various kinds of representation (e.g., textual, visual) of ADLs?
	Answers: It does not come as a surprise that in the majority of cases textual and visual representations go hand-in-hand, the former complementing the latter. Moreover, textual representations are easier to version control, while visual representations are instrumental to communicate high-level architectural information as well as to facilitate interaction with non-technical stakeholders. Representation needs seem to concentrate on (traditional) visual modeling and (emerging) free-hand sketching tools. As a consequence, research in ALs should concentrate on supporting these two representation paradigms.

TABLE 6
Tool-related questions

in practice: extensibility emerges as a relevant feature of ALs. Formal verification is not commonly used in practice and practitioners are not convinced about the need of making verification. Finally, reuse is not commonly performed during architecting activities.

5.2.3 Use in practice of ALs – associated tools

Table 6 reports answers to questions related to tools that are associated to ALs. From their answers, the majority of respondents are satisfied about tools. Nevertheless, they point out also several missing features of existent tools, as shown in Figure 7. Collaborative supporting tools are quite common but mostly general purpose, such as SVN. Respondents report a (not so surprising) combination of textual and graphical representations, and free-hand sketching modeling emerges as a promising research direction.

5.2.4 Why practitioners stopped using ALs?

In addition to questions d.1-d.24, we included two questions (d.25 and d.26) reserved to those participants that declared to have used ALs in the past, but do not use

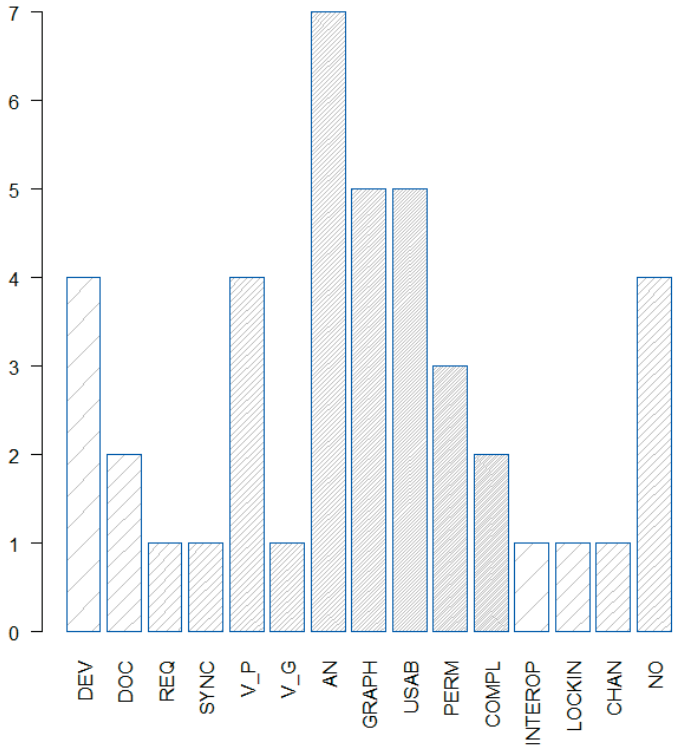

Fig. 7. Missing features of the tool, where: **DEV** = Poor link to implementation/code, **DOC** = Poor documentation generation, **REQ** = Poor traceability to/from requirements, **SYNC** = Hard to sync with info outside the tool, **V_P** = Poor support for multiple viewpoints, **V_G** = No reverse architecting support, **AN** = Better support for analysis, **GRAPH** = Poor graphical representations, **USAB** = Poor usability, **PERM** = Not permissive, **COMPL** = Incomplete, **INTEROP** = No interoperability with other tools, **LOCKIN** = Tool/vendor lock-in, **CHAN** = Poor support for managing changes in the SA, **NO** = No missing features.

them anymore (see also arrow labeled 'only Past' in Figure 3). The summary of their responses is given in Table 7.

d.25)	**Question:** If you used an ADL in the past, why you do not use it anymore?
	Answers: Only five respondents used ALs only in the past. Only one brought a technical reason (being formal ALs too formal for his purposes), while the remaining four simply changed role or responsibilities.
d.26)	**Question:** If you used an ADL in the past, can you please list which were its positive and negative aspects?
	Answers: The respondents further provided feedback on the positive and negative aspects of the AL they eventually have used. On the negative side, four of them consider formal ALs far too formal, requiring specialized competencies for little benefits (bad for communication and understandability). The fourth respondent is positive but did not provide any elaboration.

TABLE 7
Using ALs in the past only

Among the reasons to stop using ALs, the formality of some ALs is considered to be a relevant issue, important enough to hinder the respondents using formal ALs in future projects. This shows how different concerns (from different stakeholders) drive selecting (or neglecting) the AL to be used. Some respondents, while recognizing the importance of unambiguous and common notations, consider as a major problem the over-formalization as well as the inability to model design decisions explicitly in the AL. The following two quotes well express this problem:

> *"On the positive side if you know the formal AL well and use it correctly, it's fairly unambiguous; provides a common basis/language, to enable teams of architects to collaborate. However, on the negative side requires specialized skills and tools; tends to be over-formal and "stiffening", limiting creativity; not typically understandable by business people, so different representations of the solution will be required anyway."*

> *"In my view, formal ALs tend to over-formalise the description of systems. In general, they are very precise to describe aspects of the system that are trivial or uninteresting, whether the most interesting discussions and design decisions are no documented in the AL and we have to use a different notation."*

This result is in line with our first main finding, highlighting the need to cover a combination of features addressing different types of concerns for both external communication and internal goals of architecting (once again reflecting the extrovert-introvert nature of the role of architects).

5.2.5 Why practitioners never used an AL?

Five practitioners declared to have never used an AL. For such a case (shown by arrow labeled 'Never' in Figure 3) our survey included four specific questions (in cluster d.27) reported in Table 8. The aim is to understand the reasons behind this decision, as a way to identify useful insights for future research directions on ALs.

About 62% of the respondents did consider using an AL but decided not to pursue this endeavor further, while 38% of the respondents did not consider using any AL. In the former case, motivation against using ALs includes insufficient perceived ROI, too much formality, too little support for communication, and absence of an industrial standard. In the latter case, the general opinion is that ALs are too heavyweight (referring to formal ALs). Quoting one of the respondents:

> *"For me and my purposes (formal) ALs seem a bit too heavy, too much effort to learn it. Currently, I do not see a real benefit."*

d.27)	**Question:** Have you ever "considered" to use an ADL? If the answer is yes, please answer questions d.27a, d.27b, and d.27c. If the answer is no, please answer to question d.27d.
	Answers: About 62% of the (13) respondents that never used an AL before, did consider using it.
d.27.a)	**Question:** Why you (or your organization) decided not to use ADLs? Please, answer to this question only if you answered yes to question d.27.
	Answers: The motivation against using ALs includes insufficient perceived ROI (i.e., Return On Investment), and too much formality (of formal AL) or too little support for stakeholder communication. Fewer participants mentioned the absence of an industrial standard.
d.27.b)	**Question:** Are there any ADL features the lack of which influenced your decision? Please, answer to this question only if you answered yes to question d.27.
	Answers: As for the specific missing AL features, the participants again mentioned tools support, usability, integrability and lack of standards.
d.27.c)	**Question:** Are there any features that you would require in an ADL? Please, answer to this question only if you answered yes to question d.27.
	Answers: Some additional AL features mentioned as required are *traceability* and *cross-view consistency*.
d.27.d)	**Question:** Why you (or your organization) never considered to use an ADL. Please, answer to this question only if you answered no to question d.27.
	Answers: For those participants who never even considered using an AL, the general opinion is that ALs are too heavyweight (referring to formal ALs), again in terms of required investments, effort, and formality.

TABLE 8
Never used AL

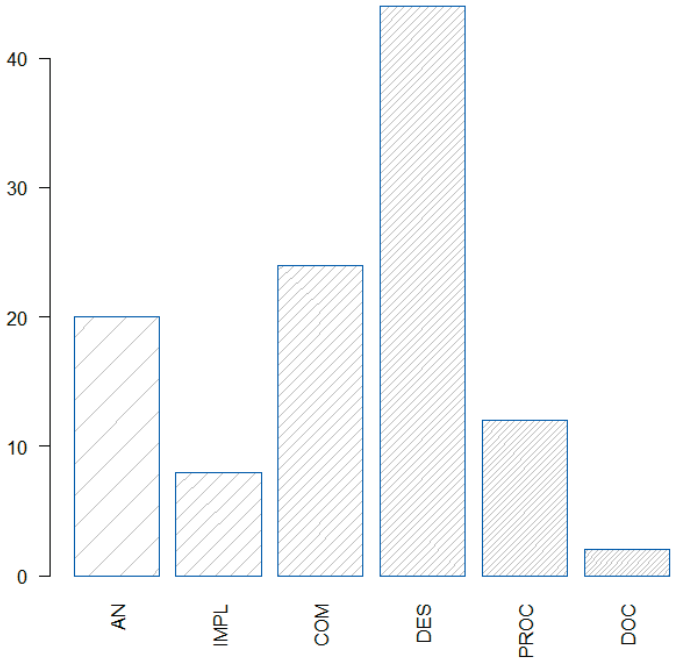

Fig. 8. Type of needs, where: **AN** = Analysis, **IMPL** = Implementation, **COM** = Communication, **DES** = Design, **PROC** = Process, **DOC** = Documentation

5.3 Use of ALs in one specific project

As discussed in Section 3 and as part of section *"d. ADL Usage (d.1-d.27)"*, we also asked participants to focus on their use of ALs in the context of one specific project they worked on (see cluster of questions d.5 to d.14 in Figure 3). Our aim is to dig into details of recent experience. By looking at the results summarized in Table 9, we can draw the following observations.

First, the decisions leading to the selection of an AL *do not* mainly depend on the specific domain. This result discredits a belief commonly shared in the research community, namely that an AL is selected to fit at best the system to be realized or its application domain. Instead, what emerges from d.5 is that when selecting an AL the presence of a pre-existing community or tool support, and in-house competencies are the features that matter.

Second, Figure 8 illustrates that the most important needs stakeholders want to be satisfied are support to the design (d.6 and d.8), followed by communication and then analysis (d.6).

The need for code generation seems to be case- or project-specific: while in the specific context of question d.6 *there is almost no need for code generation*, in question c.2 (referring to AL use in general) practitioners identified it as a relevant feature. However, what emerges from a qualitative data analysis is that the only need that results in a good level of satisfaction is design, while for other criteria the respondents are either not satisfied or did not express any opinion.

The answers to d.7 reinforce the result emerging from d.6, that is: *the most important unsatisfactory need is support for communication*. While architecting is mainly concerned with reasoning, tradeoff analysis and negotiation of extra-functional properties with non-technical stakeholders, ALs do not support at best this need. In the same vein, the second most important unsatisfactory need was about analysis, which again is confirmed by the limitation in expressing extra-functional properties. The data extracted from d.11 shows once again the role of analysis of architecture descriptions. Analysis is not only a need (as highlighted in d.6), but it is also practiced by ≈74% of the interviewed practitioners. The most important reason for analysis is to check for extra-functional properties: when combining the results from d.7 and d.11, it becomes evident that the insufficient expressiveness for extra-functional properties (considered to be so relevant) turns to be a main limitation of current ALs.

Third, the role of ALs for requirements engineering emerges (d.8 and d.9): even if most of the respondents use ALs in the design phase, 48% of the respondents make use of ALs for inception and early elaboration of requirements and focus on the conceptual phase of architecting. Moreover, 84% of the respondents use an AL before finalizing the requirements. However, excluding EAST-ADL, practitioners using (formal) ALs do generally associate their use to situations in which requirements are frozen. This would suggest that using ALs for requirements engineering is in practice bound to UML.

d.5)	**Question:** Based on which criteria did you choose the ADL? Please list them.
	Answers: On the one hand, a major driver for AL selection is system/project specific: 28% of the respondents select the AL best fitting with the system to be realized, the domain of the system, or the specific project/client needs. On the other hand, the other three most important drivers (summing up to 50% of the respondents) are independent from the characteristics of the system/project under development, and rather fit in the industrial context in which an AL has to function. These are: the existence of a community providing AL-related support (25%), current architects/developers skills and competencies (25%), and tool support and costs (25%).
d.6)	**Question:** Explain what have been your needs and to which degree the ADL satisfied your needs.
	Answers: Surprisingly enough, the least important need (4 respondents, ≈12%) is for code generation and deployment support. The same holds for process support (6 respondents, 18%). The most important need is design (22 respondents, ≈66%) followed by communication- (12 respondents, ≈36%) and analysis support (10 respondents, ≈30%).
d.7)	**Question:** Explain what are the possible limitations of the ADL you are using.
	Answers: The two most recurrent identified limitations are related to the insufficient expressiveness for extra-functional properties (12 respondents, ≈37,5%), and insufficient communication support for non-architects (8 respondents, 25%).
d.8)	**Question:** In which phase(s) in the development life-cycle did you use the ADL?
	Answers: Most respondents (94%) use ALs in the design phase. This is in line with previous answers, showing that the practitioners using ALs exploit them for design purposes and are satisfied with the provided support. More interesting is the frequent use of ALs for requirements engineering (48%).
d.9)	**Question:** Did you use the ADL before the requirements are finalized? Please comment on that.
	Answers: 85% of the respondents do use ALs also when requirements are not finalized. Those respondents mentioning canonical ALs (or what we referred as ADLs) do generally associate their use to situations in which requirements are frozen, with only one exception being EAST-ADL (which supports RE).
d.10)	**Question:** Did you model the entire system with the ADL, or only part of it? Please comment on this.
	Answers: From the responses it results that the system is modeled in full (at different levels of abstractions) or in parts with a similar distribution (42% in both cases). Only one interviewee combines different modeling notations when needed. About 33% of the respondents focus on relevant system concerns (and therefore system coverage depends on the scope of the concern itself), while ≈30% focus on some subsystems only to limit complexity.
d.11)	**Question:** Did you analyze the architecture description produced with the ADL? If the answer is yes, please describe which kind of analysis was performed, and the needed skills to perform it. If the answer is no, please describe why you did not analyze architecture descriptions.
	Answers: About 74% of the respondents do analyze the SA modeled in ALs. Of these, 32% carries out analysis manually. Further, the top-most reason for analysis is to check extra-functional properties (≈48%), followed by behavioral concerns (≈24%). Other concerns are far less practiced (below 12%). The respondents that do not carry out any analysis are about 26% (in most of the cases using the AL-based model for documentation purposes). Motivation for not carrying out analysis are no value perceived (≈44%), ALs too limited/imprecise (≈44%) and lack of skills, competencies or available resources (≈11%).
d.12)	**Question:** Did you generate code from architecture descriptions? If the answer is yes, please describe how you generate code from SA descriptions (for example, what is the target language, if you generate the full system or not, etc.) If the answer is no, please describe why you do not generate code from architecture descriptions.
	Answers: About 62% of the respondents declare to do not generate code from architectural descriptions. Of the latter, ≈62% generate glue code (or code for some system portions) and ≈23% generate the code of the complete system, while ≈15% use intermediate notations for subsequent code generation. When provided, reasons brought for not generating code are: no need (≈38%), different level of abstraction between SA description and target code (≈24%), and limited AL expressiveness (≈10%).
d.13)	**Question:** Did you describe software architectures by means of multiple views? If the answer is yes, please answer questions d.13a and d.13b. If the answer is no, please answer to question d.13c.
	Answers: About 85% of the respondents declare to use multiple views for architectural description.
d.13.a)	**Question:** Which kinds of view do you use? For example, structural, behavioral, financial, etc. Please, answer to this question only if you answered yes to question d.13.
	Answers: The types of views mentioned are: structural (≈76%), behavioral (48%), physical (45%) and conceptual (41%). Other views are business-oriented (e.g., manpower allocation, strategies and costs - 31% of respondents mentioning them). All remaining views have a significantly lower frequency. Overall, these responses confirm the general trend of mostly focusing on structural aspects. Nonetheless, industrial practice covers aspects that are less common to canonical ALs. Above results show that behavioral, physical, conceptual, and business aspects are fundamental, hence requiring coverage in the definition of ALs. This partially confirms the theory put forward by Medvidovic et al. [12] on the need to extend the ALs scope with domain and business aspects.
d.13.b)	**Question:** Do you use mechanisms for ensuring views consistency? How? Please, answer to this question only if you answered yes to question d.13.
	Answers: (Further on using multi-view architectural descriptions) About 59% of the respondents declare to use mechanisms to ensure cross-view consistency. In fact, while 17% do not give details, the remaining 53% simply capture all system elements requiring consistency into a single model, and ≈18% ensure consistency only partially by using standardized syntactic rules like naming conventions. It is worth noting that 41% say that they do not use any explicit mechanism for checking views consistency. This sometimes depends on the specific project. Of this latter set, ≈50% of the respondents check view consistency manually (i.e., with no formal or formalized method).
d.13.c)	**Question:** Why you do not use multiple views while architecting? Please, answer to this question only if you answered no to question d.13.
	Answers: Only five respondents, or 15%, do not describe SA with multiple views. Notwithstanding these relatively low numbers, the responses give interesting research directions for multi-view support in future ALs. Indeed, two respondents do not use multiple views because of the lack of tool support for visualization, presentation mechanisms, and incremental definition. This calls for better research and development of tools and techniques. The remaining three respondents, however, do not see additional benefits in using multiple views in their current practice (but envisage using them in the future). This could indicate more fundamental problems requiring better research in the modeling needs.
d.14)	**Question:** Did you use more than one ADL in your last project? If the answer is yes, how do you synchronize architecture descriptions of the same system? If the answer is no, have you ever considered to use more than one ADL in a project? Please comment on this.
	Answers: About 79% of the respondents do not use multiple ALs within the same project. Most of them think that it would be too complex (≈30%). Some do not actually see additional benefits (≈21%), or never consider using them (≈21%), or they do not have the resources to manage multiple ALs (≈17%). The rest either never considered the option, or do not have the resources to do so. Only ≈21% of the respondents use multiple ALs to describe the same system: some use them for transformation purposes, while the others either keep the different representations independent or keep them aligned by using predefined rules.

TABLE 9
AL-specific questions for one project

Overall, it is difficult to find a solution that can match with all the needs. As quoted from one of the respondents:

> *"There is no perfect technology solution, you just choose which has the most convenient balance between the good things it provides and the problems it solves. If the balance is positive, you choose that solution, even if it has some flaw (e.g., first versions of Visio were horrible, but they used it because even if it crashed a lot of times, it never destroyed diagrams)."*

5.4 Usefulness of AL features in past projects and for future projects

So far we have discussed the results related to our first research question (about what industry would need from an AL). To answer our second research question (about the features supported by existing ALs) we asked the participants to rank what AL features they think are useful for their past projects, and what features would be useful for future projects. The asked question is reported in Table 10.

e)	**Question:** Based on your experience, assess the usefulness of the following list of ADL features. Please, assess both their usefulness in past projects (table column "useful in past projects") and their possible usefulness for future projects (table column "useful for future projects").

TABLE 10
Usefulness of AL features in past and future projects

The list of features have been defined based on the analysis of the ALs identified in Phase 1 of our study. The ranking is between -2 (definitely not useful) to +2 (definitely useful). Responses are summarized in Table 11.

We analyzed those features that we considered "definitely useful" (i.e., the features that are graded +2 by at least ten participants, i.e., 25% of the participants). We have found that seven AL features are considered to be useful in past projects. They are, (i) tool support *(Tool)*; (ii) support for iterative architecting *(ItArch)*; (iii) analysis *(An)*; (iv) support for multiple architectural views *(Views)*; (v) versioning *(Vers)*; (vi) graphical syntax *(GrapSy)*; (vii) well-defined semantics *(Sem)*.

When participants are asked what features they would find useful for their future projects, we have found twelve AL features considered definitely useful. These twelve features include the seven features already reported to be definitely useful, indicating that on top of the success they had with those seven features, they desire to have five more features. These additional five AL features are: (i) life-cycle wide support *(Life-cycle)*; (ii) extensibility *(Ext)*; (iii) support for collaborative architecting *(Collab)*; (iv) textual syntax *(TextSy)*; (v) support for the alignment of SA description with the implemented system *(CodeA1)*. Amongst the most important features, tool support, support for iterative architecting and support for multiple architectural views stand out.

Figure 9 provides a different view of the usefulness of past and future AL features. It illustrates AL usefulness weighted by the number of responses (per feature) and the relative ranks (between -2 and +2) expressed by each participant.

We have further analyzed the results by segregating those participants who used ADLs (as defined in Section 2) and those who use non-ADL ALs. We have found that they differ in opinions on the support for multiple architectural views. The ADL users on average think that architectural view is less useful, whereas the non-ADL AL users think it to be very useful. The difference between the two groups is statistically significant when they consider their past and future projects (F=8.364, p=0.007 and F=11.326, p=0.002, respectively). It is an indication that the ADL users focus on a single aspect of a design, whereas the non-ADL group needs to use multiple view to represent different aspects of an architecture design.

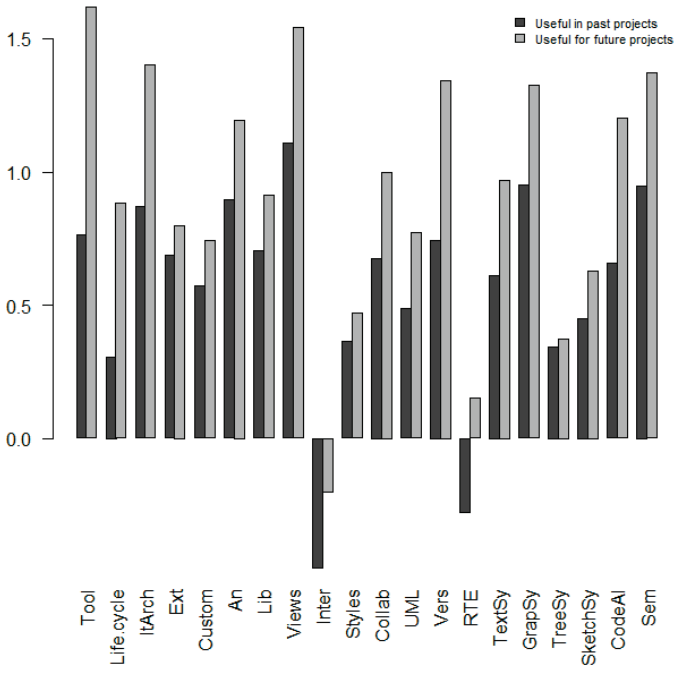

Fig. 9. Usefulness of AL features in past projects and for future projects, where: **Tool** = Tool support, **Life-cycle** = Life-cycle wide support, **ItArch** = Support for iterative architecting, **Ext** = Extensibility, **Custom** = Customization, **An** = Analysis, **Lib** = Support for a library of components, patterns, etc., **Views** = Support for multiple architectural views, **Inter** = Interoperability with other other ALs, **Styles** = Architectural styles support, **Collab** = Support for collaborative architecting, **UML** = UML support, **Vers** = Versioning, **RTE** = Code reverse/forward engineering, **TextSy** = Textual syntax, **GrapSy** = Graphical syntax, **TreeSy** = Tree-based syntax, **SketchSy** = Sketch-based syntax, **CodeAl** = Support for the alignment of SA description with the implemented system, **Sem** = Well-defined semantics.

	Useful in past projects							Useful for future projects						
	-2	**-1**	**0**	**+1**	**+2**	**No exp.**	**Blank**	**-2**	**-1**	**0**	**+1**	**+2**	**Don't know**	**Blank**
Tool support	3	4	7	9	15	8	2	0	0	1	11	22	3	11
Life-cycle wide support	2	6	13	9	6	4	8	0	3	9	1	11	4	10
Support for iterative architecting	1	4	7	13	13	2	8	1	0	2	13	19	2	11
Extensibility	2	3	8	13	9	5	8	1	2	10	12	10	2	11
Customization	1	4	11	12	7	5	8	1	2	11	12	9	2	11
Analysis	2	2	7	14	13	2	8	1	1	3	16	15	2	10
Support for a library of components, patterns...	1	4	9	14	9	2	9	1	2	5	17	9	3	11
Support for multiple architectural views	2	2	4	11	18	2	9	1	0	4	5	27	1	10
Interoperability with other ALs	9	5	10	6	1	9	8	8	5	6	7	4	7	11
Architectural styles support	0	3	19	7	4	6	9	1	3	13	13	4	4	10
Support for collaborative architecting	1	1	13	12	7	6	8	2	1	2	18	10	3	12
UML support	3	4	9	17	6	2	7	2	1	9	14	9	2	11
Versioning	3	2	8	15	11	2	7	1	1	2	12	19	1	12
Code reverse/forward engineering	7	8	11	6	3	5	7	4	5	10	10	4	5	10
Textual syntax	1	3	13	11	8	3	9	1	1	8	10	12	3	13
Graphical syntax	0	4	8	12	15	1	7	0	0	5	13	16	2	12
Tree-based syntax	1	2	13	12	1	10	9	3	0	8	11	2	11	13
Sketch-based syntax	3	2	10	10	6	9	8	2	2	7	9	7	9	12
Support for the alignment of SA description with the implemented system	2	3	9	12	9	5	8	1	1	2	13	13	7	11
Well-defined semantics	1	2	7	16	12	2	8	0	1	4	11	19	3	10

TABLE 11
Usefulness of AL features - past and future

On the usefulness of AL features for future projects, ADL users and non-ADL users slightly differ in their average opinions about the thirteen most useful AL features. The figures shown are averages based on the opinions, in Likert scale, of the participants' ranking from -2 (least useful) to 2 (most useful). Table 12 shows that in addition to support for multiple architectural views, average opinions on some other features are different. These are life-cycle wide support, extensibility, support for collaborative architecting, support for alignment of SA description and implemented system, and well-defined semantics.

There are AL features considered to be "definitely not useful". We define this to be 10% of all participants who ranked a feature -2. For both past and future projects, interoperability with other ALs *(Inter)* and code reverse/forward engineering *(RTE)* are considered to be definitely not useful.

In section *f* of the questionnaire we ask the participants if they would add any feature an AL should expose to be usable in practice. Many commonly known features have been mentioned with similar frequency. The only exception is flexibility (40% of the participants) meaning that ALs should be tunable, extensible, and seamlessly adaptable to system-specific characteristics. Features already mentioned in previous questions are usability (20%) and traceability (either to code - 20% and to other artifacts - 20%). A new feature not previously mentioned is the ability to capture conceptual feature sets (16%), like support for "real-time concepts", "exception and error handling", "design decisions linked to artifacts", "linked structural and behavioral aspects". This suggests a new trend in AL research, namely on supporting conceptual clusters of connector types reusable across architectural styles and similar to the concept of traceability link as defined in [58]. It also shows the need to adapt the used AL to the domain of the system being developed in a reusable and systematic way [51].

AL Features	**Average usefulness by ADL users**	**Average usefulness by non-ADL us.**
Tool support	1.5	1.6
Life-cycle wide support	1.17	0.83
Support for iterative architecting	1.6	1.46
Extensibility	1.4	0.7
Analysis	1.33	1.08
Support for a library of components and patterns	0.97	0.6
Multiple arch. views	0.5	1.74
Support for collaborative architecting	1.2	1.04
Versioning	1.33	1.38
Textual syntax	1	1
Graphical syntax	1.2	1.37
Support for alignment between SA description and impl. system	1.4	1.04
Well-defined semantics	1.6	1.42

TABLE 12
Future Useful Features ranked by ADL and non-ADL users

5.5 Further remarks

We finally asked the participants if they wanted to add additional feedback. Most participants further emphasized aspects already addressed throughout this study. Though, we found two answers especially interesting. One practitioner highlighted the lack of evidence of the usefulness of ALs in industrial projects. Specifically, he writes:

> "*The additional benefits over just semiformal UML models is not currently seen. We are lacking some industrial case study (or some winning story) in which they say "we used this ADL, it really helped and it saved a lot of effort using it, and we did not make so many mistakes". There is a lack of Evidence.*"

This quote calls for empirical studies providing such evidence in practice. A second interviewee provides, in our opinion, a nice summary of what a practicing architect would like to find back in the ideal AL. In his words, desirable AL features are:

- *accessible and easy-to-use language;*
- *bridge the gap between real world and research, fit what practitioners really need;*
- *support multiple views (similar to UML but with automated consistency checked between views);*
- *incremental adoption, i.e., allow the partial definition of the system, without constraining the architect to model it in full;*
- *no fancy tools, just something that works and is reliable;*
- *extensible, in both language- and tool support.*

6 Discussion

This section provides a discussion of the findings of this work. First, in Section 6.1 we analyze the two main findings discussed in Section 4. Then, we look at the results from some specific perspectives we found particularly interesting (Section 6.2). We also compare the results from previous studies with those presented in this study (Section 6.3). Lastly, we go back to our two research questions and answer them in the light of the discussion (Section 6.4).

6.1 Discussion on the two main findings

The **first finding** reveals the *introvert versus extrovert nature of architects role*. Two main identified needs, namely analysis and communication (see Section 5.3), provide evidence to support the introvert versus extrovert nature of the software architect role. An architect needs informal ALs when communicating with external stakeholders and (s)he needs more formal ALs with well-defined semantics (and methodologies) when dealing with technical and development-related concerns. In fact, respondents needing support for analysis also expressed AL limitations for communication with non-architects. This highlights a tradeoff between *semantic analysis* support (typically calling for more formality) and *support for communication* (typically calling for more intuitive, and less formal ALs).

When respondents indicate the need for either design or communication, we observe that the main limitations they found in ALs concern (again) expressing specific system properties, poor understandability towards non-architects, and insufficient formality. When they mentioned analysis as a need, they also mentioned limitations in expressing specific system properties (e.g., performance, error-handling, extra-functional properties in general). This indicates the need to further improve ALs in this direction: ALs need to be simple and intuitive enough to communicate the right message to the stakeholders involved in the architecting phase, but they shall also enable formality so to drive analysis and other automatic tasks. This is in line with the work of e.g., Kruchten [59] and Hoorn et al. [60] reporting on the dual role of the architect: as communication bridge (communicating with the customer and the business, i.e., the extrovert nature of the architect role) and as design and development means (i.e., realizing its introvert nature).

This finding resounds two clear (and distinct) needs that were described in software architecture papers. Indeed, in the early years, academic focus was mostly oriented toward the use of ALs for conducting (formal) analysis [1], [30]. This is why a considerable amount of academic research effort has been spent in producing formal ALs. On the contrary, the industry focus has been mostly on communicating design decisions and solutions [21], that hampered the advent of UML as a language to describe software architectures. Practicing architects nowadays (as confirmed in this survey) emphasize the need to reconcile informal and communicative notations with more formal and analyzable ones.

We do not suggest that *informal* ALs are *the* answer to reach out to industry and ensure industrial adoption. On the contrary, formal ALs do offer rich expressive power that is necessary, and desirable, in many situations. Unfortunately they are often too costly (in terms of e.g., effort to learn them or investment to introduce them in the current development practice) for industry to adopt them. For these (and other) reasons industry seems to settle for informal (yet ambiguous hence error-prone) ALs that are less suitable for analysis but that do more easily support communication purposes.

Communication, however, could be interpreted in two ways: (1) as the need to model an architectural design solution (and hence use its documentation as a means for communicating its characteristics); or (2) as the need for different stakeholders to effectively communicate about the architectural solution and about how it satisfies their various concerns. The communication perspective has been broadly discussed in works specifically focused on the *extrovert* role of the architect as communicator and negotiator (e.g., [61], [62], [63]). While ALs typically support the modeling perspective, they often neglect the communication perspective. Recent works investigate how to support both by helping designers during

design sessions, without ALs interfering with the natural interaction dynamics [64].

In fact, to address both modeling and communication perspectives, the expressive power of an AL should explicitly address different types of stakeholders at the same time, and hence act as a natural communication means. The main generic approach addressing this problem is provided by the ISO/IEC/IEEE 42010 Standard on architecture description [50], which both regulates how to describe an architecture by using viewpoints (and multiple model kinds), and explicitly links them to the communication needs of multiple stakeholders (by means of stakeholder concerns). We think that future research in ALs should follow a similar example, by making explicit the way languages do support communication toward which types of stakeholders. This would also provide a way to better identify what languages could be combined in a cost-effective way (e.g., by supporting the communication needs of the specific stakeholders involved in a certain development project). In this way, industry could flexibly select the right ALs for the situation at hand. Of course, for this to be possible further research is needed to define language and tool mechanisms supporting AL combination in a more flexible way.

Further, as discussed in Section 1.2, existing works have identified different types of AL requirements. For instance, Clements [21] classifies AL features as system-, language- and process-oriented; Medvidovic et al. [12] identify technology-, domain-, and business-related needs. In fact, our study highlights that different language requirements and needs are not equally important, or at least do not have the same weight when we target industry adoption. As a matter of fact, business needs (like time to market and flexible/easy fitting in the working practice) are clearly major drivers, and overpower language-oriented ones (like level of formality) if tradeoffs are necessary. To cope with this, future ALs should at the same time (i) reconcile informal and communicative notations with more formal and analyzable ones, and (ii) address business needs during the architecting activity. Explicit techniques and mechanisms to flexibly and dynamically adapt AL selection and support seem the key to achieve industry adoption yet mitigating the risks brought by informality.

The **second finding** confirms that *top used ALs have been produced by industry*. What emerges from the collected data is that the 42% of the respondents make an exclusive use of UML and UML profiles, the 35% uses ADLs and UML together, while only the 17% makes and exclusive use of ADLs when describing software architectures.

Further, if we exclude UML and UML profiles for SA modeling (used by the 86% of the respondents), many are the ALs used by the interviewed people (see Figure 10[15]). First of all, it is important to note that if we compare our initial list of ALs and those elements shown in Figure 10, some new ALs emerged, they are: MIND, SDO, SAMM, CCL, SLX, HOOD, SDL, IAF, KISS, Yourdon DFD. If we consider those elements, we can see that some of them cannot be considered as ALs (e.g., IAF is an architecture framework, KISS is a development method, etc.), whereas others can be considered as ALs (e.g., Modelica, SAMM, etc.). However, as stated in Section 3, our initial list of ALs has been used with the only purpose of identifying practitioners to be interviewed; indeed, in this work we wanted to reach those industrial software architects which have been somehow exposed to the concept of architecture description language. So, discovering new ALs or understanding what an AL is not part of our study.

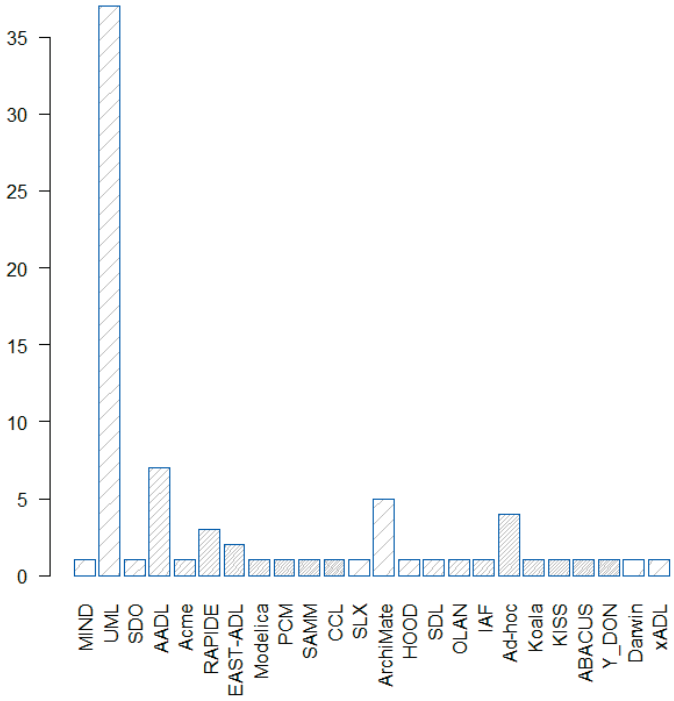

Fig. 10. ALs used by the interviewed population

15. MIND is an implementation of the Fractal component model for the development of embedded software - http://mind.ow2.org/, UML includes also UML profiles, SDO (Service Data Object) is a specification for a programming model that unifies data programming across data source types, Modelica is an object-oriented and equation based language - https://www.modelica.org/, PCM is the Palladio Component Model, SAMM is the Q-ImPrESS Service Architecture Model, CCL is the Construction and Composition Language - http://www.sei.cmu.edu/predictability/tools/ccl/, SLX is a language for discrete-event simulation - http://www.wolverinesoftware.com/, HOOD is a method of hierarchical decomposition of the design into software units, SDL stands for Specification and Description Language - http://www.itu.int/ITU-T/studygroups/com10/languages/Z.100_1199.pdf, IAF stands for Capgemini's Integrated Architecture Framework, Ad-hoc stands for ad-hoc and in-house languages, KISS is an object-oriented systems development method, ABACUS is a suite of 8 products - http://www.avolution.com.au/products.html,Y_DONstandsforYOURDONDFD.

What is relevant is that the three top-ranked ALs used in industry, that are AADL, ARCHIMATE, and industrial ad-hoc notations, sum up to 41% among all the ALs. This leads us to conclude that, based on our study population, the top used ALs in industry are those introduced by industry.

An impact study might be performed in order to understand how ALs introduced by the academia have laid the foundations for industrial ALs. For example, as mentioned in the introduction, AL research of the 90's directly influenced the definition of UML 2.0, and AADL [24] was influenced by both Meta-H [25] and Acme [23].

Moreover, a possible interpretation of Figure 10 is as follows: even though software architects have a controvert nature (i.e., ALs need to be simple and intuitive enough to communicate the right message to the stakeholders involved in the architecting phase, but they shall also enable formality so to drive analysis and other automated tasks), if they have to choose, they prefer a simple and intuitive language, well supported by tools, with respect to a language with strong formality.

6.2 Results Analysis

This section reports some interesting results that can be considered orthogonal to the main findings reported in Section 5. For the purpose of this section, we made a horizontal analysis of the various questions, and identified some perspectives of interest, that are discussed below.

Need of Empirical results: what emerged through different answers is the need of empirical results to be used to drive practitioners' decisions. This need has been clearly expressed by one of the participants, that reported that:

> *"ALs known to us are usually slightly detached from day to day development. The applicability for large-scale systems is unclear. There are limited success stories from industry for the use of ALs known to us. The overhead for a more formal specification of an architecture must bear significant benefits, which are currently not clear for us for the existing ALs (cost-benefit analysis necessary)"*

Analysis: what emerged very clearly is that practitioners do analyze software architecture descriptions, but they are unsatisfied with current ALs and tools. Analysis is considered one of the most important needs associated to ALs. However, AL and tool limitations force practitioners to extend the ALs they use, or to perform manual analysis. Extra-functional analysis represents the most commonly performed analysis. However, ALs expressiveness for extra-functional properties is considered insufficient.

Multiple Views: practitioners use multiple views and others would like to do so. Different types of views are utilized by practitioners, being structural, behavioral, and physical the most frequent ones. Multi-view consistency checking is applied by half of the practitioners making use of multiple views. However, ALs have to be improved for a better management of multiple views (43% of AL extensions and customizations are operated to add new viewpoints), and additional tool features are required for cross-view consistency.

ROI: practitioners not using ALs commented that they do not, since the learning curve is perceived as too steep while the perceived ROI (Return On Investment) is too little. Researchers could work on this issue by (i) simplifying the effort needed to begin using ALs, (ii) providing clear guidelines about the usage of specific ALs, and (iii) providing a more concrete evidence about the usefulness of ALs, for example by means of experimental results.

Needs and Limitations: as remarked throughout this paper, design, communication, and analysis are the most important needs recognized by practitioners. While practitioners are generally satisfied by the provided features for designing the architecture of the system, they are not satisfied by the communication and (extra-functional) analysis abilities of current ALs. ALs need to overcome limitations related to multi-view modeling, cross-view checking, and analysis. As far as ALs tool support is concerned, many features are missing in current tools, such as the support for analysis, for multiple views management, graphical representation and improved usability.

Code Generation: it came as a surprise that there is limited needs for code generation. While code generation is considered an interesting feature, it is rarely used. It is also not considered a very useful feature for future projects. The reasons provided by respondents are very different, among them we can quote the following one:

> *"architecture description [is] on a too high level"*
>
> *"[code generation] Requires too much details in the descriptions"*
>
> *"[code generation] was not the intended goal of the architecture definition! Architecture did have other focus"*

Version Control: A relevant amount of respondents specifically mentioned that they use version control systems. However, those features are typically not provided by existing ALs, and are implemented through external tools like Topcased, CVS/SVN, or Eclipse.

6.3 A comparison with previous studies

Both foundational works on SA [1] and [45] consider (automated) *analysis* as a primary purpose in SA. As quoted from the former work:

> *"... the primary purpose of specifications is to provide automated analysis of the documents and to expose various kinds of problems that would otherwise go undetected..."*.

As already discussed above, analysis is still considered a very important need and is very much practiced by

practitioners. Going deeper, Perry and Wolf in [1] identify the types of analysis they considered fundamental to be supported. They are:

> *"consistency of architectural style constraints; satisfaction of architectural styles by an architecture; consistency of architectural constraints; satisfaction of the architecture by the design; establishment of dependencies between architecture and design, and architecture and requirements; and determination of the implications of changes in architecture or architectural style on design and requirements, and vice versa."*.

From our study emerges that some features in an AL are still considered important, such as requirements analysis, dependence and change impact analysis, and the alignment between SA descriptions and its implementation. The common approach to solving these problems is to use a formal language to provide semantic support. They cater for design consistency and compatibility checking (strongly advocated in [30]), but they are not easy to use. The respondents of this study point out that ALs need to be intuitive and easy to use, and architects must be able to use ALs to communicate ideas. These new extra-functional requirements of an AL are in conflict with what the current AL approach can offer. As such, researchers must also consider extra-functional features of ALs to facilitate their adoption.

Other related studies, briefly discussed in Section 1.2, have analyzed AL limitations, strengths and needs. By comparing their results to our current results, we want to highlight that the usability of ALs is an important consideration in their adoption. The important needs are summarized below:

Multi viewpoint management has long been recognized as a need as well as a limitation of existing ALs. Based on our study, multi-view modeling has become a very common practice, useful in previous and future projects. A wide range of views are used today for SA description. However, ALs and tool support is still limited: existing ALs have to be extended in order to cope with multiple views, and tools still miss multi-view consistency analysis features.

Similarly, *tool support* has been recognized as a need, and the lack of good tools for ALs has been considered a strong limitation against their adoption in industrial contexts. As for this study, the level of satisfaction on ALs tool support let us understand that a certain improvement has been done along this line. Good and mature tools exist today, as reported in the answers to questions D.18 to D.24, however, there is still space for improvement to better support practitioners' needs. An example is to support easy, yet precise, representation of an SA design.

The ability to model *domain specific concerns* is another historically recognized need. From past studies, controversial limitations can be identified: in some studies, ALs are considered too general and lacking domain specific information, in others, ALs are considered too domain specific, suitable only for specific domains, and with superimposed models of computations and restrictive assumptions. As per our study, it is surprising to learn that most of the interviewed practitioners use generic notations (C.3), and that when they are asked to chose for an AL, domain specificity does not seem to be an important driver (D.5). On the other hand, when domain specific concepts or concerns is needed, ad-hoc languages, UML profiles, or AL extensions are applied.

In the related papers, some needs are not considered as mandatory, but our respondents considered them as important. The respondents suggest that in current and future projects, ALs need to provide *collaborative work*, *extensibility and customization*, *versioning*, and *communication*. We believe that investigating these aspects in future research work is fundamental to make a leap forward in the field of software architecture modeling.

6.4 Research Questions

The survey presented in this article has been centered around two main research questions. The findings are summarized in the following.

RQ1 asked *"What are the architectural description needs of practitioners?"*. Our study results uncover that, so far, academic ALs provide features that do not fully fulfill main industrial needs. Academic researchers share common beliefs such as the need for having formal and domain-specific ALs. These needs are not echoed by the practitioners. Whilst it is important to analyze and precisely represent an SA design, ALs and associated tools must also support other needs (see Section 5). In particular, the need for ALs to be usable with the goal of supporting communication among diverse stakeholders reveals what we called the *introvert versus extrovert nature of architects role*.

RQ2 asked *"What features typically supported by existing ALs are useful (or not useful) for the software industry?"*. We challenged the survey participants on the features elicited from the state-of-the-art ALs (as discussed in Section 5.4). Features like tool support, support for iterative architecting, analysis, support for multiple architectural views, graphical syntax, versioning and well-defined semantics stand out as *definitely useful*, while others (more commonly agreed) like code forward/reverse engineering and AL interoperability are considered as *definitely not useful*. Again, this further emphasizes the need for *better* ALs centered on new features.

Understanding what software architects want in ALs can provide a direction to AL researchers, and to come up with tools that software architects may use. AL tools from industry can cater for the needs of practitioners but they lack the rigor of formal ALs. Conversely, formal ALs are not currently adopted by the industry, mainly for usability deficiencies. A bridge needs to be built to close this gap.

7 Research Validity and Limitations

Sampling. We use two sampling methods to recruit participants. Firstly, we invite industry participants who have participated in ADL development or experience, the PA group. Since this group of 15 people is close to the research community, they may be biased towards the applicability of the ADLs that they have reported. This bias is mitigated by the open questions we ask them. If they are not satisfied with the ADL and the tool, then it is unlikely that they will continue to use them in current and future projects. Because of the sampling bias towards ADL users, the interpretation of the usage patterns of ADLs should be adjusted accordingly. Secondly, we invite industry participants from our industry contacts, the IC and OT groups. In this study, we decided to use by convenience sampling as sampling method; this choice may somewhat bias our results towards those participants who are sympathetic to our purposes. However, participants came from a diverse background, they are from 16 countries and 13 industry domains. This indicates a good spread of participants' profiles to increase the confidence that our results do not suffer from relevant biases with respect to the sampling methods we use.

Survey Instruments. We used two instruments to conduct the survey: online questionnaire and interviews. This is done because of the availability of the participants. Some participants overseas are not contactable during the normal working hours and we had to resort to online questionnaire. Participants who were interviewed yield additional information because follow-up questions can be asked to clarify the answers whereas we cannot ask follow-up questions to the participants who fill in an online questionnaire. In order to mitigate this possible bias, we (i) asked participants to provide a detailed explanation of the given answers by means of a follow up open-ended question, and (ii) we then apply a coding procedure on the detailed explanation of the given answers.

Terminology. Despite that we have defined AL according to ISO/IEC/IEEE 42010 to include ADLs like Acme, Darwin, AADL and other description languages like UML, the interpretation of the terminology by the participants has not been consistent. In answering questions *c5* and *d1*, some participants said they had used UML and they considered that as ADL, other participants said that they had used UML but they did not consider that as ADL. There is clearly a different notion of whether UML is considered as ADL. In order to avoid terminological interpretation in paper writing, we mapped the questionnaire generic term ADL (used to refer to all different architectural languages), into a more organized terminology that distinguishes among AL, ADL, industry ad-hoc AL, and UML notations.

We survey existing ALs through literature search. Both ADLs and industry-based languages are grouped into architecture description language we call AL. We do not prejudge on how practitioners may use them. We try to avoid researcher biases by using open questions to understand how ALs are being used. We ask the participants to describe what, how, who, when and why they use ALs. There is a part in the survey where we list twenty features and ask participants to rank their usefulness for past and future projects. These features come from analyzing the ALs that we have studied. Whilst we think that the list covers most of the important features, it is possible that some participants may have other features they see as important. In order to find those features, we asked in an open question for any other desirable AL features.

Internal validity. This survey allows us to understand the use of ALs in the industry. The findings have agreed with the general expectations of ALs usage. Firstly, ADLs are not widely used. Respondents told us that they use it in some specific areas, and they also use other types of ALs in conjunction. They told us the reasons why, confirming our suspicion that ADLs come with high costs of training and usage. Secondly, popular ADLs are produced by industry. It appears that these ADLs are still used by the software industry. The causes of this finding agree with common expectation that the software industry creates and uses tools that are useful to them. Therefore industry's active participation in building theories and tools for ALs do matter to their acceptance. The responses to open questions in the survey have provided valuable insights to help us understand the causal relationships of ALs usage and the reasons.

8 Conclusions

This paper has presented the results of a survey of the use of architectural languages in industry. Motivated by the gap between the large (and ever increasing) number of ALs produced by academia and the ones that reach industrial adoption, we decided to investigate what are the needs of practitioners and what is their opinion of the features supported by current ALs. The survey has been performed by 48 practicing architects carefully selected to represent professionals with experience in using ALs in various countries, business domains, and roles.

Our analysis produced two overall findings and a number of more detailed results associated with clusters of questions. The findings bring two general messages for future research in-the-field: (1) the ALs used in industry are also those originated in industry, academic ALs seem not to fulfill industrial needs, even though they might have inspired the industrial ALs in some ways; (2) ALs should have features to support communication among different types of stakeholders, and individual activities like analysis.

The detailed results from the clusters of questions provide research directions for specific concerns. To mention a few, support for design is more important than support for communication or analysis; domain-specific languages seem to be less important than generic ones (but yielding a well established community of adopters);

support for requirements coverage and requirements engineering phases is crucial; simplicity, pragmatism and support for collaboration is preferred over the heavy-weight formal ALs that support analysis and code generation.

As discussed in Section 7, although we carefully designed and carried out our study to avoid biases and maximize external validity, our study is still faced with some limitations. Nevertheless, our findings and detailed results provide a reflection for future research in architectural languages to meet industry needs.

ACKNOWLEDGMENTS

We would like to thank the practitioners that invested time and effort in participating in this survey, and the members of the IFIP 2.10 Working Group on Software Architecture for providing us valuable feedbacks on the design of the questionnaire. We also acknowledge discussions with Eoin Woods (Artechra, UK), Philippe Kruchten (University of British Columbia, Canada), Rich Hilliard (consultant), and André van der Hoek (University of California, Irvine) on the extrovert versus introvert nature of architects' roles.

This work received partial support from the European Community's Seventh Framework Programme FP7 20072013 under grant agreements: number ARTEMIS-2010-1-269362 (project Presto - imProvements of industrial Real time Embedded SysTems development prOcess http://www.presto-embedded.eu) and number 257178 (project CHOReOS - Large Scale Choreographies for the Future Internet http://www.choreos.eu).

Ivano Malavolta Ivano Malavolta is a Research Fellow in Computer Science at the Computer Science Department of the University of L'Aquila, Italy. He recently got a Ph.D. degree in computer science at the University of LAquila. His research interests include Software Architecture, Mobile Applications Development, Model-Driven Engineering techniques, and Wireless Sensor Networks (WSN). Ivano is also a freelance developer and designer of Mobile and Web Applications. More information about his research and academic work can be found at http://www.di.univaq.it/malavolta, information about his professional and industrial activities are available at http://www.ivanomalavolta.com.

Patricia Lago Patricia Lago is Associate Professor at the VU University Amsterdam, the Netherlands. Her research interests are in software-oriented and service-oriented architecture, architectural knowledge management, green IT and sustainable software engineering. Lago has a PhD in Control and Computer Engineering from Politecnico di Torino. She co-edited various special issues and co-organized workshops on topics related to service-oriented and software-oriented architecture, architectural knowledge management, and green IT. She has been Program Chair of the IEEE/IFIP Working Conference on Software Architecture, and is member of IEEE and ACM, and of the International Federation for Information Processing Working Group 2.10 on Software Architecture. She is Area Editor Service Orientation for the Elsevier Journal of Systems and Software. She published over 100 peer-reviewed articles in international journals, conference proceedings, and books. She is local coordinator of the Global Software Engineering European Master Programme.

Henry Muccini Henry Muccini is an Assistant Professor at the University of LAquila in Italy. Henrys main research interests are related to the role of software architectures for producing quality software. In this direction, Henry has been investigating how software architectures can be used for the verification and validation of complex and dynamically-evolving software systems. More specifically, Henry has been researching methods for testing, analyzing, and monitoring software systems based on their software architecture, as well as approaches and languages to describe architectures to improve their testability and ability to be validated. More recently, Henry has started working on research related to the architecting of wireless sensor networks and on testing context-aware mobile applications. Henry has been co-organizing a variety of workshops at WICSA, ASE, FSE, ICSE, and CompArch, and has been recently co-chairing the Program Committee of QSIC 2012 and Euromicro SEAA 2012. Henry holds a Ph.D. in Computer Science from the University of Rome - La Sapienza, Italy. More information about his work can be found at http://www.henrymuccini.com.

Patrizio Pelliccione Dr. Patrizio Pelliccione: Laurea in Computer Science (2001), Ph.D. in Computer Science (2005) both from University of L'Aquila, Italy. From April 2005 to February 2006 Patrizio was Senior Researcher and Coordinator of the CORRECT project at the University of Luxembourg, Luxembourg. The research topics are mainly in software engineering, software architectures verification, and formal methods. He has co-authored over 70 publications in journals and international conferences and workshops in this thematic. Patrizio is founding member of SERENE - Software Engineering for REsilieNt systems - ERCIM Working Group. He has been on the PC of several workshops and conferences, and reviewer of top journals in the software engineering domain. In its research activity Patrizio collaborated with several industries such as Thales Italia, Selex Marconi telecommunications, Ericsson, Siemens, TERMA, etc. More information are available at http://www.di.univaq.it/pellicci.

Antony Tang Antony Tang is an associate professor in Swinburne University of Technologys Faculty of Information and Computer Technology. His research interests include software architecture design reasoning, software development processes, software architecture and knowledge engineering. Tang received his PhD in information technology from the Swinburne University of Technology. Hes a member of the ACM and IEEE Computer Society. Contact him at atang@swin.edu.au.

REFERENCES

[1] D. E. Perry and A. L. Wolf, "Foundations for the study of software architecture," *SIGSOFT Softw. Eng. Notes*, vol. 17, pp. 40–52, October 1992.

[2] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord, and J. Stafford, *Documenting Software Architectures: Views and Beyond (2nd Edition)*, 2nd ed. Addison-Wesley Professional, Oct. 2010.

[3] H. Muccini, A. Bertolino, and P. Inverardi, "Using Software Architecture for Code Testing," *IEEE Trans. on Software Engineering*, vol. 30, no. 3, pp. 160–171, March 2003.

[4] P. Zhang, H. Muccini, and B. Li, "A classification and comparison of model checking software architecture techniques," *Journal of Systems and Software*, vol. 83, no. 5, pp. 723–744, 2010.

[5] D. Petriu, C. Shousha, and A. Jalnapurkar, "Architecture-based performance analysis applied to a telecommunication system," *IEEE Trans. Softw. Eng.*, vol. 26, pp. 1049–1065, November 2000.

[6] P. Pelliccione, P. Inverardi, and H. Muccini, "Charmy: A framework for designing and verifying architectural specifications," *IEEE Transactions on Software Engineering*, vol. 35, pp. 325–346, 2009.

[7] Fujaba Project, http://wwwcs.uni-paderborn.de/cs /fujaba/publications/index.html, 2005, University of Paderborn, Software Engineering Group.

[8] M. Abi-Antoun, J. Aldrich, D. Garlan, B. Schmerl, N. Nahas, and T. Tseng, "Improving System Dependability by Enforcing Architectural Intent," in *Proceedings of the 2005 workshop on Architecting dependable systems*, ser. WADS '05. New York, NY, USA: ACM, 2005, pp. 1–7.

[9] J. Bosch, *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach*, 1st ed. Addison-Wesley Professional, May 2000.

[10] G. H. Fairbanks, *Just Enough Software Architecture: A Risk-Driven Approach*, 1st ed. Marshall and Brainerd, August 2010.

[11] E. R. Poort and H. van Vliet, "Architecting as a risk- and cost management discipline," in *Proceedings of the 2011 Ninth Working IEEE/IFIP Conference on Software Architecture*, ser. WICSA '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 2–11.

[12] N. Medvidovic, E. M. Dashofy, and R. N. Taylor, "Moving architectural description from under the technology lamppost," *Information and Software Technology*, vol. 49, pp. 12–31, 2007.

[13] N. Medvidovic, D. S. Rosenblum, D. F. Redmiles, and J. E. Robbins, "Modeling Software Architectures in the Unified Modeling Language," *ACM Trans. on Soft. Eng. and Methodology (TOSEM)*, vol. 11, no. 1, 2002.

[14] D. Garlan and A. Kompanek, "Reconciling the needs of architectural description with object-modeling notations," in *UML 2000*, 2000.

[15] E. M. Dashofy, A. van der Hoek, and R. N. Taylor, "An infrastructure for the rapid development of XML-based architecture description languages," in *ICSE '02*. New York, NY, USA: ACM Press, 2002, pp. 266–276.

[16] N. Medvidovic and R. N. Taylor, "A Classification and Comparison Framework for Software Architecture Description Languages," *IEEE Transactions on Software Engineering*, vol. 26, no. 1, January 2000.

[17] E. Woods and R. Hilliard, "Architecture description languages in practice session report," in *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 243–246.

[18] E. Woods, "Architecture description languages and information systems architects: Never the twain shall meet?" Artechra White paper, 2005.

[19] R. K. Pandey, "Architectural description languages (adls) vs uml: a review," *SIGSOFT Softw. Eng. Notes*, vol. 35, pp. 1–5, May 2010.

[20] R. Hilliard and T. Rice, "Expressiveness in architecture description languages," in *Proceedings of the third international workshop on Software architecture*, ser. ISAW '98. New York, NY, USA: ACM, 1998, pp. 65–68.

[21] P. C. Clements, "A survey of architecture description languages," in *Proceedings of the 8th International Workshop on Software Specification and Design*, ser. IWSSD '96. Washington, DC, USA: IEEE Computer Society, 1996, pp. 16–.

[22] B. Kitchenham and S. L. Pfleeger, "Principles of survey research: part 5: populations and samples," *SIGSOFT Softw. Eng. Notes*, vol. 27, pp. 17–20, September 2002.

[23] "Acme," http://www-2.cs.cmu.edu/∼acme/, Since: 1998, Carnegie Mellon University.

[24] H. P. Feiler, B. Lewis, and S. Vestal, "The SAE Architecture Analysis and Design Language (AADL) Standard," in *IEEE RTAS Workshop*, 2003.

[25] P. Binns, M. Englehart, M. Jackson, and S. Vestal, "Domain-specific software architectures for guidance, navigation and control," *International Journal of Software Engineering and Knowledge Engineering*, vol. 6, no. 2, pp. 201–227, 1996.

[26] S. T. Redwine, Jr. and W. E. Riddle, "Software technology maturation," in *Proceedings of the 8th international conference on Software engineering*, ser. ICSE '85. Los Alamitos, CA, USA: IEEE Computer Society Press, 1985, pp. 189–200.

[27] M. Shaw, "The coming-of-age of software architecture research," in *Proceedings of the 23rd International Conference on Software Engineering*, ser. ICSE '01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 656–.

[28] D. Garlan, R. T. Monroe, and D. Wile, "Acme: Architectural description of component-based systems," in *Foundations of Component-Based Systems*. Cambridge University Press, 2000, pp. 47–68.

[29] M. Pinto, L. Fuentes, and J. M. Troya, "DAOP-ADL: an architecture description language for dynamic component and aspect-based development," in *GPCE'03*, 2003, pp. 118–137.

[30] R. Allen and D. Garlan, "A formal basis for architectural connection," *ACM Trans. Softw. Eng. Methodol.*, vol. 6, pp. 213–249, July 1997.

[31] D. Cassou, B. Bertran, N. Loriant, and C. Consel, "A generative programming approach to developing pervasive computing systems," in *Proceedings of the eighth international conference on Generative programming and component engineering*, ser. GPCE '09. New York, NY, USA: ACM, 2009, pp. 137–146.

[32] R. van Ommering, F. van der Linden, J. Kramer, and J. Magee, "The Koala Component Model for Consumer Electronics Software," *Computer*, vol. 33, no. 3, pp. 78–85, 2000.

[33] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.-B. Stefani, "The fractal component model and its support in java: Experiences with auto-adaptive and reconfigurable systems," *Softw. Pract. Exper.*, vol. 36, pp. 1257–1284, September 2006.

[34] J. E. Robbins, N. Medvidovic, D. F. Redmiles, and D. S.Rosenblum, "Integrating architecture description languages with a standard design method," in *Proc. 20th Int. Conf. on Software Engineering*, 1998.

[35] P. Kruchten, "Architectural Blueprints - The "4+1" View Model of Software Architecture," *IEEE Software*, vol. 12, no. 6, pp. 42–50, November 1995.

[36] C. Hofmeister, R. Nord, and D. Soni, *Applied Software Architecture*. Addison-Wesley, 1998.

[37] H. Gomaa and D. Wijesekera, "The Role of UML, OCL and ADLs in Software Architecture," in *Proc. Of the Workshop on Describing Software Architecture with UML*, Toronto, Canada, 2001.

[38] M. M. Kandé, V. Crettaz, A. Strohmeier, and S. Sendall, "Bridging the gap between IEEE 1471, Architecture Description Languages and UML," *Software and System Modeling*, vol. 2, pp. 98–112, 2002.

[39] B. Selic, "On modeling architectural structures with uml," in *Proc. of the Workshop on Describing Software Architecture with UML, in ICSE 2001*, Toronto, Canada, 2001.

[40] M. Goulo and F. Abreu, "Bridging the gap between Acme and UML for CBD," in *Specification and Verification of Component-Based Systems.*, 2003.

[41] S. Roh, K. Kim, and T. Jeon, "Architecture Modeling Language based on UML2.0," in *Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC'04)*, 2004.

[42] J. Ivers, P. Clements, D. Garlan, R. Nord, B. Schmerl, and J. R. O. Silva, "Documenting Component and Connector Views with UML 2.0," CMU, SEI, Tech. Rep. CMU/SEI-2004-TR-008, 2004.

[43] J. E. Perez-Martinez and A. Sierra-Alonso, "UML 1.4 versus UML 2.0 as languages to describe Software Architectures," in *Proc. EWSA 2004. LNCS n. 3047*, 2004.

[44] D. Garlan and M. Shaw, "An Introduction to Software Architecture," in *Advances in Soft. Eng. and Know. Eng.*, vol. 2, 1994, pp. 1–39.

[45] M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*. Prentice-Hall, Englewood Cliffs, 1996.

[46] E. M. Dashofy, A. V. der Hoek, and R. N. Taylor, "A Highly-Extensible, XML-Based Architecture Description Language," in *WICSA'01*, 2001.

[47] J. Magee, J. Kramer, and D. Giannakopoulou, "Behaviour Analysis of Software Architectures," in *First Working IFIP Conference on Software Architecture, WICSA1*, 1999.

[48] D. Garlan, "Formal Modeling and Analysis of Software Architecture: Components, Connectors, and Events," in *Formal Methods for Software Architectures*. Lecture Note in Computer Science, 2804, 2003, pp. 1–24.

[49] W. Qin and S. Malik, "Architecture description languages for retargetable compilation," in *in The Compiler Design Handbook: Optimizations & Machine Code Generation*. CRC Press, 2002, pp. 535–564.

[50] ISO/IEC/IEEE, "42010:2011 Systems and software engineering – Architecture description," 2011.

[51] D. Di Ruscio, I. Malavolta, H. Muccini, P. Pelliccione, and A. Pierantonio, "Developing next generation ADLs through MDE techniques," in *ICSE 2010*, 2010.

[52] M. B. Miles and M. Huberman, *Qualitative Data Analysis: An Expanded Sourcebook(2nd Edition)*, 2nd ed. Sage Publications, Inc, 1994.

[53] A. N. Oppenheim, *Questionnaire Design, Interviewing and Attitude Measurement*. Continuum, Oct. 2000.

[54] M. Kasunic, "Designing an Effective Survey," Handbook CMU/SEI-2005-HB-004, Software Engineering Institute, Carnegie Mellon University, Tech. Rep., 2005. [Online]. Available: www.sei.cmu.edu/pub/documents/05.reports/pdf/05hb004.pdf

[55] M. M. Lankhorst, H. A. Proper, and H. Jonkers, "The Anatomy of the ArchiMate Language," *IJISMD*, vol. 1, no. 1, pp. 1–32, 2010.

[56] D. C. Luckham, "Rapide: A language and toolset for simulation of distributed systems by partial orderings of events." Stanford University, Stanford, CA, USA, Tech. Rep., 1996.

[57] P. Cuenot, P. Frey, R. Johansson, H. Lönn, Y. Papadopoulos, M.-O. Reiser, A. Sandberg, D. Servat, R. T. Kolagari, M. Törngren, and M. Weber, "The east-adl architecture description language for automotive embedded software," in *Proceedings of the 2007 International Dagstuhl conference on Model-based engineering of embedded real-time systems*, ser. MBEERTS'07. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 297–307.

[58] P. Lago, H. Muccini, and H. van Vliet, "A scoped approach to traceability management," *J. Syst. Softw.*, vol. 82, pp. 168–182, January 2009.

[59] P. Kruchten, "What do software architects really do?" *Journal of Systems and Software*, vol. 81, no. 12, pp. 2413 – 2416, 2008.

[60] J. F. Hoorn, R. Farenhorst, P. Lago, and H. van Vliet, "The lonesome architect," *Journal of Systems and Software*, vol. 84, no. 9, pp. 1424 – 1435, 2011.

[61] D. Hendricksen, *12 Essential Skills for Software Architects*. Addison-Wesley Professional, 2011.

[62] L. Capretz and F. Ahmed, "Making sense of software development and personality types," *IT professional*, vol. 12, no. 1, pp. 6–13, 2010.

[63] P. Kruchten, "What do software architects really do?" *Journal of Systems and Software*, vol. 81, no. 12, pp. 2413–2416, 2008.

[64] N. Mangano and A. van der Hoek, "The design and evaluation of a tool to support software designers at the whiteboard," *Automated Software Engineering*, pp. 1–41, 2012.