ADDRESSING ENERGY EFFICIENCY IN SYSTEM DESIGN: A JOURNEY FROM ARCHITECTURE TO OPERATION

EOIN WOODS

A thesis submitted in partial fulfilment of the requirements of the University of East London for the degree of Doctor of Philosophy

December 2018

Abstract

Digital-transformation initiatives have led to major efficiencies and cost savings but at the cost of consuming nearly 10 percent of the world's electricity. Energy consumption research has increased datacentre, network, and hardware efficiency, but a neglected aspect of energy research has been the energy consumption of the software applications that underpin digital transformation. To date, software architects have lacked the knowledge, guidance, and tools to allow them to understand the energy properties of their systems.

The research reported in this thesis begins to address this situation by developing practical knowledge, techniques, and tools to allow software architects to play their part in controlling the energy consumption of our modern digital world.

The work commences with an investigation into formal architectural description languages, through a literature review and a case study, resulting in two research contributions, namely a comprehensive systematic survey of architecture description languages from 1991 to 2015, and a case study of practical ADL use at scale in industry.

The second part of the research investigates how to assist architects in prioritising energy efficiency through a study of how experienced architects focus their attention for maximum effectiveness, which leads to the development of a model to guide architecture practitioners, which is validated and refined through a large survey of practising software architects. The research contribution is a refined and validated model for architectural effort prioritisation.

The third aspect of the research examines the energy-related guidance available to architects and having found little generally applicable advice, analyses a significant industrial case study to understand how leading-edge practitioners addressed energy efficiency, contributing a set of three energy-related architectural principles, which can be used to guide architects in improving application energy efficiency.

Finally, we consider the practical problem of understanding the runtime energy properties of a system, and designed a novel approach to estimate the energy consumption of execution scenarios via application execution tracing and a cost-based energy model. We created a proof of concept implementation of the approach and validated its consistency and correctness through practical testing. The contribution of this work was twofold, namely the design of a practical system for allocating energy to application execution scenarios, and a tested, open-source, proof-of-concept implementation of the system.

Hence, the result of this work is six distinct contributions to knowledge in the area of ADLs (the survey and practical case study), architectural practice (the prioritisation model and the architectural principles for energy efficiency) and application energy efficiency (the design of the energy allocation system and the proof-of-concept implementation), which collectively can help architects to treat energy efficiency as a first class architectural concern in their work

20181218.3

Contents

List of Figures

List of Tables

Acknowledgements

Every PhD student thanks their supervisor, but in this case it is particularly important. Without the guidance, support and friendship of Dr Rabih Bashroush, this research would not have been started, let alone completed. Thank you, Rabih. You've been a strong, calm and knowledgeable guide through this complex, and at times intimidating, process.

I'd also like to thank the many members of UEL's Computer Science and Informatics group who have encouraged, advised or guided me during the process including my second supervisor Dr Syed Islam, Professor Allan Brimicombe, Dr Andres Baravalle, Dr Aloysius Edoh, Dr Paolo Falcarin and Mr Mike Kretsis. Charlotte Forbes has also provided invaluable guidance and support for graduate students in the department.

The other group of people who have knowingly or unknowingly inspired, supported or encouraged me during this long process are those I have collaborated with when developing my ideas of software architecture. Special mention and thanks is due to my long-term collaborator Nick Rozanski, who has been my co-author, constructive critic and friend during everything we've done together, most notably our book. The wider software architecture research community has also been a friendly and supportive place in which to develop ideas, particularly the communities who attend the ECSA and WICSA conferences. I fear that I will undoubtedly forget someone, but I am most grateful for the friendship, guidance and support at different times of Professor Philippe Kruchten, whose work has inspired me throughout my career, Dr Eltjo Poort of CGI whose own long journey through a part-time PhD encouraged me to keep progressing mine, Dr John Klein, Dr Grace Lewis, Dr Rod Nord and Dr Ipek Ozkaya of the SEI who have encouraged me repeatedly along the way, Professors Hans van Vliet and Patricia Lago from Vrije Universiteit, Amsterdam, Dr George Fairbanks of Google and Michael Keeling from IBM who always inspire me to think more deeply about things that I think I understand, and Cris Zamora for many thought-provoking conversations.

I'd also like to thank my colleagues from Endava, who create a collegiate, friendly and supportive environment in which to work, particularly my managers John Cotterell our CEO, Rob Machin our COO, and all of my colleagues in our delivery units, and our strategy and operations groups.

Finally, but most importantly, I'd like to thank my family who have encouraged me throughout, despite never being quite clear what I'm doing, or why on earth I am doing a PhD in my 40s. Thank you, Lynda and Katherine, for putting up with the long periods while I've been stuck in my study during evenings and weekends. Thank you, Mum and Dad, for your continual support and belief in me. Thank you, Nick, Marcia, Shiann and Kallum, for all you bring our family and reminding me to step away from the research sometimes when more important things need some attention!

Eoin Woods
September 2018

I dedicate this work to my family. My wife Lynda and daughter Katherine, who make it all worthwhile. My parents, Desmond and Anne, who have supported me in so many ways and to whom I owe so much. And Nick, Marcia, Shiann and Kalum, who bring so much love and light to our family.

Chapter 1

Introduction

1.1 Context and Motivation

The IT industry and the penetration of IT services into the life of most people has entered a new era. The number of devices connected to the internet has been growing steadily. Cisco estimated that by 2008 the number of internet-connected devices had exceeded the number of people in the world and that by 2011 the internet usage of 20 typical households was generating more internet traffic than the world's entire internet use in 2008 [49]. Partly due to this network capacity, we are witnessing a parallel growth in data, driven by more affordable storage systems and new applications of the internet including mobile applications, IoT, social media, and smart cities. This combination of data and connectivity has resulted in so-called "digital transformation" in many industries, leading to a huge ecosystem of software applications, from business analytics of customer behaviour to mobile apps that can allow a farmer to monitor the climatic and ground conditions in their fields in real time through local sensor systems. These new applications rely on being internet-connected, which results in constantly increasing demand for private and "cloud" data centre capacity. This is why large Tech companies and major enterprises are continuously expanding their computing capabilities.

Currently, data centres consume a substantial amount of energy and are thought to produce more greenhouse gas emissions than the entire aviation sector. In 2013, data centres in the U.S. alone consumed an estimated 91 billion kWh of electricity, and this is

expected to rise to 140 billion kWh recent survey of data centre managers showed that energy efficiency is seen as their single biggest challenge [38]. This is particularly important for colocation and managed service data centre providers who are based within or close to large cities where grid power availability is limited.

So, can the data centre community evolve to cope with this ever-increasing demand and support the world's relentless digital transformation? We cannot be sure, but it is clear that the challenges that it poses are widely understood. A large number of industrial, governmental and academic research programmes [14, 48, 69, 103, 171] have been investigating and continue to explore topics that can help, from new cooling technologies to more energy-efficient servers and building designs in the physical domain, from runtime workload consolidation to energy consumption monitoring in the software domain and even economic models for balancing system quality properties and power consumption from cross-disciplinary research. However, the software architecture community has been slower to recognise their potential contribution and to mobilise to meet this challenge. Addressing energy efficiency at the architecture level is still far from being mainstream. Can we continue designing systems without any consideration of their energy and power efficiency and let others worry about running them in an energy-efficient way? Should energy efficiency be a bolt-on system property or a quality attribute that is addressed at design time?

We believe that software architects may not be prioritising energy efficiency for a number of reasons.

Firstly, we currently have very little understanding of the impact of design decisions on energy efficiency or an understanding of how it affects other system qualities such as user experience, reliability, and performance. Without this knowledge, it is difficult to perform trade-off analysis to understand the benefit or cost of improving energy efficiency. Minor changes to the system design could yield substantial benefits, such as avoiding unnecessary component redundancy or eliminating low-priority housekeeping tasks that prevent equipment from entering lower-power states. However, a lack of relevant design tools and frameworks mean that it is still difficult to understand the energy characteristics of software at an architectural level, let alone understand or test the energy implications of design decisions.

Second, in order to achieve the next order of magnitude in energy efficiency, we need to think outside traditional design boundaries. This will require people from different specialisations and departments to prioritise energy efficiency as a design goal and to work together. This can often be difficult to achieve, given current organisational governance structures, with different teams sometimes having competing objectives, not to mention human dynamics and political barriers.

Finally, with the exception of mobile applications, where battery life is a visible reminder of the need for energy efficiency, energy rarely features as a high priority requirement or concern for system acquirers or end-users. On one hand, there is the problem of split incentives where operators of systems (e.g. administrators or data centre managers) do not pay the energy bill (which usually comes from the facilities budget). This means that they would see very little return from any savings made from energy efficiency. On the other hand, while energy costs can be anywhere from 25% to 60% of total data centre operating cost [170], this is often a relatively small percentage of an organisation's overall spending. So, when cost savings are required, it may be easier to achieve them by reducing cost in other areas.

We believe that this situation can be addressed by creating tools and guidance which is aimed specifically at software architects and system stakeholders, to allow them to understand the energy efficiency implications of architectural design decisions. The work reported in this thesis is a first step on the road to provide this.

1.2 Objectives and Research Questions

The objective of this research is to improve the assistance available to software architects to help them understand the impact of their work on the energy efficiency of the systems that they design. This process started in the area of architectural description languages (ADLs) because the unambiguous description of the architecture that they provide offered the potential to provide the architect with visibility of the energy implications of their design decisions.

To reach our objective, this work aims to answer four specific research questions, namely:

RQ1 What architecture description languages exist and can they be used to reason about the energy properties of a system?

RQ2 How can architects prioritise energy efficiency as an architectural concern?

RQ3 What design guidelines can we provide to assist architects to improve the energy efficiency of their systems?

RQ4 How can we make architects aware of the runtime energy characteristics of their systems?

Answering these questions has involved the investigation of the use of ADLs for large-scale architectural description, the identification of practical advice for how architects can focus attention on critical topics (such as energy efficiency), the identification of design principles to guide energy-efficient architectures, and the creation of a practical approach for estimating the energy usage of request processing in distributed applications.

1.3 Research Methodology

The research reported in this thesis comprises four areas of investigation, aligned with the four research questions defined above. Each area of investigation has utilised different combinations of research techniques, which are outlined below. Fuller discussion of the research methdology for each piece of work is provided in the relevant chapters of the thesis.

1.3.1 Architectural Description Languages Investigation

The ADL investigation work began with a systematic literature review [88], which was presented in Section 2.1, with the goal of systematically identifying, analysing and understanding the work that had been done to date in creating and applying architectural description languages.

The second stage of this research was to undertake a practical exercise to apply an ADL from the research results to the description of a large industrial system to understand how practical and effective this would be.

This research enabled us to answer research question RQ1.

1.3.2 Study of Prioritising Architectural Effort

Our study of architectural effort prioritisation began with a narrative literature review to survey the state of research in this area [22], which is described in Section 2.2. The literature review did not identify a satisfactory approach to the problem of prioritising architectural effort, so we undertook the process of creating one. This a was a four-stage process involving surveys, model building and validation. Full details are provided in Chapter 4, but in summary, the four stages of research were:

Stage 1 gathering primary data using semi-structured interviews with practitioners, using a written introduction to the question we wanted to answer and then some specific questions to illustrate our area of interest.

Stage 2 analysis of the primary data and creation of a preliminary model through a simple application of Grounded Theory [29].

Stage 3 validation of the preliminary model via a structured online questionnaire [61], completed by practitioners in relevant architecture roles (primarily software, solution and enterprise architects).

Stage 4 analysis of the validation data and refinement of the preliminary model into a final, validated model.

To ensure practitioner involvement in the research, we used the LinkedIn and Twitter social media networks to publicise and engage with architecture practitioners and to report preliminary results.

The result of this research was a validated model to guide the prioritisation of architectural effort, which allowed us to answer research question RQ2.

1.3.3 Development of Energy Efficiency Design Principles

This aspect of the research was undertaken to answer research question RQ3 by attempting to identify energy efficiency related design guidance for practicing software architects.

The work began with a narrative literature review to survey the state of research in this area, described in section Section 2.3, which identified some useful ideas in the research literature, but relatively little design guidance relevant to the software architecture practitioner.

To identify some initial design guidance we decided to study an industrial situation that had been successful in reducing energy consumption and found a case study from a major internet firm that had managed to reduce energy consumption considerably [47]. We analysed this scenario and synthesised and captured key principles from it.

This enabled us to answer research question RQ3.

1.3.4 Design and Implementation of Application Energy Monitoring

This section of the research aimed to answer research question RQ4 through the design and implementation of a proof-of-concept implementation of a technical solution to provide architects with visibility of the energy consumption of their systems.

The work began with a narrative literature review to survey the state of research in this area, reported in Section 2.4. This exercise discovered that some application energy measurement systems have been designed and reported in the literature, but most are not available for general use and the solutions proposed all had some significant limitations when their application to industrial systems was considered.

This work designed a novel approach to capturing representative energy usage for application execution scenarios, that allocates estimated server energy usage to the applications running on that server, rather than trying to calculate an absolute energy consumption for each. The approach was implemented using modern mainstream technology (such as Linux, Docker and Java) and then validated using a set of practical, realistic tests, to validate its internal consistency and external correctness with respect to its runtime environment.

The design is presented in a technology independent way in Chapter 6, the concrete implementation is described in Chapter 7 and the validation process is described in Chapter 8.

A fuller description of the research methodology for the entire exercise, described in the three chapters, is presented in Section 6.3 in Chapter 6.

The competion of this work enabled us to answer research question RQ4.

1.4 Contribution

The research described in this thesis has contributed elements to software architecture research and energy efficiency research and in particular, has contributed to bringing the two closer together. The specific research contributions in this thesis are as follows:

1. **A comprehensive systematic survey of architecture description languages**. This literature review surveyed all ADLs created from 1991 to 2015, reviewing 135 potential languages, with 51 of them being confirmed as architectural description languages, according to our criteria. A detailed characteristation of the languages and the field were produced from this analysis.

2. **A case study of practical ADL use at scale in industry**. Having surveyed the ADLs, we considered how to apply them to a real problem for a real stakeholder for a large industrial system. This experience led to the academic ADLs being abandoned and a simple, lightweight, specialised notation to describe the architectural style used in the system being used instead. A set of lessons learned and constructive suggestions for future ADL research were derived from the experience.

3. **A model for architectural effort prioritisation**. We interviewed expert practitioners and created a model for effort prioritisation based on common approaches that they (unknowingly) shared. This model was then validated and refined using the results of a survey of 84 software architecture practitioners from across the world.

4. **Principles for energy-efficient architectural design**. Having found no energy specific architecture principles and relatively few generally applicable energy efficiency tactics in the literature, we identified a small set of principles from a successful industrial case study that reduced energy consumption for application services.

5. **A system for allocating energy to application scenarios**. Our literature review revealed that a number of application energy estimation systems had been proposed and prototyped. However, all of these systems measure the energy consumption of operating system processes, which makes the information of limited immediate value to the application architect. Our contribution has been to design and create a proof-of-concept implementation of a system which estimates the energy consumption of execution scenarios through the application (using application tracing), so providing the architect with significantly more insight into the effect of their architectural decisions. As far as we know, this scenario based energy usage calculation has not been attempted before in prior research.

6. **Open source implementation of Apollo**. The proof-of-concept implemenation of the system for allocating energy to application scenarios is available as open source

software, under an Apache license, from the well known Github source repository site at `http://github.com/eoinwoods/apolloenergy`.

Much of the work reported here has been previously published in conference proceedings and journals. The publications arising from this work are listed below:

Woods, Eoin, and Bashroush, Rabih. "Using an Architecture Description Language to Model a Large-Scale Information System - An Industrial Experience Report." In *Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012 Joint Working IEEE/IFIP Conference on*. IEEE, 2012.

Woods, Eoin, and Bashroush, Rabih. "Modelling Large-Scale Information Systems Using ADLs - An Industrial Experience Report." *Journal of Systems and Software* 99 (2015).

Bashroush, Rabih, Woods, Eoin, and Noureddine, Adel. "Data Center Energy Demand: What Got Us Here Won't Get Us There." *IEEE Software* 33, no. 2 (2016).

Bashroush, Rabih, and Woods, Eoin. "Architectural Principles for Energy-Aware Internet-Scale Applications." *IEEE Software* 34, no. 3 (2017).

Woods, Eoin, and Bashroush, Rabih. "A Model for Prioritization of Software Architecture Effort." *European Conference on Software Architecture*. Springer, Cham, 2017.

Woods, Eoin, and Bashroush, Rabih. "How Software Architects Focus Their Attention." *Journal of Systems and Software*. Submitted.

1.5 Structure of Thesis

This thesis is structured into 9 chapters, each presenting a specific aspect of the research work. The structure of the thesis is illustrated by the diagram in Figure 1.1.

Chapter 1 is this introductory chapter, setting the motivation and context for the work, defining the research questions, explaining the research approach and explaining the structure of the thesis.

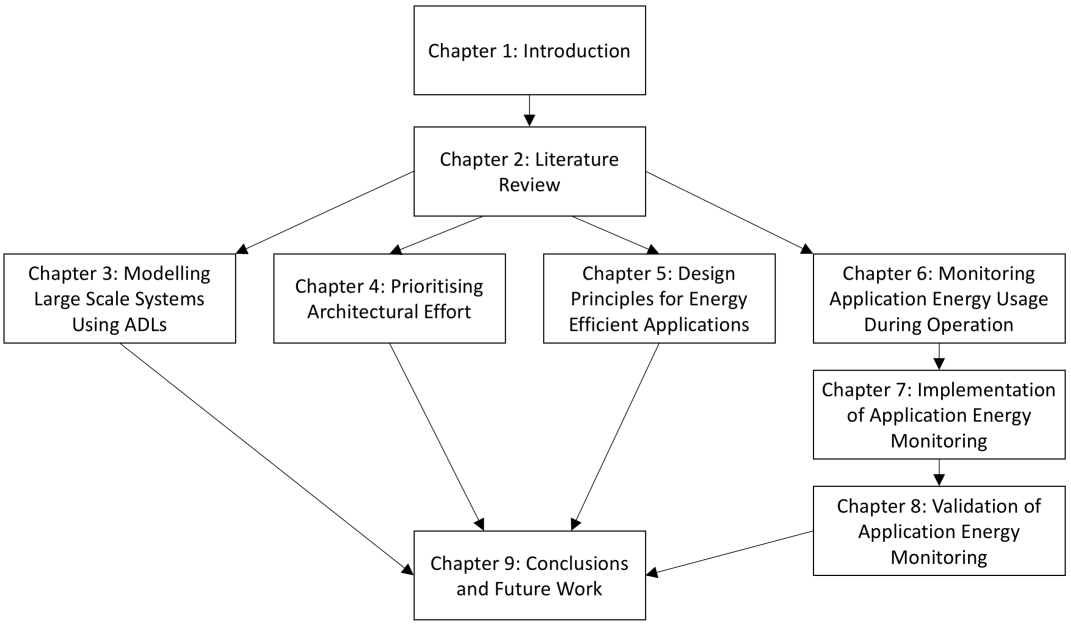


FIGURE 1.1: Structure of the Thesis

Chapter 2 contains a literature review, structured into four parts, exploring the research literature in the areas of architectural description languages (ADLs), how architects should prioritise their focus for maximum effectiveness, design guidelines for energy efficiency and runtime energy consumption for IT systems.

Chapter 3 explores research question RQ1 and discusses how ADLs can be used to describe large-scale software systems and presents a significant industrial case study that explored how effective this was in practice.

Chapter 4 investigates research question RQ2 and explores the area of prioritisation of work from an architect's perspective, asking how architects prioritise their time for maximum effectiveness and presenting the results of an industrial survey into the approaches used by experienced practitioners and a validated model to guide architects, based on the insights from the survey.

Chapter 5 explores energy related design principles to answer research question RQ3 and presents a small set of heuristic design principles for designing energy-efficient software applications, derived from the experience of an industrial case study in reducing energy consumption through architectural change.

Chapter 6 addresses the fundamentals of research question RQ4 by asking how application energy usage can be monitored and estimated during the operation of a system and presents a theoretical model for solving this problem.

Chapter 7 addresses the practical aspects of research question RQ4 and presents a proof-of-concept implementation of the energy estimation model, specialised for estimating energy usage of a group of microservices processing incoming requests.

Chapter 8 validates the energy estimation approach as part of answering research question RQ4 and presents the work performed to validate the energy estimation technology. This involved running a number of carefully controlled test scenarios and using the tool to estimate their energy usage, while also deriving the same estimation through a separate, independent technique, using these secondary estimations to validate the outputs of the model and the tool.

Chapter 9 summarises the research work, draws conclusions from it to answer the research questions and discusses possible future research work in each area.

Chapter 2

Literature Review

2.1 Architectural Description Languages

Software architecture has been an active research field since the mid-1990s and one of its recurring research topics has been how to create, communicate and maintain effective architectural descriptions. A range of techniques have been proposed over the years, but a recurrent theme is the idea of a specialised architectural description language (or "ADL").

The first ADLs appeared in the early 1990s and 10 significant languages from the first 10 years of research were the subject of a seminal literature review by Medvidovic and Taylor in January 2000 [114]. Perhaps inspired by this work, there has been an explosion in the number of ADLs created since that time, but based on our industrial experience and reading of the research literature, there has been little indication of a corresponding increase in their use in industry.

We are interested in how to assist architects to consider the energy properties of their systems as a first-class architectural concern and this led us to ask whether we could use an ADL as the basis of any solution that we designed. This led to our first research question namely, *RQ1 - What ADLs exist and can they be used to reason about the energy properties of a system?*. Our goal was to understand the possible applicability of existing architectural description languages to our problem and assess the degree to which the languages that have been created would be useful in industrial practice.

As part of answering this question, we undertook to identify and review the relevant research literature that has been created over twenty-five years of research in the area. Our aim was to characterise the ADLs that have been developed and consider their possible applicability to addressing the energy properties of industrial software applications.

2.1.1 Supplemental Research Questions

As soon as we started to perform the initial investigation into architectural description languages, we realised that it has become a complex and multi-faceted field. Hence, to approach the review in a structured way, we posed a number of research questions specific to the survey, in order to understand the characteristics of the ADLs that have been designed and their possible applicability to our problem:

ADL.RQ1 *Which architectural viewpoints does each ADL support?* It has been long understood that an architecture contains many structures, not just one. This challenge is addressed by structuring an architectural description into views defined by viewpoints [169]. Surveying the set of viewpoints supported by an ADL allows us to understand which architectural structures it can represent.

ADL.RQ2 *Does the ADL provide structuring mechanisms for large architectural descriptions?* Many academic tools and methods are only tested using small examples whereas industrial systems are often orders of magnitude larger. Our focus on the industrial application of ADLs meant that we wanted to understand which ADLs included features for structuring large architectural descriptions.

ADL.RQ3 *Does the ADL support the analysis of an architecture?* Another possible motivation for using an ADL is the ability to perform automated analysis of a machine-readable architectural description, and this could allow the ADL to provide the basis for automated energy estimation and analysis. Hence we were interested to understand which ADLs allow this and what sort of analysis could be performed.

ADL.RQ4 *Can system qualities or quality requirements be captured in the ADL?* A critical aspect of industrial software architecture work is ensuring that systems exhibit their key quality properties, so we wanted to establish what support each ADL provided to support this process.

ADL.RQ5 *Were prototype or production quality tools developed with the ADL?* It is unlikely that an ADL will be seriously applied in industry unless it has robust and user-friendly tools available to support it, so we wanted to verify the level of tool support provided with each ADL.

ADL.RQ6 *Has the ADL been applied to non-trivial problems outside the group of people who created it? (e.g. significant research projects from outside the originating group, industrial case studies or industry standards.)* A software architecture practitioner is likely to want some evidence of the effectiveness of an ADL before adopting it on a significant project. Therefore, we wanted to know whether researchers had acknowledged this barrier to adoption and had addressed it through realistic case studies or use on real projects outside the originating research group.

It is worth noting that we do not ask if the language supports first-class components because this is a prerequisite to the language being included in the study. (Our view is that languages that do not support first-class components are not architectural description languages.)

2.1.2 Research Methodology

We identified the research literature to include in the study using an electronic literature search, augmented by manual scanning of reference lists in the papers found and our own background knowledge of the field, that led us to identify additional relevant candidate literature (that for example may not have been tagged with the keywords we expected).

We began by searching a range of electronic sources for papers that included the keywords "ADL" or "architecture description language" in their title or keywords. The four primary sources we used were the ACM Digital Library (advanced search), Google Scholar, IEEEXplore and Microsoft Academic Search.

Predictably these queries returned many references, however it was clear from our existing knowledge of the field that these keyword-based searches were not returning all of the relevant ADL related literature that had been published.

To find further relevant literature we then performed an exhaustive search of Google Scholar, using the relevant keywords, which returned thousands of references which were manually scanned for relevant primary studies that we might have missed. This list contained many false positives, but these were discarded via manual inspection.

Having searched traditional literature review sources, we also performed manual searches of specific publication venues where ADL researchers were known to publish their findings, specifically the specialist conferences WICSA, QoSA, ECSA and ICSE.

Finally, we performed forward and backward reference checking on the primary studies that we had found. Search engines were used to find citations of the primary studies identified that could be of relevance to the review (forward reference checking). The reference lists of the primary studies were then checked for any potentially relevant studies missed (backward reference checking). At this point, we had 135 potential primary studies for the survey.

Throughout these search activities, we limited the dates of the studies that we included, to limiting our scope to literature published between January 1991 and May 2016. The start date was selected to be early enough to include all those ADLs in the original work [114] that inspired us to undertake this later comprehensive survey and as noted in [109] the concept of an ADL was not well defined before this point. Our literature search was concluded in May 2016, which is the reason for the end date.

To focus our efforts on the most relevant ADLs, our initial set of primary studies was filtered further to a more manageable set using the following exclusion criteria:

EC1 The ADL is a minor enhancement or minor extension to an existing ADL, or the ADL is a different version of an included ADL.

EC2 The ADL focuses on a single area of architectural analysis (e.g. Concurrency) rather than being a general-purpose description language.

EC3 There is not enough detail in the references discovered to address the study research questions.

EC4 The ADL not suitable for modelling a software-intensive system at an architectural level of concern (for example a hardware design language or source code module description language).

EC5 The primary study is not available in English or is a short paper (less than 3000 words), abstract, keynote, opinion, tutorial summary, panel discussion, technical report, presentation slides, a compilation of work or a book chapter. Book chapters were only included if they were conference or workshop proceedings (e.g., as part of the LNCS or LNBIP series) and are available through the data sources included in our review.

The result of this further selection exercise was a list of 51 ADLs to include in the survey and 84 ADLs that did not meet our inclusion criteria. A full list of the characteristics of the ADLs that met our inclusion criteria is provided in the tables in Appendix A.

2.1.3 Analysis of the Results

The first aspect of the ADLs we were interested in was the *basic information* about each and specifically the institutions who developed them, the dates when the language was first published, the application domain that they address and the breadth of the application that they have been applied to.

When considering the breadth of application of the languages, we identified five possible degrees of application of an ADL that were of interest to us, namely:

Examples meaning that the language has only been used to create characteristic examples of its use;

Experiments where it has been used to model realistic problems, but only for the purpose of investigating the language;

Case Studies meaning that it has been applied to realistic problems from outside the originating research group but by the language creators;

Research Projects where the language has been used on other research projects by researchers other than its creators; and

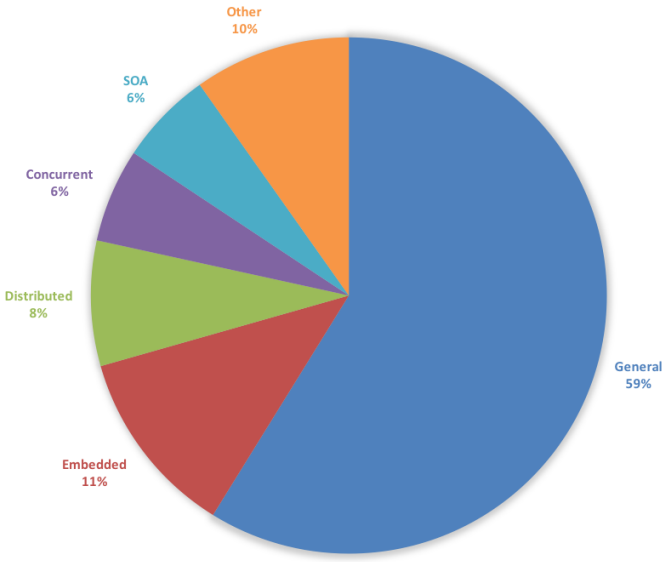


FIGURE 2.1: ADL Target Application Domain

Industrial Projects meaning that the language has been used by industrial software engineering teams on real projects (rather than industrial researchers, who would be classified as research project use).

We were obviously particularly interested in how many ADLs had been applied beyond its creating research group on other research projects, or ideally on industrial projects.

The complete data set for the ADL's basic characteristics extracted from the literature can be found in Table A.1.

Two characteristics of the ADLs we wanted to understand were the application domains that they targeted and the degree to which they had been applied.

The analysis of the intended application domain of the languages can be found in Figure 2.1. Interestingly very few of the languages were created for a specific business domain (e.g. financial analysis or industrial control systems) as over half the ADLs in the study do not explicitly target any business or technical application domain but are for general use. There are a smaller number of ADLs specialised for embedded systems, distributed systems, highly concurrent systems and SOA, along with a number of individual ADLs for niche domains such as cyber-physical systems.

The analysis of the breadth of application of the languages can be found in Figure 2.2. Unfortunately, as can be seen, less than 20% of ADLs have been used beyond the case

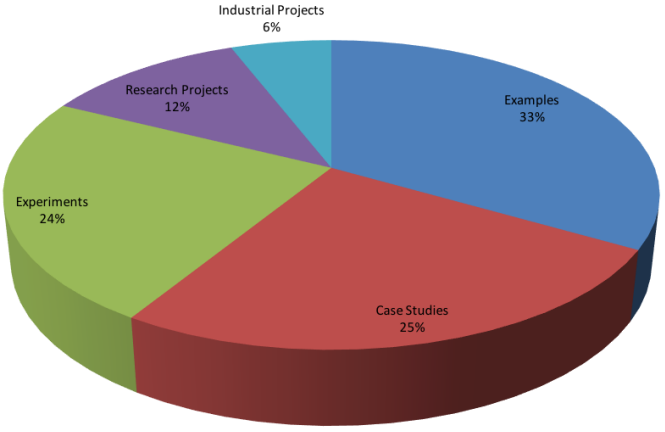


FIGURE 2.2: ADL Breadth of Application

study level to perform significant research or industrial projects, suggesting a low degree of validation and practical experience with most of the languages.

The second area of interest to us were the *architectural concepts* available in the different languages, to see if common industrial architectural concepts were in the languages or would need to be added.

The characteristics we analysed the ADLs for were the viewpoints that the ADL directly supports, the architectural concepts that they provide, whether they provide the ability to define behavioural semantics, whether they provide first-class connectors and whether they provide first-class architectural configuration constructs. We chose to focus on these architectural concepts because of their wide use in the existing research literature and their general familiarity as concepts in industrial practice.

We were particularly interested in which viewpoints each ADL could support, as industrial architectural description nearly always needs a number of views to describe it, and the views supported provide a good insight into what the language can be used for.

None of the ADL descriptions discuss a specific set of viewpoints that they define, so we analysed whether they provided effective support for the 6 viewpoints from [153] (which are Functional, Concurrency, Information, Development, Deployment and Operational). We class a language as having first-class connectors if the connector is defined separately to components and so is potentially reusable. Similarly, we consider architectural configuration to be a first-class concept if it is described separately to the architectural

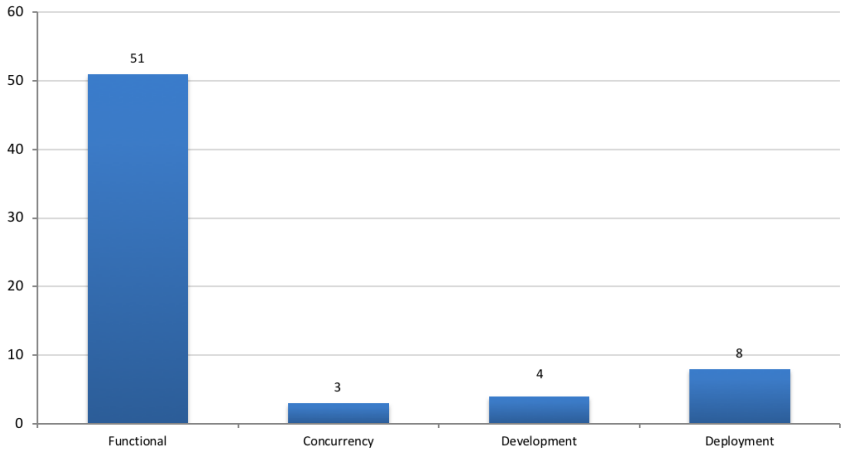


FIGURE 2.3: Viewpoints Supported by ADLs

elements and defines how they are combined, rather than being defined implicitly as part of the definition of the elements.

The complete data set for the architectural concepts available in each of the ADLs can be found in Table A.2.

The analysis of which viewpoints are supported by the different ADLs is presented in Figure 2.3.

What is immediately evident from this analysis is that most ADLs only focus on the functional view of a system (i.e. its functional components and connectors and their organisation). While clearly a key part of a system's architecture, most architects actually spend a lot of their effort working on other parts of an architecture (such as the deployment of the system). So most of these ADLs are at best a partial solution to the problem of industrial architectural description.

The analysis of the number of ADLs that provide support for first-class connectors and architectural configuration as a first-class concept is shown in Figure 2.4.

This analysis reveals a very positive result, as the clear majority of ADLs in the study provide some form of first-class configuration, while less, but still nearly two-thirds, provide support for first-class connectors, which are both possible motivating factors for architects to use ADLs as existing informal and semi-formal notations tend not to support these concepts directly.

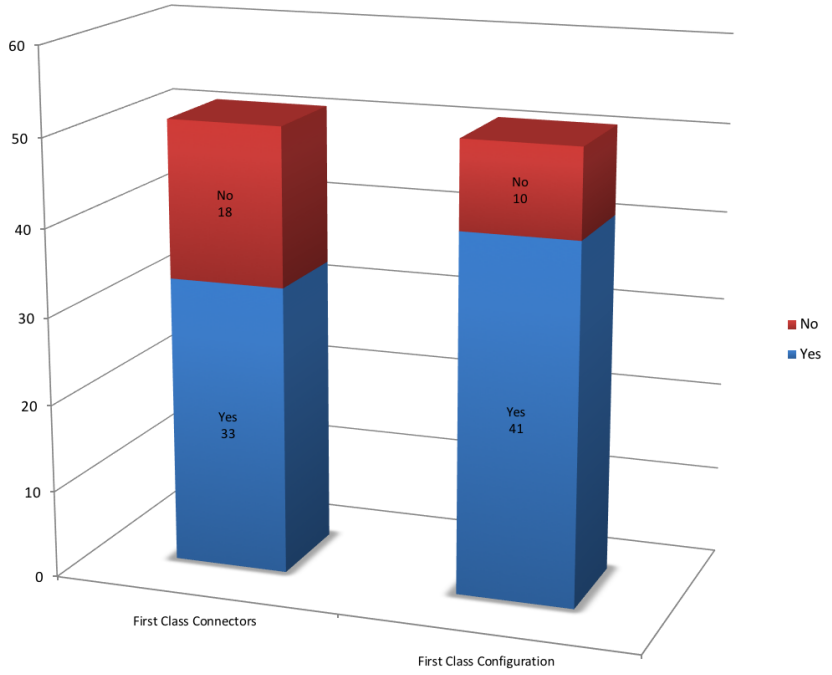


FIGURE 2.4: Connector and Configuration Support

The third area of interest to us were the *language mechanisms* available in the different languages, to assess the languages to see whether they could address common challenges (such as structuring and evolution) for large industrial architectural descriptions.

The attributes of the language that we analysed the literature for were as follows:

Structuring - what mechanisms are available for structuring a large architectural description?

Evolution - what mechanisms are provided to allow an architect to evolve an architectural description? (Such as the ability to describe architectural variations, the ability to version all or parts of the description or support for dynamic architectures).

Qualities - how provided or required architectural properties can be captured in the architectural description (e.g. properties, attributes, related models etc.).

Syntax - what concrete syntaxes are available to capture architectural descriptions in the language?

Analysis - how analysis of an architectural description could be supported using the language and any supporting technologies associated with it.

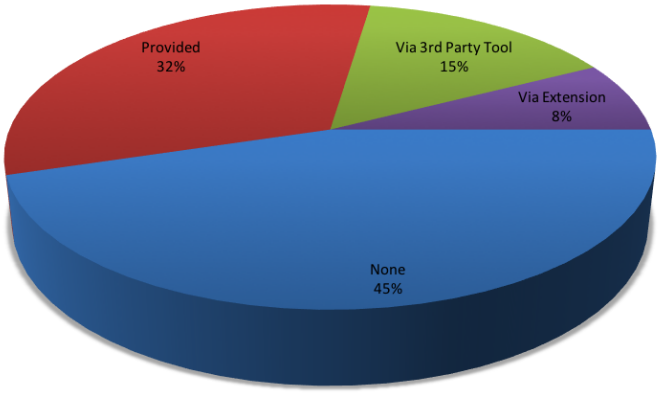


FIGURE 2.5: ADL Support for Architectural Analysis

Tools - what maturity are the tools that have been created to support the language? This can be "none", "prototype" (meaning an initial tool implementation applied to small problems), "research" (meaning a fully implemented tool applied to realistic problems by researchers), and "commercial" meaning that one or more tools have been implemented and used in an industrial context by people other than the tool's creators.

A common justification for using ADLs is the ability to perform automated analysis on the architectural description once it is represented using an ADL. Therefore, we were interested to understand how many ADLs provided some sort of direct support for analysis of architectural descriptions.

When we performed this analysis, we found that it was quite difficult because the analysis capabilities depend on support tools as much as the language and different ADLs provide quite different types of analysis capabilities. To allow us to answer the question, we have defined four types of analysis capability:

Provided - where the ADL has specific support in the language to capture the data necessary to allow an automated tool to use it for analysis and explicit consideration has been given to making this possible.

Via Extension - where the ADL has been designed such that its extension mechanisms could be used directly to support automated analysis via a tool.

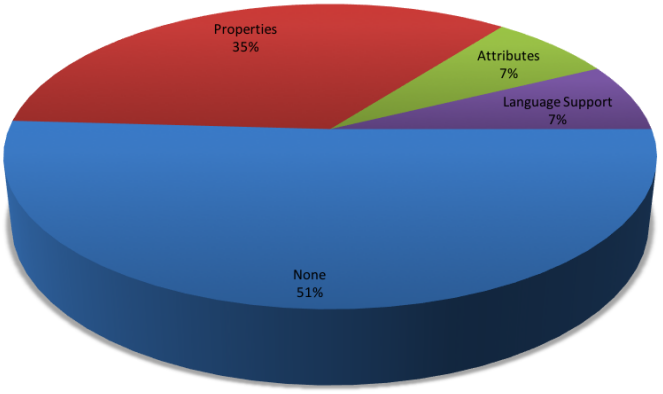


FIGURE 2.6: ADL Support for Capturing System Qualities

Via 3rd Party Tool - which means that the ADL provides some generic facilities that could allow a 3rd party tool to perform automated analysis, but where no explicit support for it is provided.

None - where the language does not appear to be amenable to automated analysis.

This analysis is presented in Figure 2.5. As can be seen, less than half of the languages appear to provide realistic possibilities for automated analysis (and of course of those that do, many do not have working tools available for them). We conclude therefore that automated analysis is only of interest in a subset of research groups working on ADLs. Our concern with this situation is that an important motivator for adopting ADLs does not appear to be addressed in most ADLs.

A key goal of software architecture is to ensure that a system achieves the set of quality properties required for it to be successful. This led us to expect that ADLs would provide strong support for quality properties and we were interested in the types of mechanisms used to represent them. Having read the literature, we discovered that there were three broad levels of support for capturing quality properties in an architectural description:

Properties - where a generalised mechanism of (possibly typed) name/value pairs was available in the language and could be used to capture non-functional requirements and qualities but is not specifically provided for that purpose.

Attributes - where specific pre-defined attributes relating to specific qualities (such as "transactions per second" for performance or "max connections" for scalability) can be captured within the language framework.

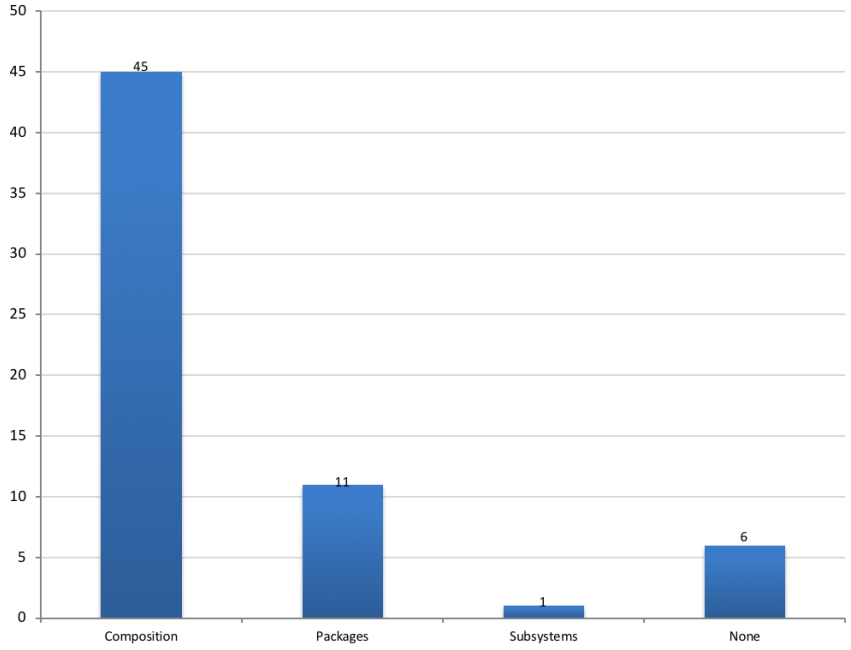


FIGURE 2.7: Support for Structuring Architectural Descriptions

Language Support - the case where languages provide a specific mechanism within the language for capturing quality requirements and capabilities (such as capturing security mechanisms and goals as first-class language elements or providing a general purpose QoS or quality requirements sublanguage).

Figure 2.6 presents our analysis of this aspect of the capability of the ADLs. It shows clearly that half of the languages provide no support for capturing system qualities, but that about a third (35%) do have a generic properties mechanism which could be used to capture quality-related information. A much smaller number provide the ability to capture specific attributes (7%) or have quality property features in their language (8%).

Many industrial systems are large, much larger than any case study or prototype experiment in the research domain. A typical industrial system today can contain 500,000 to 1mm lines of code and dozens to hundreds of architectural elements. Such systems cannot be described using languages that do not have effective structuring mechanisms to allow a system description to be broken down into smaller discrete parts. This led us to investigate the mechanisms that each of the ADLs in the study provided for structuring the architectural description. This analysis is shown in Figure 2.7.

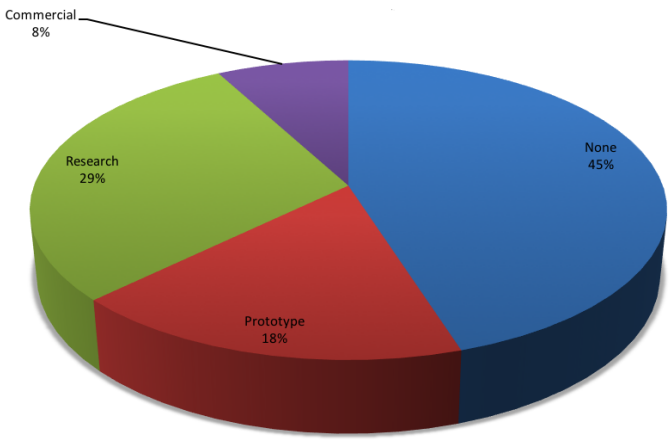


FIGURE 2.8: Tool Support Available for ADLs

While a few of the ADLs (about 12%) don't provide a structuring mechanism, most do, with nearly all of them offering *composition* and a few offering *packages* or *subsystems* (in most cases in addition to composition - hence the total of values in the chart is larger than the number of ADLs in the study). This is an interesting result, suggesting that most ADLs can be structured for large architectural descriptions, but that most of them utilise composition as the mechanism to achieve this, rather than providing a separate structuring mechanism like packages or subsystems. This is likely to mean that there are restrictions in the flexibility of the structuring facilities available in those languages where the only structuring mechanism available is composition.

ADLs are often developed in conjunction with supporting tools to help architects to use them. Depending on the nature and effectiveness of the tools, this can make the ADLs more attractive for use on significant projects. Figure 2.8 presents our analysis of this feature of the ADLs in the study.

The chart in the diagram shows that a very small percentage of the ADLs have commercially proven tools available to them, while about a third of them have tooling being used on research projects. Nearly two-thirds of the ADLs in our study provided no effective tool support.

2.1.4 Conclusions

Having systematically surveyed the research literature in the field of architectural description languages and analysed the results, we were now able to answer our ADL literature review research questions. Our answers are presented below.

ADL.RQ1 *Which architectural viewpoints does each ADL support?* All the ADLs in the study can represent functional views of the system and most of them only provide support for this view, but a small number of them allow deployment, concurrency or development views to be created too. Hence we conclude that the focus of most ADL research groups is how to represent the system's functional structure. This isn't surprising given how central a functional view is for most systems, but given the general acknowledgement of the importance of other viewpoints [13, 25, 92, 153] it does suggest that most of the existing ADLs will not be a complete solution to the problem of representing a software architecture.

ADL.RQ2 *Does the ADL provide structuring mechanisms for large ADs?* Most ADLs in this study (45) provide the ability to structure a large architectural description by allowing composition of architectural elements (of course composition can also be used for other purposes, such as information hiding). A smaller number provide specific mechanisms for structuring such as packages (11) and subsystems (1). A few ADLs, surprisingly, do not appear to provide a structuring mechanism (6). Some of the languages provide more than one mechanism that can be used to structure an AD (e.g. packages and composition) and this is why the numbers above sum to more than the number of ADLs in the study.

It is encouraging that most of the ADLs we surveyed provide at least basic facilities for structuring a large architectural description. However, large-scale project experience suggests that composition is not a particularly flexible structuring mechanism. This suggests that some consideration has been given to the use of the ADL for realistic problems, but it reflects a general lack of industrial validation. The languages that don't allow structuring are presumably in an early stage of development or are not intended for industrial use.

ADL.RQ3 *Does the ADL support the analysis of an architecture?* We found that about half of the ADLs (24 or 45%) do not appear to allow a realistic option for automated analysis of architectural descriptions, which was something of a surprise to us. Some of the languages do provide this though, with about 32% of them providing direct support in the language, while 15% allow this by providing mechanisms for 3rd party tools to embed information in the architectural description and 8% allow for analysis by providing an extension mechanism that could allow analysis information to be added to an architectural description.

A clear motivation for capturing and maintaining an architectural description is the ability to gain useful and reliable automated analysis that can provide insight into the design that is otherwise difficult to obtain. The fact that many ADLs being developed do not appear to provide analysis capabilities suggests that the problem of how to motivate others to use the language is often not part of the research process.

ADL.RQ4 *Can system qualities or quality requirements be captured in the ADL?* Quality properties are central to the role and activities of the software architect and so we hoped for strong support for capturing qualities and quality requirements in the ADLs. In fact, we found that more than half of the ADLs studied (28) do not appear to offer a facility to capture qualities and of those that do, most of them just provide a generic "properties" mechanism which can be used for a range of purposes including capturing qualities. Only about 7% of the languages provide quality property specific attributes in the language or include the ability to describe qualities as first-class elements of the language. This is a surprising situation if the ADLs are expected to be used in an industrial setting. Years of practical experience have taught us that achieving quality properties is a key objective of a software architect [25, 153], so we would have expected that supporting quality properties would have been an important requirement for an ADL.

ADL.RQ5 *Were prototype or production quality tools developed with the ADL?* Given the importance of tool support in achieving adoption of new software technologies, we were surprised to find that 45% of the ADLs in this survey do not appear to offer tool support that is ready for widespread transfer to industry and use. 29% of the ADLs provide a tool that has been used for a research problem, and only 8% of the ADLs have an associated tool that has been tested in an industrial context.

An important factor in applying ADLs on industrial projects is good tool support, ideally through extending or tailoring tools that industry uses already. In fact, we would go so far as to suggest that industrial adoption of any ADL without practical tool support is unlikely.

ADL.RQ6 *Has the ADL been applied to non-trivial problems outside the group of people who created it?* Given the effort required to develop ADLs, we assume that most of them are intended for eventual technology transfer to industry. Assuming so, the current degree of transfer out of the research groups is disappointing. We found that 58% of the ADLs have only been used by their creators, to create simple examples or experiments. Another 26% of the languages have only been used for case studies, again by their creators. 12% appear to have been used for research projects, outside the creating group, while a mere 6% of the languages have been applied in an industrial context.

In our opinion, a technology can only be considered have had an impact when it is used by people other than its creators, and when considering the products of research groups, this means significant usage outside the originating research group and ideally in an industrial context. Nearly all the ADLs we have surveyed fail this test, with less than 20% of them having been used outside their originating group (based on the publications we could find). It is possible that some of these ADLs have been used industrially but the case studies not published, but we feel that this is unlikely given the positive impact that publishing such case studies would have. We believe that this finding in itself is cause for reflection within the ADL research community (and it is similar to a previous similar finding, some years ago, from a workshop at a major software architecture conference [184]).

We continued our investigation into architecture description languages by undertaking an industrial project to create an architectural description to apply an ADL in an industrial setting. We describe this experience in Chapter 3.

2.2 Prioritisation of Architectural Effort

An observation we have made from software architecture practice is that the prioritisation of an architect's activity is a complex process, with many factors being taken into account. The software architect has broad responsibilities on a project and they can be involved in almost any technical aspect of a project at some point in its lifecycle. This can make it difficult for an architect to prioritise their effort in such a way that they have the time available to prioritise quality properties like energy efficiency that are often not explicitly prioritised by many of the system's stakeholders (we've observed that similar problems often occur with security, performance and scalability properties).

This situation led to our second research question *RQ2 - How can architects prioritise their attention on energy efficiency?*

We have not observed any formal, role-specific, heuristics or techniques in common use in industry, and so we were interested in whether there were approaches in the research literature which could be taught to new architects, to help them address energy efficiency as an architectural concern.

We were aware of generic time management techniques (like [4] and [90]) but we wanted to provide more tailored and prescriptive advice, specific to software architects, rather than the more general advice of the sort found in these techniques.

We were also already aware of an entire architectural method, called Risk and Cost-Driven Architecture (RCDA), designed by Eltjo Poort [142], which guides software architects to focus their attention using risks and costs. This method transforms the architect's approach from defining finished architectural structures at the start of a project, to working throughout the project to provide a stream of decisions, using the risk and cost of open decisions to prioritise the architect's work. It can certainly help architects to focus attention on their most important concerns and might well lead architects to consider energy efficiency more often if it was widely applied. However, we know that RCDA is not very widely applied across the industry and so we were interested in whether there were simpler approaches that required less commitment to adopt.

2.2.1 Research Methodology

We identified the research literature to include in the study by searching commonly used literature catalogues as well as manually inspecting the reference lists of relevant papers and following other relevant references found.

After some experimentation, we defined our search query for online catalogues to be *("software architect" or "software architecture") AND ("prioritize" OR "prioritisation" OR "focus") AND ("attention" or "effort")*. The syntax of the query had to be refined for different catalogues as they provide different search facilities, but this was a simple matter of performing multiple simpler queries against them.

We searched key online literature catalogues for papers that matched our query in their title, abstract or keywords. The four catalogues we used were the ACM Digital Library (advanced search), Google Scholar, IEEEXplore and Microsoft Academic Search.

The different catalogues returned different result sets for the query, with ACM Digital Library returning 11 results, IEEEXplore 12, Google Scholar nearly 80 and Microsoft Academic Search 12.

We then consolidated these results, by concatenating and removing duplicates from the ACM and IEEE results and manually scanning the results from Google Scholar to identify studies which suggested some aspect of prioritisation in their title (which resulted in 6 additional unique entries). Finally, Microsoft Academic Search returned relatively few entries and most were obviously not relevant to our search, but 4 additional items were added manually from this source.

When we inspected the results, it became obvious that there was a significant body of literature on the subject of prioritising system quality requirements. This is not exactly what we ask in our research question, which is more general, but we judged that such approaches might provide answers to the question, so we investigated them further. A manual process of reference following and additional searches revealed that many of the studies in this area have little or no consideration of industrial usage or validation. However, we did find some that appeared to have potential industrial relevance and this resulted in another 20 studies which had not been returned by our search query but we believe are representative of this research area.

At this point, we had a list of 46 candidate studies for consideration.

To narrow the list to the most relevant for our needs, we identified the following criteria for inclusion in our survey.

IC1 The study has been peer reviewed and is at least 2500 words long.

IC2 The study addresses some aspect of how architects prioritise their effort and attention (rather than just decision-making for example).

IC3 The advice, technique(s) or approach(s) in the study can be applied by an industrial software architect. (For example, they have some industrial validation and do not rely on a research tool or technique).

IC4 For practical reasons the study must be written in English.

When we applied inclusion criterion IC1, this removed 5 items from the list and applying IC2 (to focus on architectural effort prioritisation) left 13 for consideration. When we applied IC3, to ensure industrial applicability, this left 6 studies, due to the others not having any industrial validation. All of the studies we considered were written in English so IC4 was unnecessary.

2.2.2 Analysis of the Results

In his paper *What do software architects really do?*, Kruchten recommends a prioritisation of effort into 50% architecting (design, validation, prototyping, documenting), 25% getting input (from users, for requirements, on technology) and 25% providing information (communicating the architecture, assisting stakeholders) [93]. The recommendation is a result of his experience in managing a 10 person architecture team for a large-scale critical system in the early 1990s. To support his theory, he shows how other time allocations can cause architectural (behavioural) anti-patterns. This is anecdotal rather than empirically-based advice but is based on large-scale industrial experience. It does not help the architect to know which topics to focus on but is clear advice on the types of activities to spend their time on and the amount of time to spend on each.

In their paper *Decision-making techniques for software architecture design: A comparative survey*, Falessi et al report a survey of the decision-making techniques in the academic literature [51]. This is only partly related to prioritisation but it was one factor in their study. They found that the priority of a quality attribute could be defined by direct weighting by stakeholders, an elicited weight which involves "decomposing a multiple-criteria decision-making problem into a quality attribute hierarchy" (and they note the significant effort involved in this) or using a utility curve (which again is a significant effort). However their results are inconclusive, reporting that "no decision-making technique is more (or less) susceptible than any other technique to the entire set of difficulties taken into account in this study, that is, no decision-making technique dominates (is always better than) any other one". Therefore there is little concrete advice for the software architecture practitioner to take from this study.

Another survey paper is *Mature architecting – a survey about the reasoning process of professional architects* by van Heesch and Avgeriou [173], which reports the results of a survey of 53 industrial software architects to find out how they reason during decision-making. Prioritisation was part of the study scope by the only prioritisation insights in the results were that "the quality attribute requirements are clearly found more important than the functional requirements" and "requirements should be prioritized; the most important ones and the ones that are hardest to fulfil should be regarded first, as they bare potential risks" (sic). These insights sound like good advice but don't provide a significant degree of guidance to an architect on how to focus attention in a particular way.

The industrial study *Prioritization of quality requirements: State of practice in eleven companies* by Svensson et al reports on a study of architects in 11 industrial companies to find out how they prioritise their system quality requirements [167] . Representative techniques that they suggest are Numerical Assignment (Grouping), Pair-Wise Comparison, Cost-value approach, Cumulative voting ($100-Dollar Test) and Ranking. They had three main findings, that (1) ad-hoc and grouping (numerical assignment) of requirements are the dominant methods for prioritizing quality requirements; (2) although numerical assignment is used frequently in quality requirement prioritization, quality requirements are by default often considered to have the lowest priority; and (3) the reasons for not prioritizing quality requirements was not related to the prioritization process itself, but rather how quality requirements were treated in the overall requirements engineering process.

They also found that It is most common to have no specific or explicit criterion defined when prioritizing quality requirements. So for the architecture practitioner, one useful finding is that ad-hoc and grouping are used to prioritise quality requirements - these are techniques that industrial architects could easily apply. And some comfort can be taken from the second finding confirming the difficutly of prioritising quality requirements - it is a common problem in many environments.

Another study focusing on quality requirements is *Can we know upfront how to prioritize quality requirements?* [35] by Condori-Fernandez and Lago. The context of this study is service design for smart transport systems and involved studying a group of postgraduate students solving a service design problem set by an industrial partner company. The study aimed to identify which quality requirements are most important from the software architect's viewpoint and which are most stable over the service design process. The study does investigate the prioritisation decisions made, it doesn't ask how they are made, so does not significantly contribute towards answering our question. The main prioritisation conclusion was that "the design space specification phase of our service design method can play a crucial role in QR prioritization", which is not a generally usable result for most architects.

Finally, in the recent study *An industry experience report on managing product quality requirements in a large organization* [117], Mohagheghi and Aparicio report their experience in managing quality requirements in a Norwegian government department's large agile development programmes. There isn't much discussion of prioritisation of the quality requirements in the study with the only significant insight being that stakeholder workshops were necessary to prioritise quality requirements. However, there isn't any information on how the architects focused their effort during the process.

2.2.3 Conclusions

From a large body of research literature in the area of architectural prioritisation and particularly prioritisation of quality requirements, we found that very little of it is directly applicable to an industrial practitioner as much of it is speculative, theoretical or unvalidated. The literature we found that was of potential value, was an architectural design approach (RCDA) that helps architects to focus their attention by considering costs and

risks, an experience-based view of the amount of time the architect should spend on different activities (Kruchten), and an industrial survey result that most prioritisation is ad-hoc or based on simple (numerical) grouping of requirements (Svensson et al). RCDA stands out as the most comprehensive approach available, but is an entire architectural design method and so will not be attractive to practitioners who already have an architecture design approach. Therefore, the advice we have falls some way short of the amount of generally applicable guidance that we had hoped to be able to provide to an architecture practitioner. We will return to this topic in Chapter 4 to consider what further guidance could be provided on this topic.

2.3 Architectural Guidance for Energy Efficiency

As awareness of power usage and the need for improved energy efficiency grows across the ICT sector, attention is turning from an initial focus on data centre efficiency [42] to how the software applications running within them can contribute to a reduction in net energy consumption. Yet a previous survey of software architecture practitioners [20] has revealed both a growing awareness of the need to treat energy efficiency as a first-class concern and also a lack of tangible advice and guidance on how to achieve this.

Our goal is to raise awareness of energy as an architectural concern with software architecture practitioners and so we wanted to investigate whether there was existing research which had resulted in practical advice that software architects could use to address energy as a property of their systems.

This survey of the research literature partly addresses research question *RQ3 - What design guidelines can we provide to guide architects to improve the energy efficiency of their systems?*

2.3.1 Research Methodology

We identified the research literature to include in the study by searching commonly used literature catalogues as well as manually inspecting the reference lists of relevant papers and following other relevant references found.

After some experimentation, we defined our search query for online catalogues to be *"software architecture" AND "(energy OR power) (consumption OR efficiency)" AND (principles or tactics)*. The syntax of the query had to be refined for different catalogues as they provide different search facilities, but this was a simple matter of performing multiple simpler queries against them.

We searched key online literature catalogues for papers that matched our query in their title, abstract or keywords. The four catalogues we used were the ACM Digital Library (advanced search), Google Scholar, IEEEXplore and Microsoft Academic Search.

Predictably, the different catalogues returned very different results, with ACM Digital Library returning 95 results, IEEEXplore 63, Google Scholar nearly 3,000 and Microsoft Academic Search adding only 3 additional items.

We then consolidated these results, by concatenating and removing duplicates from the ACM and IEEE results and manually scanning the results from Google Scholar, adding any entries that had not been returned by the previous searches (which resulted in 16 additional unique entries, as well as confirmation of most of the entries returned by the ACM and IEEE searches). Finally, Microsoft Academic Search returned relatively few entries and most were obviously not relevant to our search, but 3 additional items were added manually from this source.

From this list, we identified the most obviously relevant items (those with titles indicating energy-related tactics or principles or architectural techniques) and found a small number of additional items that could be relevant, which were added to the list.

At this point, we had a list of 182 candidate studies for consideration.

To narrow the list to the most relevant for our needs, we identified the following criteria for inclusion in our survey.

IC1 The study has been peer reviewed and is at least 2500 words long.

IC2 The study addresses some aspect of energy efficiency as a quality property of a system.

TABLE 2.1: Classification of Architectural Guidance Studies

Classification	Count	References
Cloud Energy Efficiency Tactics	4	[143, 145, 146, 147]
Cyber-Foraging Tactics	2	[99, 100]
Energy Perspective	2	[78, 80]
Design Practices	1	[144]

IC3 The reference provides some tangible advice (such as a tactic, a design principle or a technique) which could be used to help address the energy qualities of a software system.

IC4 The advice provided by the study is relevant to addressing energy as an architectural concern (rather than, for example, code path optimisation).

IC5 The study is relevant to large-scale enterprise applications such as those found in large end-user organisations, the products of software companies supplying them, or those applications found in Internet-oriented companies, such as SaaS software providers.

IC6 For practical reasons the study must be written in English.

When we applied inclusion criterion IC1, this removed two items from the list and applying IC2 (to focus on energy as a quality property) removed a large number of the studies, leaving 43 for consideration. Of these 13, met the requirements of IC3 (to provide tangible advice) and 11 of them met the requirements of IC4 (i.e. they were architecturally relevant). Finally, applying criterion IC5 resulted in the elimination of another two studies, leaving 9 to be analysed further.

Having identified the 9 references of interest, we analysed their content in order to allow further classification. This proved to be a very straightforward process due to the small number of studies that met our inclusion criteria and very clear similarities between the references. Our classification is summarised in Table 2.1.

Four of the references [143, 145, 147] are all related to work performed at Vrije Universiteit in Amsterdam to investigate architectural tactics to address energy efficiency in cloud computing environments (so-called "green architectural tactics").

Two of the references [99, 100] are related to work on "cyber-foraging" (the process of allowing mobile devices to extend their computing power and storage by offloading computation or data to more powerful servers located in the cloud) performed at the Software Engineering Institute (SEI) at Carnegie Mellon University and Vrije Universiteit in Amsterdam.

Two of the references [78, 80] are the result of research into treating energy efficiency as an architectural concern, performed at Utrecht University in the Netherlands, with the industrial software company Centric Netherlands BV. This work led to the creation of an architectural perspective [185] that provides guidance to a software architect in how to improve the energy characteristics of the system.

The last reference [144] is an empirical evaluation of the energy consumption impact of two specific architectural practices, again based on research from Vrije Universiteit in Amsterdam.

2.3.2 Analysis of the Results

The studies in the *Cloud Energy Efficiency Tactics* set start with a literature review [143] that identifies three strategies for cloud energy efficiency, namely *Energy Monitoring*, *Self-Adaptation* and *Cloud Federation*. The study then identifies the components and techniques that have been proposed in the research literature to address cloud energy efficiency, within these three areas. These components and techniques are not, in most cases, directly usable by an application architect, but are useful background knowledge for them.

The later papers in this group develop the work further to provide direct architectural guidance for practitioners in the shape of a set of architectural tactics that we summarise in Table 2.2, using the tactic descriptions from the original reference.

TABLE 2.2: Architectural Tactics for Cloud Energy Efficiency

Strategy	Tactic	Description

Energy Monitoring	Metering	The Metering tactic consists of collecting power metering information from the hardware through dedicated software components called energy collectors. Collectors are usually in a many-to-many relationship with physical power meters. These collectors share information via an energy communication bus (ECB) that provides a common interface for energy information. In addition, the energy consumption information is stored in a dedicated energy database that can have different levels of granularity. Finally, a GUI component called an "energy dashboard" provides graphical representations of energy information along with useful reporting for both cloud service providers and customers.
	Static Classification	This tactic consists of classifying the different resources in terms of energy efficiency through the use of energy indicators. This classification is static, based on technical specifications and characteristics of the devices themselves rather than real-time information.
	Modelling	Modeling tactic enables a dynamic estimation of power consumption values through predictive energy models. These Models are embedded in energy indicators, similar to those in the Static Classification tactic. However, these energy indicators do not statically classify physical resources, but rather provide a dynamic estimation of the power consumption of the software components. Typically, energy models are built through regression analysis based on software runtime metrics (e.g. CPU, disk, memory).
Self-Adaptation	Scaling Down	This tactic reduces deployed IT resources when not required for current system load. This tactic utilises the idea of a "scale unit" which is a pre-defined quantity of "IT resources" [152] explicitly modeled as a software component. Modeling Scale Units is useful for planning the scaling operations because it defines a finite number of configurations for the VMs. Thus, it is possible to associate each configuration with a particular level of demand or system load. The Adaptation Engine is the component that performs the Scaling operation; this role is typically played by the Hypervisor. Another key component is the SLA Violation Checker to perform the checks required during scaling operations to ensure that service-level objectives are maintained at all times.
	Consolidation	This tactic concentrates VM instances on the minimum number of servers needed, to allow unused servers to be powered down. The main component of the Consolidation tactic is the VM Allocator, the software component responsible for live VM migration, which is often part of the Hypervisor. The SLA Violation Checker is also needed as well to check that service-level objectives are maintained after VM migrations.

	Workload Scheduling	This tactic is meant to prioritize and assign workload to the different virtual resources available in order to match demand. In this tactic, a Workload Scheduler is used to dispatch workloads to VMs. The Scheduler normally uses one or more queues to arrange the workloads. Queues can be differentiated in terms of priority levels, QoS requirements or deadlines. The SLA Violation Checker ensures that all service-level objectives are met.
Cloud Federation	Energy Brokering	This tactic makes energy information about services an additional parameter for service discovery and selection. It is realized by means of two components, an Energy Broker and a Green Service Directory (GSD). The Energy Broker enables access to energy-efficient services. It receives requests for cloud services that perform a specific task and returns a pointer to the most energy-efficient service available in the multi-cloud that can perform the requested task. In this, Energy Brokers make use of a GSD, which is a repository where all the cloud providers in the multi-cloud store the energy information of the services they provide.
	Service Adaptation	The Service-Adaptation tactic describes how cloud platforms should switch to these more energy- efficient services. It is realised by two components, the Energy Orchestator and the SLA Violation Checker. The Energy Orchestrator communicates with the Energy Broker to discover energy-efficient services that fulfill a certain task and performs the registration of those services with the system. This operation is authorized by the SLA Violation Checker, which ensures that the new services meet the service-level objectives required by the system.

These tactics are clearly useful in the correct context and can guide architectural decisions that allow energy to be considered as a first-class architectural concern. However, these tactics are primarily of use to an infrastructure architect or an application architect in an environment where they have a lot of control over the infrastructure environment and can make significant changes to it. Our area of interest is application architects working in a more common situation where they have limited control over the fundamental workings of their infrastructure platform.

The papers in the *Cyber-Foraging Tactics* group provide a practical set of tactics for solving architectural problems "...to enable mobile devices to extend their computing power and storage by offloading computation or data to more powerful servers located in the cloud or in single-hop proximity" [99] and develop these ideas to provide a decision model

to allow the tactics to be used effectively. This research has developed a sophisticated and extensive catalogue of tactics, comprising 10 functional tactics (with two further variants of them) and 12 non-functional tactics (with 6 further variants of them). These tactics offer extensive architectural advice and can clearly have a dramatic impact on the energy usage of mobile applications in systems that implement them. However, these tactics are not generally applicable beyond the specific problem domain of cyber-foraging and so do not help us address our question of how to guide application software architects in considering the energy efficiency of their applications.

The studies in the *Energy Perspective* group both define an architectural perspective (that is "a collection of activities, tactics, and guidelines that are used to ensure that a system exhibits a particular set of related quality properties that require consideration across a number of the systems architectural views" [185]). The perspective aims to standardise and organise the work of the architect in meeting energy efficiency goals for their system. This includes identifying, testing and selecting architectural tactics to achieve particular goals which the architecture does not initially meet, and the perspective aims to provide a framework to guide and formalise this process.

The full definition of the perspective can be found in [78] but to briefly summarise its key activities, it recommends the following steps in addressing energy as an architectural concern:

1. **Capture energy requirements** - Energy requirements can be formulated like other requirements, but it might prove difficult to translate the requirements into quantitative goals. Cross-checking the goals with stakeholders is essential to ensure the software will fulfil the requirements.

2. **Create energy profile** - An energy profile of the software provides the stakeholder with an objective starting point and benchmark to identify "hot spots" and determine whether the desired results have been achieved. Creating the profile requires energy consumption and performance measurements and can be visualized by creating an overlay for the architectural description.

3. **Assess against requirements** - Using the energy profile an assessment should be performed on whether the software meets the requirements and if not, the assessment should show what quantitative goals are not met and the (software) aspects that are directly related to these goals.

4. **Determine adjustments** - If required, adjustments should be determined to meet the requirements. Tactics, patterns and other known solutions should be considered that affect specific "hot spots" signalled by the quality measure.

5. **Apply adjustments** - Adjustments related to infrastructure can often be applied immediately by an administrator with little effort. Changing the software may be a significant amount of effort and disrupt other software delivery. The resources required to apply adjustments should be included in their business case.

6. **Evaluate adjustments** - Determine that requirements are met without unwanted side-effects. Requires measurement in the new environment and comparison against the energy profile. Relevant stakeholders need to judge whether the results are satisfactory.

As with most perspectives, the process is an iterative one, performing these steps repeatedly as the architect's knowledge of the situation increases and the architecture is changed in response to the process.

The authors of the perspective suggest considering the cloud energy efficiency tactics that we reviewed earlier in this survey along with the tactics of *Increasing Modularity*, *Optimising Network Load*, *Increase Hardware Utilisation*, *Concurrency Architecture Variations*.

The architectural perspective appears to be a valuable artefact for an architecture practitioner and would provide them with significant assistance in understanding how to treat energy efficiency as an architectural concern. The weakest part of the perspective at present is the set of tactics, which are quite general and would be more valuable if they were based on industrial case studies. We return to this point in Chapter 5.

Finally, the paper classified as *Design Practices* is an empirical evaluation of the energy impact of two well known design practices, namely to *Use Efficient Queries* and to *Put Applications to Sleep*.

The study reports a research exercise that ran two experiments. The first experiment ran inefficient MySQL queries (SQL relational database queries) and compared the energy usage of the scenario to the same functional scenario using an efficient database query. The second tested two versions of the Apache web server, firstly the standard version, which calls the operating system *sleep* function when it is idle, and one with all calls to the operating system *sleep* function removed. The same workload was applied to both versions of the server and the resource utilisation and energy consumption of each measured. The study ran the experiments in a specific controlled environment to allow interference-free measurements of resource utilisation and physical power consumption.

The results of the study were unsurprising with the use of an efficient query significantly reducing the machine resources required to process the scenario resulting in a 25% energy saving. Similarly, putting the application to sleep had a measurable and repeatable, but smaller, impact on energy consumption (14%).

This study provides a useful result to validate two simple design practices which have been proven to reduce energy consumption as a result of the study. However, the design practices are at a fairly detailed level and are unlikely to result in significant architectural decisions. So while of interest to a software architecture practitioner, this study does not provide a significant amount of guidance on how to start improving the energy characteristics of their applications.

2.3.3 Conclusions

Our research question, RQ3, asked: "What design guidelines can we provide to guide architects to improve the energy efficiency of their systems?" Our answer, from a review of the relevant research literature, is that the practice of treating energy efficiency as an architectural concern is still immature and more assistance is needed for the industrial practitioner. The most promising, generally applicable and currently valuable advice available can be found in the Energy Architectural Perspective defined at Utrecht University and Centric Netherlands BV. This perspective provides a clear and useful approach to guide the architect in addressing energy as an architectural concern. The weakest part of the perspective is the set of architectural tactics that it has to refer to. We partly address this concern in Chapter 5.

Other advice for the practitioner is available in the other studies, and is of value in the correct context, particularly for those implementing cyber-foraging systems or designing cloud-based infrastructure to be energy efficient, or those application architects able to design their infrastructure platforms.

2.4 Application Energy Consumption Analysis

As we have explained, the energy usage of information and communications technology (ICT) systems is starting to receive significant attention due to the sharply increasing demand for electricity to run the large number of data centres that exist today.

Significant research has been undertaken to understand the nature of the energy demand of ICT systems and significant progress has been made in a number of areas. At the data centre level, there is now a fairly good understanding of how data centres use power and some understanding of how to reduce the amount of power required in the data centre environment [40, 48]. At the micro-level, there have been some promising steps in understanding how individual programs consume power, to the point where it is possible to quite accurately predict and compare the power consumption of different options for program implementation under laboratory conditions [1, 24, 75, 127].

As we noted earlier, the software architect's interest sits between these two extremes as they need to understand the energy consumption at the application level, ideally at the level of the usage scenarios of their applications, so that they can use this information when making architectural tradeoffs.

This situation was the reason for our fourth research question, which was *RQ4 - How can we make architects aware of the runtime energy characteristics of their applications?*

In order to start answering this research question, we decided to perform a survey of the relevant research literature in order to understand what research had already been performed in the area of application-level runtime energy usage monitoring.

2.4.1 Research Methodology

We identified the research literature to include in the study by searching commonly used literature catalogues as well as manually inspecting the reference lists of relevant papers and following other relevant references found.

After some experimentation, we defined our search query for online catalogues to be *(software) AND (predicting OR measuring) AND (energy consumption) AND NOT (android or mobile) AND NOT sensor*. The syntax of the query had to be refined for each catalogue as they provide different search facilities, but this was a simple matter of restating the query as multiple simpler queries in most cases.

We searched key online literature catalogues for papers that matched our query in their title, abstract or keywords. The four catalogues we used were the ACM Digital Library (advanced search), Google Scholar, IEEEXplore and Microsoft Academic Search.

The different catalogues returned different sets of results, with ACM Digital Library returning 1216 results, which had to be manually inspected to extract the ones relevant to computing (to exclude, for example, references about energy management for buildings) which resulted in 24 studies of interest. IEEEXplore returned 31, Google Scholar returned 90 results and Microsoft Academic Search returned 84 results on a wide range of subjects, which, when manually inspected added 3 more relevant items.

We then consolidated these results by concatenating the result sets and removing duplicates from the list.

We identified the most obviously relevant studies in our candidate list (those with titles indicating work directly in the area of application energy monitoring) and scanned their reference lists for other relevant studies, which added a further 14 entries.

At this point, we had a list of 79 candidate studies for consideration.

To narrow the list to the most relevant for our needs, we identified the following criteria for inclusion into, and exclusion from our survey. IC1 - IC3 are the inclusion criteria and EC1 and EC2 are the exclusion criteria. To be included in the survey a study had to meet all of the inclusion criteria and not meet any of the exclusion criteria.

IC1 The study has been peer reviewed and is at least 2500 words long.

IC2 The study describes an approach, technique or technology for runtime monitoring of the energy usage of application software.

IC3 The study is relevant to large-scale enterprise applications such as those found in large end-user organisations, the products of software companies supplying them, or those applications found in Internet-oriented companies, such as SaaS software providers.

EC1 The study will be excluded if only relevant to mobile device technology or hardware (such as processor chips or sensors).

EC2 For practical reasons the study will be excluded if not written in English.

When we applied inclusion criterion IC1, which eliminated one study, and IC2 (that the study must be about runtime monitoring of application software) this reduced the list to 31 items. Applying IC3 (to ensure that the approach could be applied to an enterprise system) reduced the list to 19 items and applying EC1 to eliminate approaches that were only relevant for mobile or hardware devices left 13 to be analysed further. All of our candidate studies were written in English so EC2 did not apply.

Having identified the 13 references of interest, we analysed their content in order to allow further classification.

2.4.2 Analysis of the Results

The 13 references comprised 1 survey paper, 2 case studies and 10 papers describing energy monitoring tools of some sort.

The survey paper *A Review of Energy Measurement Approaches* by Noureddine et al [129] was a useful additional survey of the area and usefully summarised the approaches found in this area a few years ago (which based on our survey do not appear to have changed materially). They found that there are *model-based* approaches and *measurement based* tools. They found three main model-based approaches: cost-based models, models implemented in middleware systems and models based on measured server

workload. They also found three representative measurement based tools: PowerScope that maps physical energy measurements to processes and program structures; pTop from the Linux operating system, which creates energy estimates for processes using resource utilisation profiles gathered by a kernel module; and PowerAPI and Jalen, the authors' own tools that estimate power consumption for processes using operating system statistics and power readings and allocate power consumption to the elements of the software executed via the tools. This paper helped to provide some context for understanding this area, but is an incomplete survey and does not aim to provide information for the architect to use directly.

The two case studies were *Software Energy Profiling: Comparing Releases of a Software Product* by Jagroep et al [79] and the much smaller *Unit Testing of Energy Consumption of Software Libraries* by Noureddine et al [130]. Both of these case studies show how energy concerns can be integrated into the project lifecycle.

Jagroep et al report a significant research project performed in conjunction with an industrial company, to measure and compare the energy consumption of a commercial document generation system product, over a series of releases. They used both hardware and software energy measurement tools to measure the energy consumption of the product during specific usage scenarios each time the product was being prepared for release. While they believe they made reliable measurements and their approach was sound, it was difficult to explain many of the process level measurements, which came from the software measurement tools. This is a cautionary point for all involved in software-based energy measurement.

Noureddine et al report on a much smaller case study to illustrate how their tools can be used to perform energy measurement of real application code at the unit test level. They packaged the Jalen tool as a unit testing variant, extending jUnit, called JalenUnit to allow energy unit testing. They then use it to test the RSA algorithm, Google Guava and the Violin String libraries to measure energy usage and show variation as implementation changes. While not strictly architectural, this case study is a good illustration of how energy can be integrated into the project lifecycle as a system quality concern.

The remaining 10 studies all describe one or more energy measurement tools, which could be used for some aspect of runtime application energy measurement.

One paper, *A first approach on legacy system energy consumption measurement* by Cordero et al [36], described a tool called GreeSom (or Green Software Energy Measurement), which is an approach and a tool created to allow energy consumption analysis of (legacy) Java systems. The tool requires energy data from a physical meter and instruments the application to create an execution trace that it combines with the energy data to provide insight into the energy usage of the application. Unlike the other tools, GreeSom does not have an energy model as such but would be better described as providing analytics and visualisation on the combination of the server energy and application trace data sets.

Five of the papers describe tools that utilise cost-based models for energy measurement. The cost-based models are calibrated with the energy cost of basic operations within the machine (such as CPU time or disk writes) and combine these measurements with the measured activity of the software under consideration, in order to produce an energy estimate. Three of the references are from the same authors and describe different aspects of the same tools, therefore there are three cost-based tools described, which are:

AEMT (or Application Energy Measurement Tool) which was an early tool developed at Microsoft to provide basic energy estimates for an application based on operating system resource counters and application process activity. The tool uses a cost-based model based on CPU usage and disk activity but does not appear to be available for general use. [85]

E-Surgeon which provides Java-based open source tools that can be used to monitor process and method level energy consumption at runtime. The tools use a cost-based model based on CPU usage and network activity. The tools are available as open source software. [127, 128, 130, 131]

TEEC (or Tool to Estimate Energy Consumption) which extends the cost-based models beyond CPU (or network, or disk activity) ,to include consideration of memory usage. The tool reads manufacturer memory specifications (which are provided to it as configuration data) and CPU and memory usage statistics via the Sigar library in order to estimate memory related energy consumption. One interesting detail about the tests reported in the study is how negligible memory energy is compared to CPU. The tool does not appear to be available for general use. [1]

The remaining four papers describe tools that utilise regression models to perform their energy measurement. The regression models all vary slightly but are fundamentally similar and involve running benchmarks on a test machine, and collecting physical energy meter data for the machine and resource utilisation metrics for the software components under study. This data is then used to train a regression model to allow it to predict energy usage for the software components, given different patterns of resource utilisation. Most of the studies reported good measurement accuracy from these "software meters" after they were trained, but none of the studies discussed how generally useful the trained models were and what aspects of the runtime environment could be altered before they needed to be retrained, which is often a major limitation for regression-based models. The requirement for physical meter data during the training process could limit their applicability in enterprise environments when compared to the cost-based models.

The four tools did not all have a name but were:

cWatts+ is a piece of power monitoring middleware to estimate real-time power use of application components. It uses low-level CPU performance counters and temperature metrics, rather than resource utilisation measurements from the operating system, along with physical server power measurements. The use of temperature metrics is unique to this tool (although their significance is unclear from the information in the study). The power, performance and temperature metrics are collected during benchmarking exercises and used to train a regression-based power model to predict energy consumption. The tool is described as "middleware" because it does not directly affect the application but monitors it externally by reading operating system metrics for a set of threads defined at startup. The tool does not appear to be available for general use. [138]

Murwantara et al's model aims to estimate energy consumption for a set of components comprising a "web service" (meaning an HTTP server, an application server and a database). This model also assumes training a linear regression model using server energy data from a physical meter and process resource utilisation measurements from the operating system. From the information in the paper, it is difficult to know how the process really works. The tool does not appear to be available for general use. [122]

Singh et al's model which provides energy measurements at the operating system level by collecting OS level resource usage, server energy estimates and process level resource usage and combining them via an energy model to estimate the process energy usage. Server energy estimates are assumed to come from an energy meter, suggesting that a lab environment is assumed. The energy model uses support-vector-machine (SVM) based regression to try to derive the relationship between software component resource utilisation and energy consumption. The tool does not appear to be available for general use. [161]

VPMSPCP (or Virtual Power Meter Supported Power Consumption Prediction) is a tool that attempts to measure energy usage of application software components, using the approach of capturing server utilisation, component utilisation and server power readings. The tool creates a "virtual power meter" for the server and each server component ("web service") which are calibrated power models using a trained regression model. The work is sophisticated but their testing suggests that its predictive power is currently fairly low and they plan to develop it further. The tool does not appear to be available for general use. [102]

In summary, the survey paper partially summarised the field and provided a degree of perspective over the other papers, and the two case studies provided useful validation that is is possible to treat energy consumption as an architectural concern in the project lifecycle if approached correctly.

The other papers described tools that in principle could be used to monitor application energy consumption at runtime. Most of the tools used an energy model, the two varieties being a cost-based model, calibrated based on the runtime environment's energy characteristics, or a regression-based model that is trained using energy data and resource utilisation measurements from a benchmarking exercise.

Unfortunately, most of the tools described are not available as open source projects and only the E-Surgeon set (PowerAPI, Jalen, JalenUnit and Jolinar) are available for general use.

2.4.3 Conclusions

Our research question (RQ4) asked "How can we make architects aware of the runtime energy characteristics of their applications?" and through this literature review, we have managed to provide a partial answer to that question.

Over the last 10 years or so, there has been a significant amount of academic research in the area of application energy measurement and estimation and one of the case-study papers [79] appears to suggest that within the last few years, a degree of maturity has been gained that allows leading-edge practitioners to start experimenting in this area.

However, the research today focuses on measuring energy consumption at the operating system process or even the server machine level. To use these techniques and tools effectively requires the architect to design and run specific benchmark scenarios that will make this data meaningful for them. Experience suggests that it will often be difficult to find time in the delivery cycle to perform such testing regularly - just as it is often difficult to schedule performance testing today.

How much better it would be if the techniques and tools we create for architects would provide them with *scenario level energy measurements*, based on the incoming requests to the system. If designed correctly, scenario-level tools could be used in the production environment to measure synthetic workload (as is done routinely today for reliability and performance monitoring reasons) and provide the architect with a continual view of the energy characteristics of their system. We will return to this subject in Chapter 6 and discuss how this might be achieved.

Chapter 3

Modelling Large Scale Information Systems using ADLs

3.1 Introduction and Goal

As we reported in Section 2.1, there has been a great deal of academic and some industrial research into the definition of Architecture Description Languages (ADLs) to assist with the difficult task of clearly defining the architecture of software-intensive systems and there is still a significant amount of such research underway today [37, 43]. However, there is limited evidence of significant industrial use of the ADLs that have been produced, which we believe is for a number of reasons [17, 184] including the narrow focus of most ADLs and the mismatch between their strengths and the needs of practitioners. This is particularly marked in the information systems domain, where it is difficult to find any large-scale use of ADLs, whereas there has been some documented use of ADLs in embedded and real-time systems [5, 133, 174].

In order to investigate the second part of research question RQ1 ("What ADLs exist and can they be used to reason about the energy properties of a system?") we wanted to apply one or more of the ADLs from the research literature to the description of a significant system. In this chapter, we describe the project we undertook to create a large industrial architectural description, which led to the conclusion that the existing ADLs would not be

effective. This led us to define a simpler, more specific notation which was successfully used to describe the system.

This chapter is structured into a description of the research method used, an explanation of the context of the work, an introduction to the project that we undertook as the case study, an explanation of how we decided on an architectural description language to use, a description of the architectural description language we created, the case study of the application of the language, the experience we gained, the lessons we learned and the validation of the work and its use to answer the research question.

3.2 Research Method

Prior to the practical work described in this chapter, the ADL investigation work began with a systematic literature review [88], which was presented in Section 2.1, with the goal of performing an objective and repeatable investigation into the prior research in this field and identifying, evaluation and synthesising findings from the selected studies in order to answer pre-defined research questions.

The second stage of this research is described in this chapter and was to undertake an *"improvement problem to help a client"* style exercise (in Technical Action Research terminology [177]) which took the form of a significant case study which attempted to apply an ADL from the research results to the description of a large industrial system. This work resulted in the attempt to apply an academic ADL being abandoned but instead, a lightweight ADL-like notation was developed and successfully applied to the problem, resulting in a large and effective architectural description.

The research described here enabled us to answer research question RQ1.

3.3 Context of the Work

The case study was undertaken in a financial services firm that has developed a large custom information system to run its business. The software has been developed over a period of about 15 years and has grown from quite modest beginnings to the large system

it is today, comprising millions of lines of code, storing several terabytes of information. The system includes software modules that have been developed from scratch within the organization along with modules that have been acquired as a result of organizational acquisitions and that have been modified to integrate with the rest of the system.

Today, the system comprises about 20 major subsystems and over 10 million lines of Java, C++, C# and Perl, sharing a large multi-terabyte relational database. Although some members of staff who worked on the system in its early days are still with the firm (and actively involved with the system) it has grown to a size that means no individual understands it all, even at a reasonably high level of abstraction.

At the start of the project, there was no overall unified system description, although some teams responsible for subsystems did have their own documentation. This meant that the operation and interconnectedness of the system were often difficult to judge and this was starting to hinder change and evolution.

The organization wanted to perform some wide-ranging evolution and modernization of the system's implementation and realized that a useful first step, to enable better intellectual control over the system, would be to capture a unified description of the system's architecture. This led to the project described in this paper being undertaken.

3.4 Overview of the Project

The lack of a unified system description and the need to modernise and restructure parts of the system led to a desire to create some descriptive documentation for the system. At the outset, it was not entirely clear what sort of documentation was needed but discussion and exploration led to the conclusion that a current state architecture description was required. The discussions led us to conclude that the documentation needed to provide a description of the system's architecturally significant elements, responsibilities and interactions, rather than more detailed documentation of the design of individual modules.

Having gained a remit to proceed, we defined an approach and then worked with the software development teams to create the architecture description.

In order to have some clear goals and overcome some ambiguity in the goals of the work, some assumptions had to be made and these were:

1. The goal of the work was to create a comprehensive description of the architecture of the system as it exists to:
   (a) allow the architecture to be understood and analysed to allow estimation of key qualities such as its resilience, modifiability or energy properties;
   (b) allow impact analysis to allow architectural change to be planned; and
   (c) provide a reference to communicate the architecture of the system.
2. The audience for the completed documentation was architects, designers and development teams, so precision and completeness were important attributes.

Another decision which had to be made was whether to try to provide the option of automated processing of the architecture description. This would allow automated checking and analysis for applications such as power usage estimation or consistency checking. To achieve this, the architectural description would need to be captured in a parsable form with well-defined semantics. However, this requirement needed to be balanced against the resources needed to complete the work. It was decided to capture the information in a form that would be amenable to parsing later but not to slow down the project by imposing an onerous syntax for the information.

When the software development teams were approached to discuss their involvement with the project, it quickly became clear that while there was general enthusiasm for the idea, there was very little appetite for actually performing the work required. Therefore it was obvious that tolerance for learning new concepts or reworking outputs would be quite low. Hence, it was going to be necessary to identify a simple, low-ceremony approach that was highly prescriptive in order to minimize the possibility of teams producing inconsistent artefacts that would need to be reworked.

This initial interaction with the development teams, along with our assumptions about the goals of the project and the audience for the artefacts (see Section 4), meant that there were a number of implicit emergent requirements and constraints that we needed to take into account. These were as follows:

- Simplicity - the approach needed to be simple to understand and apply, first because senior managers needed to understand it quickly to agree to its use; and second, because the software development teams who needed to produce the design documents were not prepared to expend a lot of effort on learning a new language.
- Low Adoption Effort - given the low tolerance for significant adoption effort, people needed to be able to pick up the basics very quickly and incrementally learn what they needed. This extended to tooling where there was no enthusiasm for implementing, supporting or learning specialised modelling tools for this project.
- Conceptual Familiarity - the requirement for low adoption effort also meant that the notation and approach needed to support existing concepts that people were already familiar with (so the notation needed to contain the type of architectural elements found in the system, rather than generic elements that needed to be specialised or interpreted).
- Use Existing Tools - as mentioned above, requiring a new modelling tool to be installed and used for this effort would have caused the project to fail, so we had to use the tools already available in the organisation (which meant general drawing tools and wikis, although some licenses for a tailorable UML tool were available if needed).

Having understood and defined the goals of the work, and understood the priorities and constraints of the organisation that was going to perform a lot of the work, we started to consider our choice of language to use for the architectural description.

3.5 Selecting an Architectural Description Language

As became clear during the ADLs literature review work, presented in section 2.1, a large number of ADLs have been developed in an academic context, which we considered for use in this work.

We started our consideration of ADLs by looking for a case study that had attempted to apply ADLs in a large industrial context, but we could not find any published case

studies that report on an architectural description language being used to describe a large information system. There have been a number of published reports of ADLs being used to describe embedded or real-time systems (such as [37, 55, 104, 155, 174]) but these systems differ significantly from a large information system, with different concerns and requirements of an ADL.

As we saw in Section 2.1 Many ADLs have been proposed by researchers, including xADL [86], ADLARS [16], ALI [18], ArchiMate [97] and ByADL [43], to just a few of the more recent ones. Most of these languages exhibit novel approaches to architecture description, from support for interchange and interoperability to advanced architectural analysis capabilities. However, we found all of them lacking when we experimented with them to evaluate them for our project.

In general, academic ADLs focus on analytical evaluation and rigour but in this project, in common with many other industrial situations, the focus had to be on accessibility, practicality, and the ability to obtain a reasonably complete view of the structure and behaviour of the system with a modest amount of effort. In most situations, and certainly in this project, it is difficult to persuade practitioners to use an unfamiliar formal notation for architectural description and we were sure that if we did not focus on these pragmatic factors relating to the use of the ADL and the immediate usefulness of the result, we would have failed to get the cooperation of the teams and so the exercise would not have been successful.

A number of the research ADLs (such as ACME, xADL and ByADL) do, in principle, support the kind of description we wanted to create but when we experimented with them, we found that their very generic, general-purpose nature meant that they would have needed a lot of investment in tailoring and extension to be effective in this situation, given the need for conceptual familiarity. We would also have incurred significant investment to create tutorial materials and to evaluate, integrate or build tool support (such as providing drawing support in standard tools like Visio rather than academic prototypes). This meant that the benefits we would have gained from using these languages were not large enough to justify the adoption overhead and risk to the project.

The third observation that we made was that the majority of ADL applications reported in the literature as experience reports are confined to laboratory-based case studies rather

than exploring a practical application beyond an unrealistically small example. This further reduced our confidence that any of these languages were appropriate for use in this project, where we needed to show success quickly, and build a description of a 10 million line of code system with 20 subsystems, that would provide real benefit to the organisation.

It can be argued that many of these ADLs could be used in an industrial context but simply have not been applied to significant industrial projects to date. This is true, it is possible that they could be applied successfully but as explained above, we identified a number of serious concerns about their use for a significant industrial project. When we explored the languages, we found that we could not identify compelling features that would bring enough benefit to justify the risk of what was likely to be a difficult adoption process.

While not strictly an architectural description language, we also considered the use of ArchiMate [97] given the fairly wide spectrum of features that it provides for enterprise architectural description. However, upon closer investigation, we found that the primitives in the ArchiMate language were not a particularly good fit given our need to describe system (i.e. software) architecture rather than enterprise architecture.

It is also important to acknowledge that outside the area of information systems, there have been a number of industrial applications of ADLs for embedded and real-time systems, from consumer electronics (e.g. Koala [174], π-ADL [133]) to aeronautics and automotive systems (e.g. AADL [155] and EAST-ADL [37]). We investigated these situations through the published research literature and noted that the use of ADLs in these application domains has enabled automated system analysis, and automated code generation (e.g. MetaEdit+ [162]). This could well be one of the reasons that these applications of ADL technology were successful. However, given that analysis and code generation were not primary goals of this project and that our priorities were straightforward system description with easy and low-cost adoption we did not feel that we could reproduce the success of these case studies given the published ADLs we had available to us.

Considering the combination of these factors, our conclusion was that adopting one of the existing research ADLs was unlikely to be successful and could well endanger the success of the work. Therefore we reluctantly judged that it was going to be simpler and

safer to develop our own special-purpose notation and this was much more likely to result in a successful and useful architectural description.

3.6 The Approach

Having discounted the idea of using a formal ADL, we seriously considered using a tailored version of UML, with a suitable UML profile. The architects leading the effort already knew UML well, had used this approach before, and knew that it would have provided a basis on which to build our own specific notation. However, even a tailored version of UML needs some background knowledge of the underlying language in order to use it effectively; this was lacking in nearly all of the software development teams. The use of generic UML without a profile wasn't seriously considered because we knew it would meet with a lot of resistance and we would end up with significant divergence in the models that the teams would create.

We also considered just letting teams use their own informal notations. In principle, this would have removed one of the major points of resistance to the project and would have saved the effort of developing a notation. However, this had already been attempted in the organisation and the results were so varied that the exercise did not yield a useful system-wide description, so we also discounted this option.

Eventually, given all of the factors involved in this project, we reluctantly concluded that the project was most likely to be successful if we developed a simple, well-defined, very specific, notation that just contained the element types that would be found in this particular system and then provided the teams with support for it in desktop drawing tools and a wiki.

The initial discussions with the development teams revealed a varied understanding of modelling and abstraction, which led to a further realisation that the approach used was going to have to be comprehensible to modelling novices within minutes, rather than needing much effort to learn. We concluded that in order to avoid confusion, the models were going to have to capture specific component and connector types that described the physical structure of the software (e.g. runtime processes and inter-process communication channels) rather than more abstract and generalised concepts such as software

components and responsibilities. If the teams had been asked to describe their software in terms of more abstract concepts, we believe that the project would have collapsed under the weight of debatable, unverifiable abstractions and it would not have been possible to validate the models against the implementation.

Given the resources available, it was decided that using a wiki was going to be the most effective way to capture the data underpinning a graphical representation (the system element descriptions, connection definitions, inter-element dependencies and so on). A wiki allowed this information to be captured in an accessible way, without special tools but allowed very restricted formats to be prescribed that standardised its presentation and would be amenable to basic machine parsing later if needed.

The wiki approach of creating simple hyperlinked pages also allowed the architecture description to be decomposed into a set of manageable pieces, each with clear ownership but allowed these different pieces to be linked together to provide cross-referencing and navigation through the documentation. Hyperlinking also provides a simple sort of type checking in the documentation, as names can be linked to their definitions elsewhere in the wiki and if the name is wrong, a broken link results, which is immediately obvious.

We found that a wiki provides a lot of the flexibility of a word processor but can also provide basic mechanisms to allow structuring, templating and cross-referencing via simple conventions and most software developers find them very easy to use.

What a wiki does not usually provide is any support for graphical notations but the diagrams are the part of the architecture description that people spend the most time creating and reading, so they are important to get right. As explained already, having considered the options available, it was decided to create a new highly constrained graphical notation that would encourage the creation of graphical models at the right level of abstraction. In order to create a consistent notation that was easy to use, the guidance in [119] was followed in order to design the notation systematically.

The whole project, and in particular the definition of the graphical notation, was helped by the fact that while the system had grown rather organically, it had evolved according to a specific set of architectural constraints that could loosely be identified as an architectural style. This had limited the degree of implementation diversity and so reduced the number of concepts that it was necessary to represent in the description language.

Within the system, nearly all subsystems were comprised of the following types of elements:

- Message-driven servers that performed functional processing in response to events or requests arriving from a system-wide message bus;
- "Thick" clients that provided user interfaces and business logic (and typically communicated with the message-driven servers via the system message bus);
- Web interface servers that provided web user interfaces (typically written as Java servlets or Perl modules);
- Batch programs that performed some sort of periodic processing (such as end-of-day reporting); and
- Data loaders, which were a particular sort of batch program, which imported data into the system or moved data between subsystems.

The servers, batch programs and data loaders (and occasionally clients) would in turn normally have dependencies on a fairly large number of database objects (that is tables, views and stored procedures).

This very specific set of architectural element types was used throughout the implementation of the system, which meant that a simple ADL could be defined in terms of those specific element types.

A corresponding set of wiki page templates was created to support the capture of the supporting textual description for the graphical models in order to make the format required for the descriptions clear. This also made the management of the process easier as there were relatively few concepts that needed to be explained and it made progress easy to track in terms of completed wiki pages and sections.

3.7 The Style and Its Architectural Description Language

3.7.1 The Architectural Style

An analysis of the system's implementation revealed that it generally followed a set of discernable patterns created from a small number of types of architectural elements, which could loosely be described as an architectural style (taking the definition of architectural style from Shaw and Garlan [159] to be "a vocabulary of components and connector types, and a set of constraints on how they can be combined").

To allow the element types of the system to be described, a few basic concepts were used to set the context and help people to understand the key abstractions:

- System - the entire information system that is being described, which is a conceptual structure, composed of a number of interconnected subsystems that collectively provide its behaviour and qualities.
- Subsystem - a subset of the system that has a well-defined, cohesive, set of responsibilities, and in most cases a well-defined boundary and set of interfaces to its services.
- Component - a tangible software artefact which is delivered to the production environment and which is "executed" in some way at runtime (whether directly or by being called). Nearly all components are binary releasable elements, tracked in the change management system. (Elsewhere in this paper we refer to "components" as "elements" in line with much of the software architecture literature)
- Connector - the mechanism by which two or more components collaborate (usually by passing data between them). Examples are a messaging, a file system file, a database table, or a web service endpoint and invocation.

It is worth noting that even though our definitions of concepts like "component" and "connector" were quite specific, most people didn't really understand what we meant until we made the concepts very concrete with the specific types of component and connector that they were familiar with.

TABLE 3.1: Types of Architectural Elements

User Interfaces	
GUI	A traditional GUI client written in Java Swing, C# WebForms or C++ Motif.
WebUI	A user interface implemented as a set of web pages (typically as a set of CGI scripts or a Java webapp)
Command Line	A user interface implemented as a command line program, such as a script or a Unix command line utility
Servers	
Message-Driven Server	A server whose operation is driven by the receipt of messages from the system message bus
Server	A server whose operation is driven by a mechanism other than messages (such as RPCs, database polling or temporal schedules)
Batch Program	A program that is run from a scheduler and performs its operation in a single execution, without waiting for other system elements to perform any operations or for human intervention.
Data Loader	A program whose primary purpose is to extract data from a source and move it to a destination, typically transforming it in some way during the transmission.
Data Stores	
System database	The shared system database or a set of tables from it
File	A file on the file system
External Entities	
Subsystem	Another subsystem that communicates with this one in some way
External System	An information system outside our system that a subsystem communicates with in some way
External Data Source	A Data Source outside our system that a subsystem receives data from (such as a source of security prices)

As mentioned above, the basic types of system element used within the system were user interface programs, servers, data stores, external entities and a fairly specific set of connector types were used to link them. While these generic types of element sound fairly standard, what was interesting was the limited number of variations of them that were used in most of the system. These element types are summarised in Table 3.1.

The fairly restricted set of inter-element connectors in use throughout the system is described in Table 3.2.

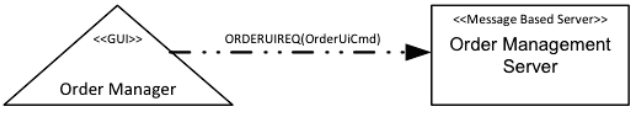
In order to allow for the inevitable special cases that are found in a system of this scale, an "other" type was also allowed for both components and connectors, which could be annotated using a UML style stereotype to make its type clear.

TABLE 3.2: Types of Architectural Connectors

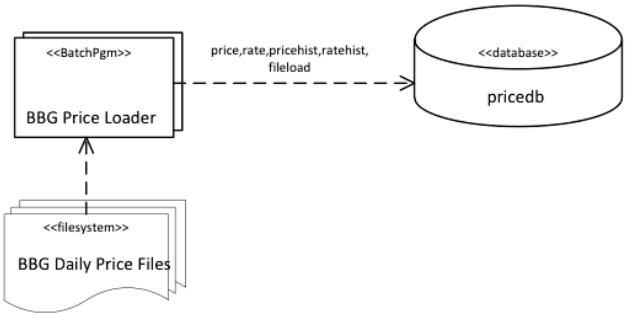
RPC	A synchronous inter-process procedure call (usually XML over HTTP)
Direct Invocation	An in-process direct procedure invocation (calling a library)
Database Data Flow	Writing data to a database table or tables to allow it to be used by another element
File Data Flow	Writing data to a filesystem file to allow it to be used by another element
System Messaging	Dispatch and receipt of messages over the system message bus via a named messaging destination

Most architectural styles limit the element and connector configurations that they allow. In this style, there weren't really any such constraints defined formally, although there were combinations that were encouraged and discouraged (e.g. UI Clients should connect to Message-Driven Servers but not access the database). However, most configurations of element and connector types could be found somewhere in the system. A number of the common patterns were captured as examples in the notation documentation.

A couple of examples of the patterns identified are shown in Figure 3.1.



(a) Thick client UI and message driven server



(b) File Loader, reads files, writes to database

FIGURE 3.1: Examples of the ADL Notation Illustrating Preferred Configurations

The notation used to express the examples is explained more fully in the next section but briefly triangular shapes represent user interfaces, rectangles represent server resident elements (servers, batch programs), files and databases are represented by the fairly conventional "record stack" and "drum" shapes, while connectors are represented by arrows using a variety of line types (the line type in example (a) being messaging, the line type in example (b) being stored data access).

3.7.2 The Architecture Description Language

Once the universe of required element and connector types was understood, we needed a notation that would allow instances of the style (i.e. the subsystems) to be clearly represented. As explained earlier, we decided to define a custom notation because the initial discussions with the teams had made it clear that getting people to use a specific tool or invest much effort in learning the notation was going to be very difficult. This was a key reason for creating a very simple notation and "just drawing pictures" rather than trying to apply a general-purpose notation or create machine-readable models.

Given people's general enthusiasm for diagrams over text, we chose to create a graphical notation rather than a more formal textual one. We could have created an equivalent textual notation to provide an alternative concrete syntax but we didn't need one for this project and as we were not trying to create a reusable ADL we had no reason (or the time) to create alternative notations.

When defining the graphical detail of the notation, the advice in [119] was particularly useful, in particular the exhortation to avoid construct overload, deficit, redundancy or excess, the suggestion to systematically consider the visual variables of each shape (shape, size, colour, orientation, brightness and texture) and the need for deliberate selection of shapes so that their appearance suggested their meaning, to help achieve semantic transparency.

We created the graphical notation by selecting a base shape for each major type of element (server, user interface, data store, external entity) and designing a variation of the shape for each subtype of the element. The diagrams were likely to be printed in black and white, so brightness and colour were used in a very limited way (just being used as an informal diagrammatic annotation, rather than having a predefined meaning). Each element had to have a name, shown on its symbol and optionally a stereotype (discussed below). Examples of the notation for some of the more important element types are shown in Figure 3.2.

A triangle was used as the base shape for user interfaces and a rectangle for server resident components. The triangle was chosen as it hinted at the head and shoulders shape of a user and the triangles were then modified slightly for each type of user interface

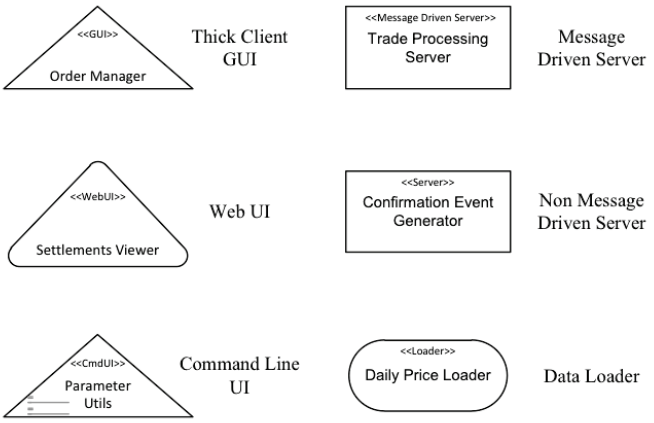


FIGURE 3.2: ADL Element Types

(the thick client having sharp corners, the web user interface having rounded corners as it blurs the distinction between "client" and "server" and the command line utility having a graphical representation of a command line interface added to it). Similarly, a rectangle is the base shape for server elements (based on long-accepted conventions) with a stereotype being used to indicate the type of server and a "lozenge" variant being used to indicate a data loader (hinting at pieces of data being transmitted through it).

An arrow of some form was used to represent all of the connector types, with the arrowhead usually indicating the direction of data flow. All connectors were defined to be one-way connections, with the exception of data access connectors, which could indicate read and write activity with arrowheads at both ends of the connector if appropriate. The convention for RPC connectors was defined to be a one-way arrow from the caller to the target. No attempt was made to represent the various complicated possibilities of dependency and initiation of interaction using the connector symbols. Each connector had to indicate what was carried over the connection, with message flows being annotated with a message data type, file and database connectors being annotated with table or record names, and RPC and direct invocation connectors being annotated with the name of the service or procedure they were calling. Examples of the notation for the main connector types are shown in Figure 3.3.

The RPC or direct procedure call is shown using a solid arrow, messaging is shown using a line with embedded dots, suggesting messages flowing over it, while data access

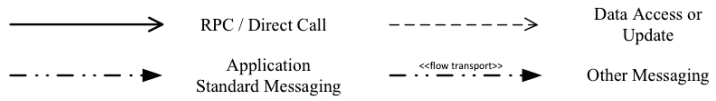


FIGURE 3.3: ADL Connector Types

is shown using a regular chain line, suggesting records being read or written over the connector.

A general mechanism used on elements and connectors was the stereotype, adopted from UML, where the type of an architectural element is made clear by annotating it with a type name using the convention "«type»" on the symbol concerned. This allowed the casual reader to understand the types of element on the diagram without having to understand the notation and allowed new element types to be easily introduced.

The semantics of the elements and connectors were generally based on the semantics of the corresponding element and connector implementations in the system: broadcast messaging in the system worked in a particular way, a relational database has well-understood behaviour, a web service call is widely understood and a message-driven server was a concept that most people understood with little further explanation. Undoubtedly there were cases where elements on diagrams had surprising behaviour because they did not behave entirely as expected given their type but on the whole, the resulting documents were good enough to form a useful architecture description.

In order to ensure that the process produced more than just pictures, we defined a set of required attributes for each type of element and connector. Part of this task was defining enumerations of expected standard values for many of the attributes, again to standardise and simplify the process of recording the information (such as standard lists of data domains ["trading", "counterparties", "securities", ...], lists of programming languages in use [C++, Java, C#, Perl] and so on).

In order to simplify and standardise the subsystem descriptions, a set of wiki page templates and a comprehensive Microsoft Visio stencil were created, along with clear instructions, quick reference material and - most crucially - a fully worked example of the documentation for one subsystem. This allowed a number of conventions, such as hyperlinking element names to allow navigation through the documents, to be illustrated and

encouraged by example. A hierarchy of empty wiki pages for the required subsystem descriptions was also created so that authors knew where to put their documents and so they could be unambiguously referenced.

The result of this process was a relatively informal definition of a simple ADL with a graphical notation and set of well-defined conventions for storing the supporting text needed to explain and fully define the subsystem descriptions. The ADL is tied very strongly to the particular architectural style of this system (its element and connector types) and we deliberately did not attempt to generalise the language, as this very tight link to the system to be described was one of its major strengths for our situation. In this way, our ADL is rather like the ADLs defined to support specific implementation frameworks like DAOP-ADL [140] which was developed to describe DAOP applications [139] and CBabel [149] which was developed to allow the definition of CR-RIO applications [105].

3.8 A Case Study of the Approach in Use

The system described in the case study is the Asset Management System (AMS) a financial asset management system used by a fund manager to support making and executing investment decisions for a large-scale investment portfolio. The example is based on a real subsystem from the case study, modified slightly in order to retain anonymity.

The primary aim of the system is to allow a fund management team to manage a portfolio of holdings in financial instruments (primarily equities in this case). The system must allow them to view the content of their portfolios and to use analytical tools and market data (such as prices, volatilities, projected interest and foreign exchange rates and projected bond yields) to make investment decisions. The system provides the ability for suggested changes to portfolios to be automatically calculated on demand or from a temporal schedule and also allows direct entry of orders to buy or sell securities to allow for investment strategies that are outside the scope of the system. Once lists of orders to buy or sell securities are generated, the system allows them to be dispatched to another system for execution and it receives the results of the execution of those orders in return, to allow the current holdings to be updated.

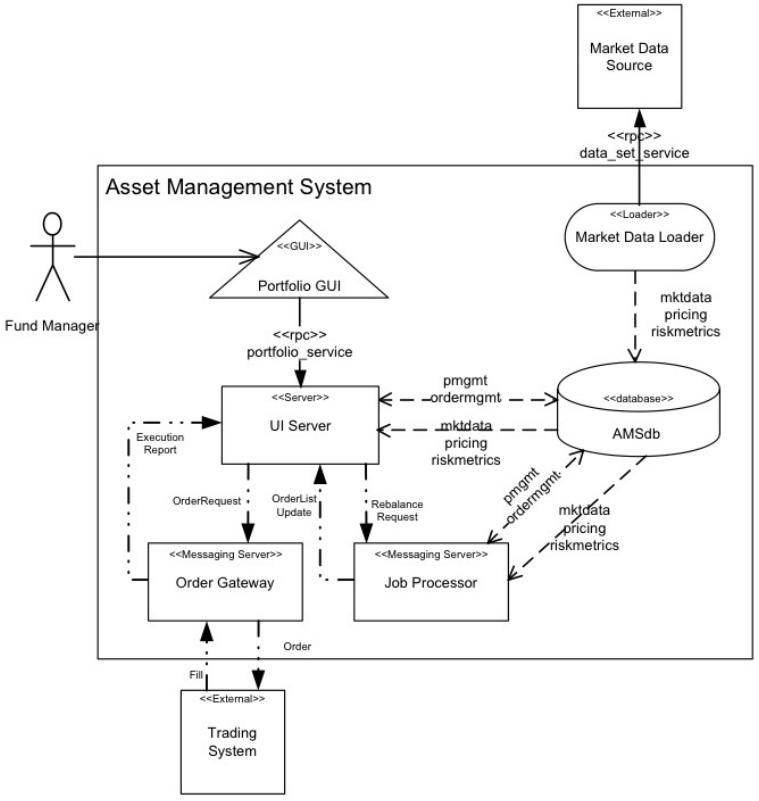


FIGURE 3.4: The Asset Management Systems

3.8.1 Architectural Description

The functional structure of the AMS is described using our system-specific ADL notation in Figure 3.4. The elements of this architectural structure are described in Table 3.3.

3.8.2 Example Scenario - Generate Order List

The key functional scenario for this system is to allow a fund manager to generate an order list to "rebalance" a fund based on an analysis that identifies the theoretically optimal holdings for the portfolio and execute that set of buy and sell orders, reflecting the results in the portfolio. The interactions required to implement this scenario are illustrated in Figure 3.5.

The interactions between system elements necessary to implement this scenario are described in Table 3.4.

TABLE 3.3: Elements of the Asset Management System

Element Name	Type	Description
Portfolio GUI	GUI	The responsibilities of the Graphical User Interface (GUI) are to provide the asset managers using the system with the ability to view and analyse their portfolios, to request (and monitor progress of) long running system operations (such as order generation) and to check, enter, dispatch and monitor orders that go for execution to trading systems. The GUI provides a human interface and requires an RPC interface to the UI Server to provide it with services and data.
UI Server	Messaging Server	The responsibility of the UI Server is to provide the data access facilities that the UI requires (accessing data from the AMSdb internal database) and to dispatch requests for orders or for long running work (such as analysis processing) to be carried out by other parts of the system. The UI Server provides an RPC interface to expose its provided services to the GUI and requires an SQL query interface to the system database and a messaging interface to allow it to request and monitor order dispatch and long running work.
AMSdb	Database	The system database's responsibility is to store the portfolio, analytical, market and (system) operational data that the system requires to operate. It provides an SQL based DML interface to allow data to be inserted, manipulated or retrieved.
Job Processor	Messaging Server	The responsibilities of the Job Processor are to execute long running processing items ("jobs") such as investment analytics and automated order list generation. The processor can be configured to run particular jobs on temporal schedules and can also be requested to execute particular jobs on demand. The processor provides a message based job control and status request interface and requires an SQL query based interface to the database.
Market Data Loader	Loader	The responsibility of the Market Data Loader (MDL) is to retrieve various forms of market data from an internal Market Data Source system and load the data into the database, handling versioning and business date identification as part of the loading process. The datasets required include securities prices, bond yields, interest rates, FX rates, volatilities, correlations and so on. The loader requires a data retrieval interface to the MDL system, allowing data sets to be retrieved on demand.
Order Gateway	Messaging Server	The responsibility of the Order Gateway is to accept incoming orders to buy and sell securities (including order parameters such as execution strategies and price limits), to forward these requests to a trading system for execution and then receive the execution reports ("fills") indicating order execution and broadcast these to other interested parts of the system. The gateway provides a message based order request interface and a broadcast status interface and it requires a message based interface to allow order submission to a trading system.

TABLE 3.4: Interactions for the Portfolio Rebalance Scenario

Step	From	To	Type	Connector	Description
1	GUI	UI Server	RPC	portfolio service	Fund manager selects a portfolio and instructs the system to create an order list for it. The GUI invokes an RPC indicating that the indicated portfolio should be rebalanced.
2	UI Server	Job Processor	Msg	Rebalance Request	The UI Server sends a request message to indicate that the portfolio should be "rebalanced". This is routed to the Job Processor.
3	Job Processor	AMSdb	DB	pmgmt and ordermgmt schemas	The Job Processor receives the message and in response initiates a portfolio analysis job to identify the theoretical optimal holdings in the portfolio and generate buy and sell orders to move the portfolio to that state. Portfolio state read from "pmgmt" and order lists written to "ordermgmt"
4	Job Processor	UI Server	Msg	OrderList Update	The Job Processor sends a status message indicating that new order lists exist, which is routed to the UI Server
5	UI Server	AMSdb	DB	ordermgmt and pmgmt schemas	The UI Server accesses the database to get the new portfolio state and associated order list state
6	GUI	UI Server	RPC	portfolio service	The GUI calls the UI Server for a status update and gets details of the new order list in return
7	GUI	UI Server	RPC	portfolio service	The GUI makes an RPC call to the UI Server to indicate that the order list should be traded
8	UI Server	Order Gateway	Msg	Execution Request	The UI Server creates a message to request the order list to be traded (including the list of orders) which is routed to the Order Gateway
9	Order Gateway	Trading System	-	-	The Order Gateway sends the orders to an external trading system and receives status updates in return as the orders are executed
10	Order Gateway	UI Server	Msg	Execution Report	As the Order Gateway gets execution updates, it creates execution report messages which are routed to the UI Server
11	UI Server	AMSdb	DB	pmgmt and ordermgmt schemas	The UI Server updates the database with the status of the orders and the effect on the portfolio
12	GUI	UI Server	RPC	portfolio service	The GUI makes RPC calls to the UI Server and gets the updated status of the orders and the changes to the portfolio in its response

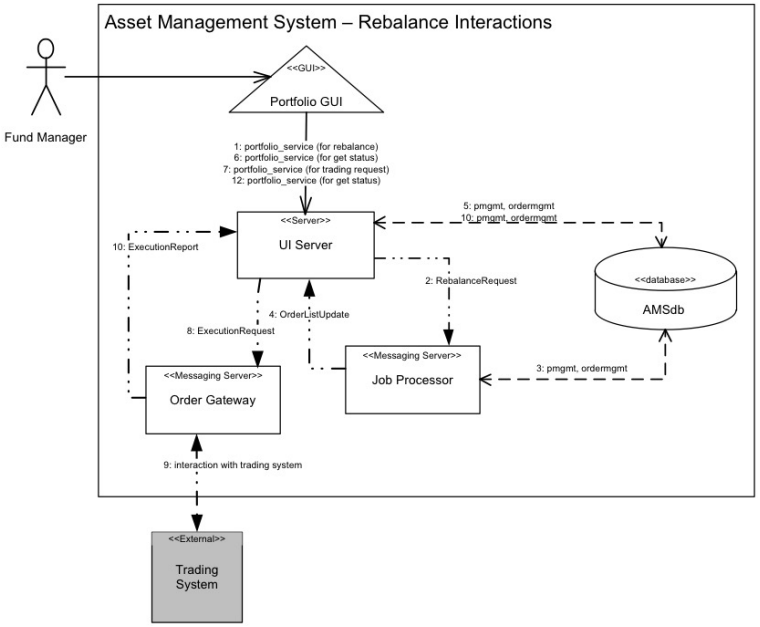


FIGURE 3.5: Portfolio Rebalance Scenario Interactions

A full architectural description for a subsystem would also include a lot of operational and implementation oriented information such as links to operational instructions, links to source code control systems and automated build systems and links to test specifications and results. We do not attempt to reproduce any of that here as the majority of such information was in the form of links to other internal systems and nearly all of the information is context-dependent and so not particularly meaningful outside the organisation operating the system.

3.9 Experience Gained

3.9.1 Creating the Architecture Description

As mentioned earlier, two experienced architects led the project to create the architecture description, which included identifying the underlying architectural style, defining a clear approach, defining the ADL and leading the work to capture the architectural descriptions. There were approximately 20 development teams who owned significant subsystems that needed to be included in the scope of the project.

In order to organise the work, the development teams were ranked in order of the criticality of their subsystems in terms of how central they were to key organisational workflows and this acted as an ordered backlog of work for the architects.

The general approach taken to the task was simple and involved approaching each team and asking for a single person to be nominated as the owner of their documentation. A conference call was then held with this person and the group manager to explain the project and the approach. The team was asked to commit time and effort to complete their documents and to commit to a timeline for completing the agreed deliverables (a team often had a number of subsystems that needed to be documented and for planning purposes, the creation of a subsystem description was decomposed into some standard subtasks). In return, the architects leading the effort offered training, practical assistance (such as drawing diagrams) and to review the descriptions produced.

The interactions with different teams varied greatly, with some teams producing their documentation largely unaided, needing only some review and minor correction, while others were simply incapable or unwilling to produce what was needed and the architects ended up writing most of the documentation for these teams.

The reasons for the problems encountered with development teams varied. In some cases, it was simply a lack of interest, often from the development manager who perhaps didn't see the value of the deliverables. In other cases, there seemed to be a genuine difficulty in understanding how to represent their subsystem. In general, this seemed to stem from an inability to abstract away from the implementation, resulting in a confusing mix of concrete and totally abstract concepts, which they then struggled to relate to each other. None of the subsystems was very difficult to represent, and in order to make progress, the architects often stepped in and simply created the models.

Another interesting problem was tooling. Everyone in the organisation had access to the wiki and knew how to use it, so document authors could fill in the tables and text without any difficulty. However, not everyone had access to Microsoft Visio and even of those that did, some obviously didn't know how to use it. Again, the solution to this was simply for the architects overseeing the process to create diagrams for some subsystems. This was a useful lesson and provided further evidence that avoiding UML and more specialised modelling tools had been a good decision. In this organisation, requiring the use of

UML and modelling tools would have been a significant barrier to getting architectural descriptions created.

Over time, a significant and useful body of subsystem descriptions emerged and this allowed the architects to create a summary level architecture description that showed how the subsystems related to each other. Some use of scripting to process the wiki subsystem descriptions and drawing tool macros to generate parts of the summary level diagrams allowed some degree of automation, although it was still a fairly manual process.

The process of capturing the architecture description took about six months, with the architects working on it approximately 60% of their time and the development teams working on it as their project schedules allowed.

3.9.2 The Results of the Project

The outputs of the project were as follows.

- A fairly consistent architecture description for most of the system that provided an accurate and largely complete view of its subsystems, their components and their dependencies. Each subsystem was described using a standardised approach, which captured the same information for each one and presented it in a consistent manner through the use of the templates provided. This made the information provided easy to navigate and check for completeness.
- An informal definition of the architectural style used across most of the system and the typical patterns used when implementing it.
- A degree of visibility and understanding of the structure, scale and interconnectedness of the system which hadn't been achieved before. The consistent presentation of system design information in a single location allowed the overall system structure to be more easily understood compared to the previous inconsistent descriptions on scattered wikis and websites. This appeared to allow a number of senior technical managers to achieve new insights into the system.

- An insight into the degree of implementation uniformity between the different subsystems of the application. While many subsystems were implemented in a very similar way, like any large system (particularly one which has had other applications integrated into it), parts of this application were implemented in ways that didn't follow the normal set of conventions. While there was already a general awareness that these less standard subsystems existed, the models made it easier for senior technical staff to gain visibility of this and decide whether they wished to direct any changes to the application as a result.

As mentioned earlier, the project did not have particularly clear goals for the architecture description once developed. A number of people did find it insightful and there seemed to be a general consensus that it was a useful description to have. However, organisational changes then meant that the architects involved moved on to other work, so the project effectively came to an end. Since then another group within the firm has adopted the architectural description and continued its use and maintenance (primarily to support production operation of the system, a use which was not foreseen at the outset of the project).

3.9.3 Evaluating the Usefulness of the ADL

Our Early practical experience led to some rapid refinement of the notation to remove ambiguities that had not been apparent to its creators and to introduce some missing concepts. However, after three or four teams had used the approach over a period of about 6 weeks, the ADL itself remained stable for the rest of the project.

As the project neared completion we started to validate what was being produced by talking to some of the important stakeholders, particularly the senior technical managers in the organisation. To do this we met with them and demonstrated what was being produced and what the completed architectural description would contain, discussing possible uses of it (such as impact analysis, pre-implementation reviews, incident post-mortems and regulatory enquiries). We were pleased to find that this stakeholder group reacted positively to what they were shown, with responses ranging from fairly neutral (where the possible usefulness was acknowledged but no specific use of it particularly

interested them) to very positive (where they wanted to start using it immediately). Given this informal but consistently positive sentiment, we felt that our notation and approach had been validated (an outcome which was anything but certain at the start of the project, when the use of a specific notation and a highly prescriptive form for the documentation had been viewed as very risky).

A factor that was constant throughout the project was that teams who had the ability to identify clear abstractions for their subsystems also appeared to find the ADL helpful and straightforward to use, as the ADL gave them a clearly defined way to represent their models and they didn't have any difficulty in representing their models using it. These teams tended to create their models with little or no assistance once they'd asked a few clarifying questions about the purpose of the models and the semantics of the notation.

In contrast, teams who struggled to identify good abstractions never really grasped how to use the ADL and needed constant assistance, to the point of needing to have parts of their architectural descriptions were completely rewritten for them. What was interesting about this stark contrast in modelling ability was that we could find no obvious factor to explain it in terms of educational background, age, team size, technology preferences, type of subsystem, geographical location or any other relevant factor. We speculate that it could be related to a person's thinking style (for example whether they tend towards abstract or concrete thinking [186]) but we did not investigate this during the study. We did observe that even in teams that produced good models, the ability and enthusiasm to do this varied and even for large subsystems we found that it tended to be one or two people in a team who did all of the modelling on behalf of the rest of the team. We don't know whether there were many other people in those teams who would have done an equally good job but based on hallway conversations, we suspect not. Our conclusion was that relatively few people in the general population of software engineers we worked with find modelling straightforward but we were not sure why this was the case.

We viewed this experience as validation of the approach that had been used. People who could create models and knew what they wanted to represent were able to use the ADL effectively with minimal training, so it was obviously usable by mainstream practitioners. On the other hand, the approach did not help those people who found it difficult to create a model. It had been hoped that the straightforward and prescriptive nature of the approach

would guide people to create useful models, even if they did not find modelling easy, and it was a disappointment that the approach failed to achieve this.

3.10 Lessons Learned From The Project

At the start of the project, no one involved in it had much experience in using ADLs in a large-scale industrial context. Our experience was limited to the use of ADLs in an academic context and some significant experience of using UML for architectural modelling in large industrial projects. Therefore, we had relatively few preconceptions as to how successful the project would be and on the whole, we judged it to be a success.

The main lessons that were learned during the course of the project were:

- A specialised ADL can have benefits over a general modelling language like UML and even a simple ADL can be used to create useful results.
- The more specialised an ADL is, and so the closer it matches the implementation style of the system being modelled, the easier people seem to find it to use. While at first glance this sounds like an obvious point, it is contrary to the conventional industrial approach of using a general modelling language like UML or SysML and also contrasts with the domain-independent nature of most academically developed ADLs.
- Carefully designing the detail of the graphical notation pays off. Using shapes that hint at their meaning and using a range of graphical dimensions to differentiate shapes helps people to remember them, even if they don't guess the link between the shape and the concept themselves. Again, this is not reflected in mainstream notations like UML or most existing ADLs, where little effort is made to identify meaningful symbols for concepts.
- Consistency in the notation is very important and having a base shape for a general concept with refinements to it for different sub-concepts appears to help people when interpreting the diagrams.
- Providing high-quality support materials including an example-based description of the approach and notation, a number of realistic completed examples and a set of templates for new documents is very important. We found repeatedly that people are much better at "filling in the gaps" rather than following a set of instructions and creating something from scratch.
- Utilising familiar tools helps with the acceptance of the approach. In this particular organisation, there were no complaints or difficulties with the use of a wiki for the text and tables information, whereas a very widely used commercial drawing tool (Visio) caused problems, even with a carefully tailored template, because it was not widely used in the organisation already.

These lessons aren't all that surprising but the importance of what seemed to be quite minor things (such as worked examples and quick reference cards) is important and is

useful to bear in mind for the future. The importance of matching the ADL to the specific domain being modelled is also a lesson that is not reflected in most modelling languages today, which tend towards the general rather than the specific.

3.11 Validation

The goal of this work was to establish whether an ADL could be used to capture the architectural structure of a complex system so that the architectural description could be used to provide insight into the architectural properties of the system such as performance, resilience, modifiability or energy efficiency.

Looking back to the specific goals we set at the start of the work, we considered whether the architectural description we had created was a useful catalogue of the current state, could allow the architecture to be understood to allow estimation of qualities such as its resilience, modifiability or energy properties, to provide a tool for impact analysis and to provide a reference to communicate the architecture of the system (see Section 3.4).

- Create a Catalogue of the Current State - the project created the first comprehensive description of the system and so provided a very useful descriptive catalogue of the current state of the architecture. The weakness of the architectural description as a catalogue was that it was only as comprehensive as the authors of each piece decided to make it. However, it was possible to cross-check it against a number of systems that were known to contain complete lists of the elements in the production system (as they were used for automated tasks relating to deployment). Sampling about 30 per cent of the architectural description and cross-checking this against the lists of deployment elements revealed a high degree of completeness, so confidence in its use as a catalogue was high.

- Allow Architectural Characteristics to be Estimated - as described earlier, the architectural description comprised a diagram and a set of tables and text for each major component within the system. While this was an effective representation for human use and was useful for impact analysis (see below) it was not all that effective for any sort of architectural quality assessment, such as energy properties of a component.

This was because of the fundamental tradeoff between the flexibility, accessibility and usability of the notation and its formality. While we were successful at enforcing conventions for the graphical models and for the representation of descriptive text and tables, it would not have been possible to persuade the development teams to use a formal, checked notation. Therefore we concluded that while the architectural description could have many uses, it would not be realistic to use it for any sort of automated architectural analysis.

- Allow Impact Analysis - the architectural description quickly proved its worth for impact analysis and helped considerably with the process of understanding the impact of proposed changes. This was primarily due to the fact that it allowed the interconnectedness of system elements to be quickly assessed, information that hadn't been easy to find before. An example of this was a small project to migrate the interface to an important internal service from a legacy RPC technology to the strategic message-based interface. The service interfaces were designed so that they could be used in parallel and the plan was to offer both and then slowly migrate users of the service to the new version. The problem with this was the time it was going to take to find all of the users of the service and so the length of time that the parallel interfaces would be needed. The model was in a late stage of development when this project started to think about migration and they were able to use it to discover nearly all of the clients of their service. So rather than relying on a service provider keeping track of the users of the service, the model provided a structure to allow the users of the service to declare their interest in the services they used, which was a much more effective approach.

- Communicate - the architectural description was quickly recognised to be a comprehensive knowledge base of the system's design information and so helped inter-team communication (when people in one team could use it to understand another team's subsystem). An example of the model being used for this sort of collaboration was when a new application, which had been acquired as part of the acquisition of another firm, was being integrated into the existing application as a new subsystem. The existing models helped the new team see how existing subsystems were integrated with each other and the model that the new team created of their subsystem helped the existing teams to understand what was being added to the system

and how it might be used. The architectural description also acted as a single place where further information could be gathered. As mentioned earlier, the architects involved in creating the architectural description moved onto other work soon after its initial creation, but it does appear to have continued to be used, to grow and to evolve, suggesting that it did fulfil this role. Eventually, it was adopted by the Production Services team in the firm, due to the value that they got from having up to date descriptions of the structure and dependencies of each application, for support tasks.

We judged the project to have met the most of the goals we set for ourselves and in particular we met the goal to create a useful shared architectural description and this was confirmed as it became a useful resource within the organisation, within a couple of months, acting as a centralised and standardised source of design information for the system.

Given the relative success of this project, it is natural to ask how generally applicable its results are and how repeatable it is likely to be. Given what we learned during the project, particularly the fact that the specialised nature of the notation was a key factor in its success, we feel that these lessons may well have general applicability but only in the broad sense. People like to be guided, they like familiar tools and techniques and they are unwilling to learn and use formal languages for design and architectural description. However, the specific tools or techniques that work will be specific to each environment and people in different environments will have different levels of enthusiasm for learning new approaches. However, when trying to get a significant amount of work done by people who are agnostic to the approach, familiarity and accessibility appear to help greatly with acceptance.

Based on our experience, the specific suggestions that we would make for future modelling languages are as follows:

- Create a language that is specific to a domain (e.g. real-time control systems or enterprise information systems) and ensure that it contains the type of modelling elements needed in that domain. Modelling languages also need to be easily extensible by their users, rather than modelling language experts, to allow missing

element types to be added. Of course specialising a language limits its possible user community but conversely, that user community is more likely to find a language that matches their problems useful and so is more likely to use it.

- Spend time creating a rich visual notation that communicates as much as possible using the shape, line, fill and other visual aspects of the notation. This makes diagrams much easier for people to understand.

- Keep modelling languages as simple as possible so that people can start using them quickly without a great deal of training. We have observed that modelling language constructs with complex or obscure semantics are rarely used correctly if they are used at all.

- Consider how people will use the language and what they will need in terms of tools and facilities for structuring and managing large models. Again simple tools (and ideally, extensions to tools that people are already likely to be familiar with) are much more likely to be successful than tools that require a lot of training and experience to use.

- As well as the language and tools, develop the materials that people will need in order to successfully adopt the language for practical use. This includes task-oriented training material, quick reference guides and plenty of samples which show the value of the language in use and provide people will examples of how to use it well (which they will almost certainly copy).

It is worth noting that our experiences from this work and our resulting suggestions are similar to the conclusions of a major academic survey of practitioner requirements for ADLs [109], which suggests that these lessons and requirements reflect the needs of a significant number of industrial software architects.

Beyond the experience we have gained in applying architectural description techniques to a large-scale problem, the particular notation and approach used in this paper may be of use to others but as explained earlier in the paper, this wasn't a goal of the project. While some of the aspects of the notation invented will be generally familiar (e.g. servers that are driven by messaging) the overall set of element types is specific to one environment and may well not be directly useful elsewhere. We did not set out to contribute yet another

general purpose ADL to the world, so reuse of the notation was not considered during its development. We report this project in order to describe a successful application of the concepts of architectural description notations, to record the factors that we believe made the project successful and to capture the lessons learned and conclusions drawn from the experience.

That said, it is possible that some interesting future work might be possible to investigate if this ADL, or a simple variant of it, might of wider applicability and indeed whether tool support could be created at reasonable cost by the reuse of existing ADL support environments. We explore this further in Section 9.2.1.

3.12 Conclusions

We wanted to investigate the practicality of using an ADL to describe a complex architecture to better understand it, and potentially allow its key quality attributes, such as performance and energy efficiency, to be analysed and understood. To this end, we worked with an organisation in the financial services industry and created an architecture description for a large existing enterprise system. In order to achieve this within acceptable cultural and time constraints, existing ADLs from the research domain proved to be unsuitable and so a simple, very specific architecture description language was defined in order to make the process of capturing the architecture description as simple and prescriptive as possible.

While it was not clear at the outset whether this approach would be successful, our minimal ADL proved to be a helpful and effective tool for capturing this specific architecture description in an entirely industrial context. The result was that a large, unified, architecture description was created, something that the organisation had not achieved before, and this allowed new understanding of the system to be gained.

The factor that appeared to make the approach generally successful was the focus on describing the specific structures in the system of interest, rather than trying to create a general-purpose approach, which would be effective for other uses too. Other factors which contributed to the success of the approach were its simplicity (which traded sophistication for accessibility), a carefully designed, consistent graphical notation, the

availability of a large amount of tutorial and reference material to guide document authors, and the use of very familiar tools, which users of the notation were already familiar with.

What the case study did not achieve was an architectural description that captured the system in a sufficiently precise manner to allow its quality attributes, such as energy efficiency, to be analysed and estimated. This was the result of a fundamental trade-off that we discovered, between creating an ADL simple and specific enough for mainstream practitioners to use, versus using an ADL from the research domain that, while rigorous, would have been too general purpose, abstract and complex for mainstream practitioner use.

We therefore reluctantly concluded that we were very unlikely to be successful in helping practitioners to understand the energy properties of their system by using an ADL as a fundamental part of the approach. If we were to use an ADL from the research domain (such as ACME [60] or xADL [86]) it is possible that we could have successfully extended it to allow energy properties to be captured and analysed. However our experience, as reported in this chapter, made it clear that this would not be used by practitioners, so defeating the object of the exercise. On the other hand, if we used a very simple and tailored ADL, such as the one that was successfully used in this case study, then the resulting architectural description would not be sufficiently precise and standardised to allow the analysis of architectural quality attributes such as energy efficiency.

In summary, while we achieved a great deal with our pragmatic, simple and context-specific ADL, we concluded that neither using an existing research ADL nor an extension to our lightweight ADL would be a practical approach for helping architecture practitioners understand and optimise the energy characteristics of their systems.

Chapter 4

Prioritising Architectural Effort

4.1 Introduction

In our practice in the field of software architecture, we have noticed and experienced how complex it is for software architects to prioritise their work. The software architect's responsibilities are broad and, in principle, they can be involved in almost any technical aspect of a project from requirements to operational concerns. In practice, this makes it difficult for an architecture practitioner to prioritise their effort to achieve quality properties like energy consumption, which acquiring stakeholders and end-users rarely prioritise explicitly due to their lack of immediate visibility. Other important quality properties that suffer from similar prioritisation problems include security [32] and performance and availability [135].

In the case of energy efficiency in particular, a previous survey of practitioners [20] revealed that the vast majority of them (83%) admit that they do not prioritise energy efficiency as a key quality attribute of their systems but that a clear majority of them (67%) thought that energy would be a key architectural concern in coming years. There are likely to be a number of reasons for the current situation, including lack of stakeholder focus, the perception of limited tools and techniques for application energy usage, and the relative complexity of addressing application energy consumption, as it requires a multi-disciplinary approach to be effective.

However, our experience with other quality properties, particularly scalability and security, suggests that even when relevant tools exist and there is general acceptance that the qualities are important, architects (and development teams more generally) often find it difficult to prioritise these concerns against short term priorities such as feature completion, fixing defects and performance fixes that address usability concerns. While it is important to address these immediate concerns, they are short-term solutions, which do not improve the fundamental quality of the system that determines its long term viability.

While addressing fundamental, rather than short-term, concerns is challenging, we observe that successful experienced software architects appear to be able to do this. These experienced architects are good at focusing their effort for maximum long-term effect, and manage to create performant, secure, highly available systems, and we hope in the future, energy efficient systems too. However not all architects have this skill and anecdotally we observe that inexperienced architects often find prioritising effort very difficult. This situation led us to wonder how the experienced architects achieve their balance between immediate and long-term concerns. As discussed in Section 2.2, there is little to guide them in the existing research literature, although they may use simple techniques like ad-hoc prioritisation or numeric grouping. They may also use generic time management techniques (like [4]) but we were interested in whether there are common role-specific heuristics which could be taught to new architects.

We decided to investigate this via a questionnaire-based study of a group of experienced architects. We discovered that there are common heuristics which experienced architects use to prioritise their work and we created a model to capture them. We then validated the model via an online questionnaire with a much wider group of practitioners and refined the model based on their input.

In this chapter, we explain the approach we took and present both the initial model that we created from the results of the interview process and the final, refined, model that we created after the validation process. The contribution of the work is not specifically the heuristics in the model, indeed most of them are quite familiar to experienced practitioners, but rather the organisation of the heuristics and the validation that they are used by experienced practitioners to guide their work. We believe that this makes the model a useful reminder for experienced practitioners and an effective teaching aid for new architects

who are learning how to perform the role.

4.2 Research Method

When planning this research, we selected a qualitative research approach because we needed to explore the "lived-experiences" of expert practitioners by asking them questions to encourage reflection and insight [98] rather than assessing performance or alignment with specific practices via quantitative means.

The process was organised into four distinct stages.

Stage 1 gathering primary data using semi-structured interviews with practitioners.

Stage 2 analysis of the primary data and creation of a preliminary model.

Stage 3 validation of the preliminary model via a structured online questionnaire, completed by practitioners in relevant architecture roles (primarily software, solution and enterprise architects).

Stage 4 analysis of the validation data and refinement of the preliminary model into a final, validated model.

We chose to gather our primary data using semi-structured interviews, where we provided the interviewees with a written introduction to the question we wanted to answer and then some specific questions to start their thought processes.

The analysis of the primary data was performed using a simple application of Grounded Theory as it is a suitable method for theory building, to understand the relationships between abstract concepts [29], which described our situation and needs very closely. We performed initial coding on the primary data and then refined this with a more focused coding exercise. As suggested in [98], the process of collection and analysis was a parallel, iterative process, rather than a linear one with fixed phases.

This exercise produced a set of themes that classify the heuristics that the architects use, as well as the heuristics themselves. A heuristic had to be mentioned by at least three of

the participants (which represented a third of them) for us to consider it significant enough to be included in the model. We combined the themes and heuristics to form a simple model (the "preliminary model") of how experienced architects go about prioritising their effort.

Once we had the preliminary model available, we published it at a research conference [182] and via a LinkedIn post (https://www.linkedin.com/pulse/focusing-software-architects-attention-eoin-woods) and created an online questionnaire [61] aimed at architecture practitioners to allow them to evaluate and comment on the usefulness of the model. We publicised the survey via LinkedIn, Twitter and via direct email to our network of architecture practitioners.

We received 84 responses to the survey, containing answers to our closed-ended questions to evaluate the usefulness of the model and also received answers to open-ended questions in 50 of the responses. We used the closed-ended questions to evaluate the usefulness of the model and analysed the open-ended responses to identify themes which needed to be addressed by the model.

The model was validated strongly across respondents from different locations, with different amounts of experience and from different architectural specialisations. Additionally, a small number of themes emerged from the answers to the open-ended questions. These themes for improving the model were used to revise and extend it slightly, creating an improved final version that reflected the input from the respondents.

A description of the four stages of the research method is presented below, along with the final version of the architectural effort prioritisation model.

4.3 Stage 1 - The Initial Study

Our primary data gathering was performed using a semi-structured, face-to-face survey of 8 experienced software architecture practitioners working across 4 countries.

We found the participants by approaching suitable individuals from our professional networks. We were looking for practitioners who had a minimum of 10 years' professional

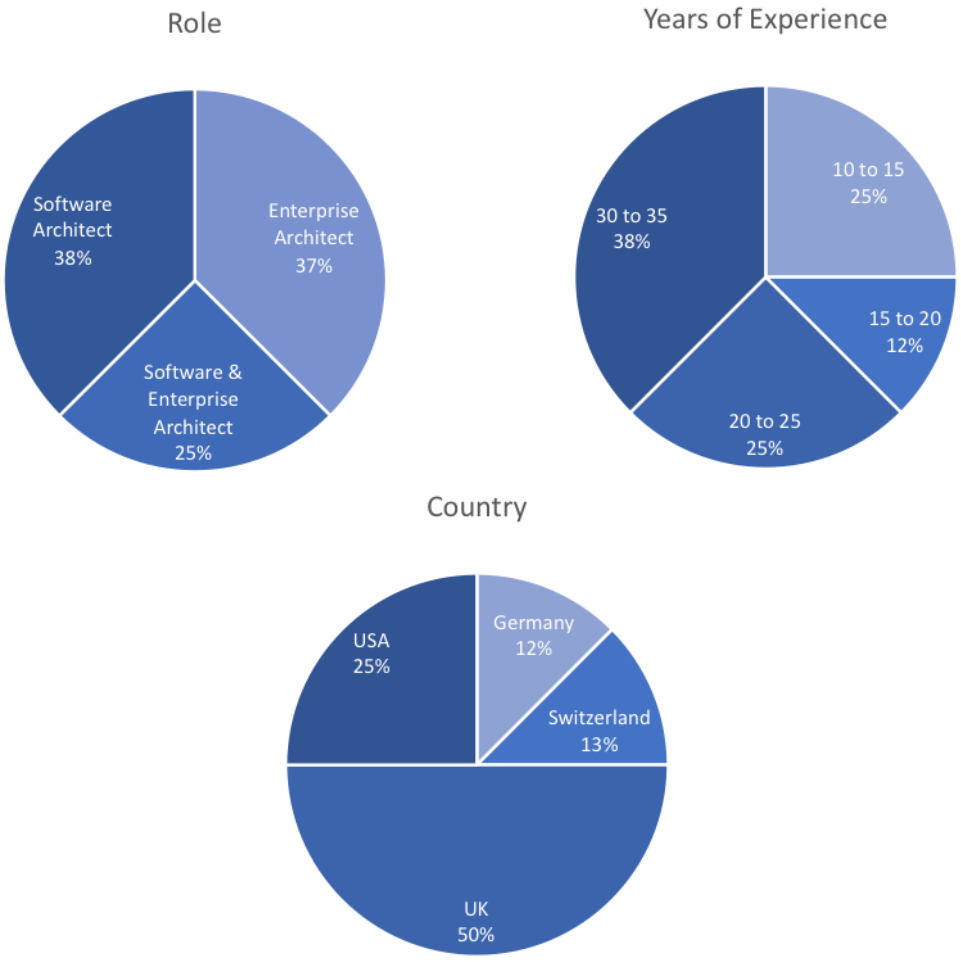


FIGURE 4.1: Study Participants (8 in total)

experience and who worked as architects in the information systems domain (rather than architects from –for example –embedded systems).

We focused on the information systems domain because we know from experience that working practices differ between professional domains like information systems and embedded systems. Hence, we thought it was more likely that we could create a useful model if we limited ourselves to one broad domain, at least initially.

We deliberately selected candidates that we knew differed from each other in organisation, specialisation and geography to get a reasonably diverse population and avoid obvious sample bias (we discuss the threat of sample bias further in Section 4.7).

Some characteristics of the participants in the study are summarised in the graphs in Figure 4.1. As can be seen, they represent a range of experience, role type and country.

All of the members of our initial practitioner set had over 10 years of post-graduate experience and some had over 30 years of experience, so ensuring that they all had a significant amount of professional practice upon which to base their answers.

We deliberately selected software and enterprise architects because this is whom the model was primarily aimed at serving.

We approached individuals in a number of countries to try to minimise the risk that we would reflect practice only in one country, although we did not manage to gain representation from beyond North America and Europe.

We used a semi-structured interview format with a written introduction to the question which each interviewee read before being asked a standard set of open-ended questions which explored how they went about prioritisation of architecture work and any specific factors that they used to guide them.

The question we asked was "how can the architect concentrate their attention so that they are most effective?" The more specific questions used to stimulate the thought process were:

- How do you go about this in your work?
- What factors do you consider when prioritising your attention?
- Do you consider what to focus on? Or what not to focus on?
- For example, how do you prioritise architectural governance compared to other aspects of the project?

The interviewer asked additional questions to understand the answers fully or to encourage the interviewee to add more detail or fill in ambiguous aspects of the answer.

The process of initial coding of the primary data resulted in 25 items, which could be associated with at least one of the interviews. A further focused coding process revealed that there were 9 underlying heuristics which appeared to be significant to the participants in the study. Then, a further analysis iteration lead to the identification of three categories of prioritisation heuristic which we use to structure our model.

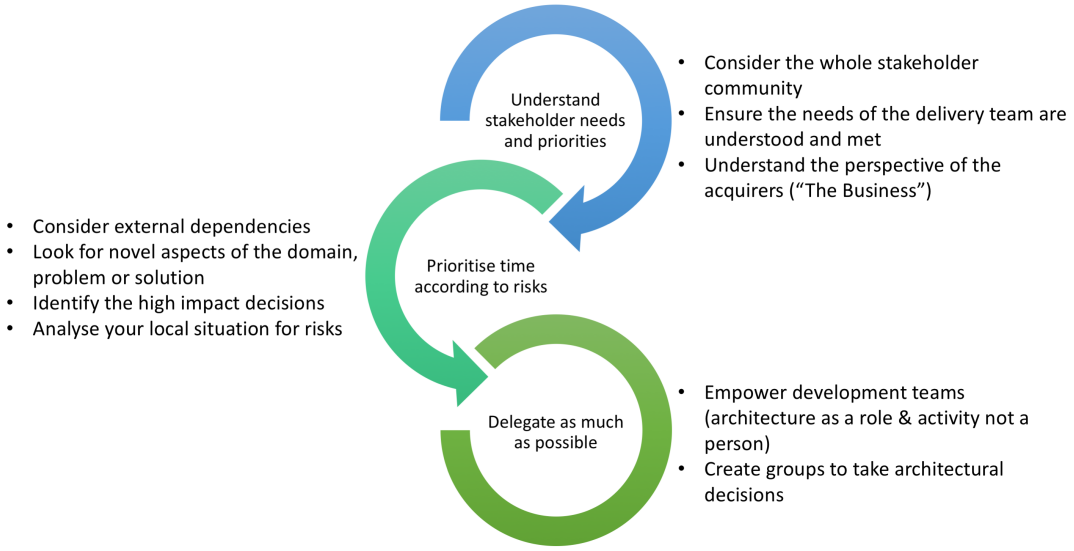


FIGURE 4.2: Preliminary Model for Focusing Architectural Attention

4.4 Stage 2 - Preliminary Model for Prioritising Architectural Effort

4.4.1 The Preliminary Model

Our preliminary heuristic model for focusing architectural effort is shown in Figure 4.2.

The three categories of heuristic that the study revealed were: first, the need to focus on stakeholder needs; second, the importance of considering risks when deciding on where to focus effort; and finally the importance of spending time to achieve effective delegation of responsibilities. These categories form the structure of our model and remind the architect of the general ways in which they should prioritise their efforts. The categories and heuristics are explained in more detail in section 4.4.2.

It is important to understand the nature of this model and how it should be used. It is not a prescriptive process for architects to follow or a process for developing an architecture. This model is an aide memoir to organise and present a set of heuristics that experienced architecture practitioners appear to find useful when prioritising their work. While we believe this to be a useful model to teach trainee architects and a useful reminder for

experienced architects, it is necessary to apply the model in a context-sensitive manner, within whatever method that the architect is using to develop software architectures.

4.4.2 Content of the Preliminary Model

4.4.2.1 Understand the Stakeholder Needs and Priorities

The first theme which emerged strongly in our study was focusing on the needs and priorities of the stakeholders involved in the situation. The principle that architecture work involves working closely with stakeholders is widely agreed upon [21, 153] and this theme reinforces that. Architects need to focus significant effort to make sure that stakeholder needs and priorities are understood to maximise focus on the critical success factors for a project and maximise the chances of its timely completion. Based on the study, three specific heuristics to achieve this were identified:

- *Consider the whole stakeholder community*. Spend time understanding the different groups in the stakeholder community and avoid the mistake of just considering obvious stakeholder groups like end-users, acquirers and the development team. As the architecture methods referenced above note, ignoring important stakeholders (like operational staff or auditors) can prevent the project from meeting its goals and cause significant problems on the path to production operation.

- *Ensure that the needs of the delivery team are understood and met*. Spend sufficient time to ensure that the delivery team can be effective. What is the team good at? What does it know? What does it not know? What skill and knowledge gaps does it have? These areas need attention early in the project so that architecture work avoids risks caused by the capabilities of the team and that time is taken to support and develop the team to address significant weaknesses.

- *Understand the perspective and perceptions of the acquirers of the system*. Acquirers are a key stakeholder group who judge its success and usually have strategic and budgetary control, so can halt the project before delivery if they are unhappy. Specifically addressing this group's needs, perceptions and concerns emerged as an important factor for some of the experienced architects in our study. Acquirers

are often distant from the day-to-day reality of a project and need clear communication to understand their concerns and ensure that they have a realistic view of the project.

4.4.2.2 Prioritise Effort According to Risks

During a project, an effective approach to prioritising architectural attention is to use a risk-driven approach to identify the most important tasks. If the significant risks are understood and mitigated, then enough architecture work has probably been completed. If significant risks are open, then more architecture work is needed. The specific heuristics to consider for risk assessment are:

- *Consider external dependencies*. Understand your external dependencies because you have little control over them and they need architectural attention early in the project and whenever things change.
- *Look for novel aspects of domain, problem and solution*. Another useful heuristic, from the experience of our study participants, is to focus on novelty in your project. What is unfamiliar? What problems have you not solved before? Which technology is unproven? The answers to these questions highlight risks and the participants in our study used them to direct their effort to the most important risks to address.
- *Identify the high impact decisions*. Prioritise architecture work that will help to mitigate risks where many people would be affected by a problem (e.g. problems with the development environment or problems that will prevent effective operation) or where the risk could endanger the programme (e.g. missing regulatory constraints).
- *Analyse your local situation for risks*. Consider the local factors unique to your situation, which you will be aware of due to the knowledge you have of the domain, problem and solution. It is impossible to give more specific guidance on this heuristic as every situation is different, but the participants in our study noted the importance of "situational awareness" [178] that allows the architect to find and address the risks specific to the local environment (perhaps due to organisational factors, specific technical challenges, domain complexities or business constraints).

4.4.2.3 Delegate as Much as Possible

Delegation was an unexpected theme that emerged from our study. The architects who mentioned this theme viewed themselves as a potential bottleneck in a project and delegation and empowerment of others was a way to minimise this. Delegation was also seen as a way of freeing the architect to focus on the aspects of the project that they had to focus on rather than all the other aspects that they could possibly get involved in.

The general message of this theme is to delegate as much architecture work as possible to the person or group best suited to perform it, to prevent individuals becoming project bottlenecks, allow architects to spend more time on risk identification and mitigation, and to spread architectural knowledge through the organisation. The heuristics that were identified to help achieve this are:

- *Empower the development teams*. To allow delegation and work sharing, architects need to empower (and trust) the teams they work with. This allows governance to become a shared responsibility and architecture to be viewed as an activity rather than something that is only performed by one person or a small group. This causes architectural knowledge, effort and accountability to be spread across the organisation, creates shared ownership, reduces the load on any one individual and prevents reliance on a single individual from delaying progress.

- *Create groups to take architectural responsibilities*. A related heuristic is to formalise delegation somewhat and create groups of people to be accountable for specific aspects of architectural work. For example, in a large development programme, an architecture review board can be created to review and approve significant architectural decisions. Such a group can involve a wide range of expertise from across the programme and beyond, so freeing a lead architect from much of the effort involved in gathering and understanding the details of key decisions, while maintaining effective oversight to allow risks to be controlled and technical coherence maintained. Similarly, a specific group of individuals could be responsible for resilience and disaster recovery for a large programme, allowing them to specialise

and focus on this complex area, and allowing a lead architect to confidently delegate to them, knowing that they will have the focus and expertise to address this aspect of the architecture.

4.5 Stage 3 - Validating the Preliminary Model

4.5.1 The Questionnaire

Once we had a preliminary model, we wanted to validate its usefulness with a much larger group of experienced practitioners. This was conducted using a structured online questionnaire. The questionnaire asked the respondents to read the model and then comment on its credibility and usefulness. We asked both closed questions, that asked respondents to rate the model on 5 point scales, and open-ended questions that allowed the respondents to consider whether there were aspects of focusing attention that we had missed and also to collect general comments on the model. Finally, we asked some closed classification questions to allow us to understand who had completed the survey while preserving their anonymity if desired.

We asked three closed-ended questions to find out whether the respondent thought that the model was credible and useful. These questions and the possible responses were:

Q1 "Is this model similar to how you focus architectural attention in your work already?" (Not at all similar / Not Very Similar / Somewhat Similar / Quite Similar / Very Similar)

Q2 "Would you find this model helpful in guiding architectural attention for maximum benefit?" (Definitely Not / Probably Not / Possibly / Probably Yes / Definitely Yes)

Q3 "Are the areas of risk mentioned in the "Prioritise time according to risks" activity valuable?" (Definitely Not / Probably Not / Somewhat / Probably Yes / Definitely Yes)

The open-ended questions that we asked were:

Q4 "Are there other general areas of risk that should be added to 'Prioritise time according to risks' that would be applicable to most (information) systems and environments? If so please list and briefly explain them."

Q5 "Are there any significant factors missing from the model which you use to focus your architectural work?"

Q6 "Do you have any other comments on the model or the survey"

The closed-ended questions we asked to allow us to classify the respondents and their possible answers were:

Q7 "What environment do you work in?"

- Industry (developing systems, consultancy and related work)
- Industrial Research (working in a research environment for an industrial employer)
- Academic (teaching or researching in university or similar environments)
- Other (please specify)

Q8 How many years of post-graduation experience do you have?

- 1-5 years
- 5-10 years
- 10-15 years
- 15-20 years
- More than 20 years

Q9 What is your main job role?

- Software Architect
- Enterprise Architect
- Software Designer
- Researcher

- Other (please specify)

Q10 Where in the world are you based?

- North America
- South America
- Europe (including the UK)
- Middle-East and Africa
- Asia-Pacific
- Other (please specify)

We also invited them to leave an email address if they wanted to be informed of the outcome of the study and provided a free text box for any final comments or questions. We did not view the email address and the final text box as survey data for the purposes of analysis.

Having trialled the questionnaire ourselves and with two other individuals, we expected most respondents to take 10 - 15 minutes to complete it.

4.5.2 The Respondents

To use the questionnaire to validate the model, we needed to find a suitable set of architects who could read it and complete the survey for us. We found our respondents via two main activities. Firstly, we created a LinkedIn post [1] which provided an outline of the model, explained that we wanted to validate it and whom we wanted to participate, and provided a link to the survey. This was posted from my account and so tended to appear in the LinkedIn news feed of practitioners, as my LinkedIn network contains many more practitioners than researchers. The post was further publicised via Twitter and Facebook to reach a larger audience. This step was moderately successful, with the post being viewed about 700 times and about 25 people completing the survey successfully.

To gain more responses to the survey, the second activity was a targeted email sent to individuals that we knew personally and whom we knew were practising software related

[1] https://www.linkedin.com/pulse/focusing-software-architects-attention-eoin-woods

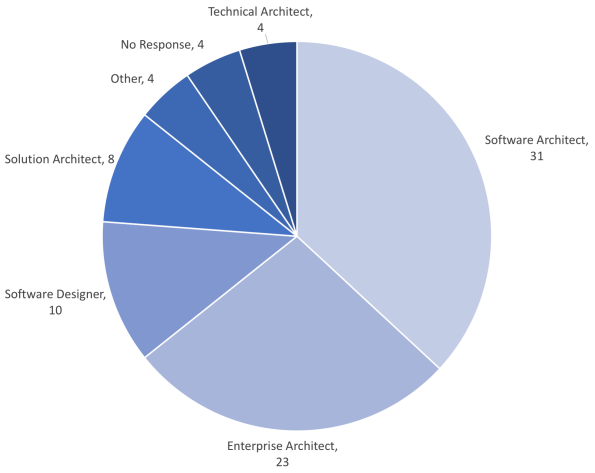


FIGURE 4.3: Respondent Roles

architects (including software, solution and enterprise architects). This second activity resulted in about 60 more responses to the survey.

In total, we received 84 completed surveys.

The job roles of the respondents are summarized in Figure 4.3. About a third of the respondents identified themselves as software architects, about a quarter as enterprise architects, about 12% as software designers, 10% as solution architects, and 5% as technical architects. Four respondents did not complete this answer and four had other job titles (a risk assessor, a technical manager and systems engineer, a project manager and a strategy consultant).

The work environments of the respondents were overwhelmingly industrial (we viewed those building systems in the public sector as part of "industry"), with only one respondent having a purely academic work environment. Five respondents did not provide an answer to this question. Of these five, two were from German IP addresses, one Swiss and two UK addresses. Four were from private ISP connections and one was from a UK public sector body, which, with another respondent who identified themselves as working in the public sector, suggested there were at least two public sector respondents.

We had some geographical distribution of respondents, as shown in Figure 4.5, although there is a clear bias towards Europe, almost certainly caused by our professional networks being centred in Europe. 55% of respondents identified themselves as coming

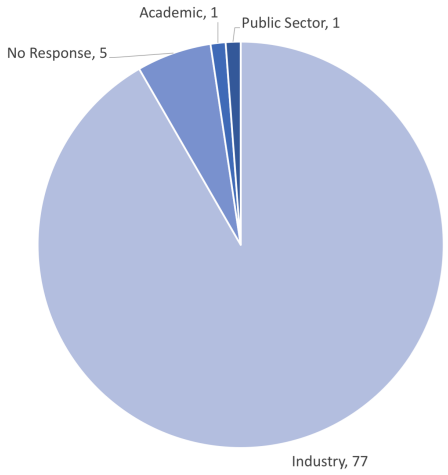


FIGURE 4.4: Respondent Work Environments

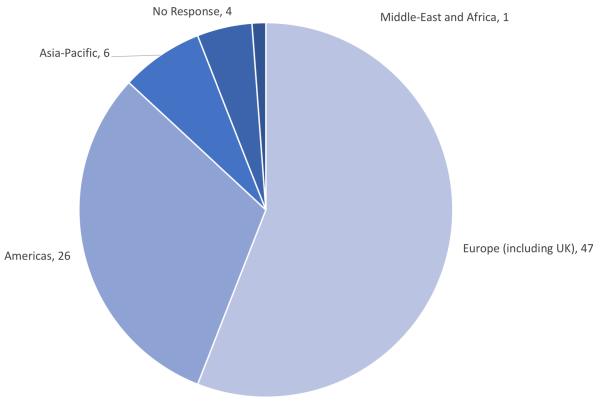


FIGURE 4.5: Respondent Geographies

from Europe, 30% from the Americas and only 7% from Asia-Pacific and a single correspondent from the Middle-East and Africa.

To delve a little deeper, we checked the geographical location of the respondents' IP addresses using the well-known "geoiplookup" command line tool [2]. Of the four respondents who did not answer the question, the IP addresses they were using were in the UK (two), Germany (one) and Switzerland (one). While this does not prove that the respondents work in those countries (they could have been travelling away from home) it does make it likely that they are from Europe, taking the European percentage to about 60%.

The result of using the respondents' IP address geographical locations to find which countries the questionnaire was completed form is shown in Figure 4.6. While we must be

[2] http://geoiplookup.net

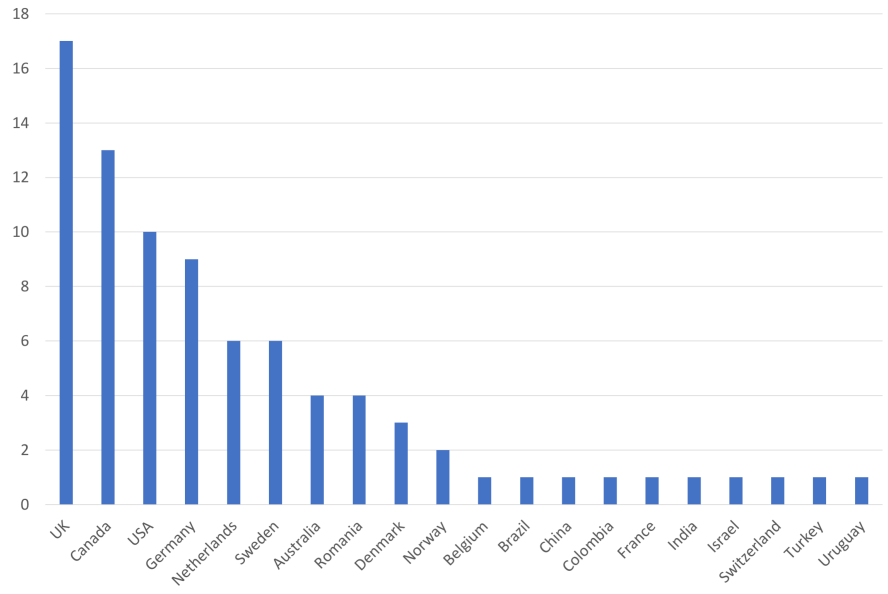


FIGURE 4.6: Respondent IP Address Locations

cautious in assuming that these geographical locations are necessarily the locations of the respondents' homes and workplaces, it is a useful cross-check on the data. We can see that 20 countries are represented, with 6 countries having 4 correspondents or more (which is roughly 5% of the survey size). These larger 6 countries are the UK, Canada, the USA, Germany, the Netherlands, Sweden, Australia and Romania. Ten of the countries (Brasil, China, Columbia, France, India, Israel, Switzerland, Turkey and Uruguay) only had one survey completed from one of their IP addresses. Just over 55% of the responses were from IP addresses located in four countries, the UK (20%), Canada (15%), the USA (12%) and Germany (11%).

We discuss the possible impact of geographical location more when we consider threats to validity in section 4.7, but we think it is fair to say that we achieved good cross-geographic participation, but still ended up with a strong bias to Western Europe and North America.

The final classification we asked our respondents for was the number of years of experience that they had. We asked this to ensure that participants had a significant amount of professional practice upon which to base their evaluation of the model and to allow us to understand if there are differences in the value of the model to architects with different amounts of experience. The degree of experiences for our respondents is summarised in Figure 4.7.

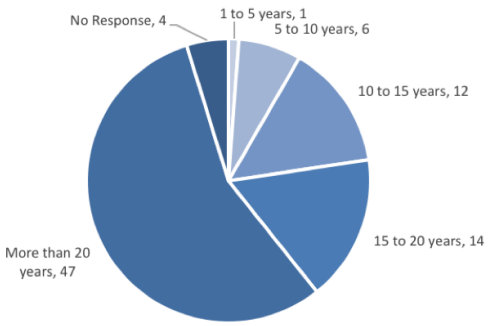


FIGURE 4.7: Years of Experience of Respondents

The obvious first impression is that our respondents were overwhelmingly experienced people, with over half of them (55%) having at least 20 years of post-graduation experience. At the other extreme only one respondent had less than 5 years of experience. About 7% had 5 to 10 years of experience, 15% had 10 to 15 years and about 17% had 15 to 20 years. Four of our correspondents did not answer this question.

4.5.3 The Results

As mentioned earlier, we structured the questionnaire into two distinct parts, the closed-ended questions that asked people to rate the usefulness of the model and the open-ended questions that asked whether we had missed important risk areas to use with the prioritisation heuristic or whether there were any significant aspects of prioritisation that were completely missing from the model.

In this section, we review and analyse the responses to the closed-ended questions in the survey. The first question we asked was to find out if the model was similar to how experienced architects already focused their attention, which would suggest that the model was credible and, if we assume that experienced architects are probably effective, a useful guide for less experienced architects, early in their career. The responses we received from all of the respondents are summarised in Figure 4.8.

Considering all of the responses, the model validates quite strongly against the participants' existing practice, with 75% of respondents stating that it was "very similar" or "quite similar" to their existing approach for focusing attention in their architecture work. 20%

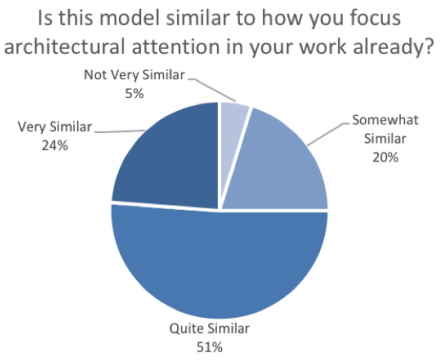


FIGURE 4.8: How Similar the Model is to Existing Practice

TABLE 4.1: Similarity of the Model to Practice by Experience Level

	Not at All Similar	Not Very Similar	Somewhat Similar	Quite Similar	Very Similar
1 to 5 Years				1 (100%)	
5 to 10 Years		1 (17%)	2 (33%)	1 (17%)	2 (33%)
10 to 15 Years		1 (8%)	3 (25%)	7 (58%)	1 (8%)
15 to 20 Years			1 (7%)	11 (79%)	2 (14%)
More than 20 Years		2 (4%)	10 (21%)	22 (47%)	13 (28%)

said it was "somewhat similar", 5% said it was not very similar to how they worked, and no respondents replied, "not at all similar".

Given that we were interested in attracting experienced architects to validate the model, we were also interested in finding out whether the number of years of experience altered their view of how similar the model was to how they worked. The summary of this information is shown in Table 4.1.

The table shows how many respondents, grouped by years of experience, rated the model at each similarity level, compared to their own approach to prioritising their work. The 4 participants who did not indicate their experience level are excluded from this table. The percentage values are the percentage of the participants in the current row that the value represents.

We are primarily interested in the top three groups, which is architects with at least 10 years of experience. What we can see from this data is that the degree of similarity between the model and the architect's current practice does vary between these three groups, with 66% of the 10 - 15 year group rating it as "quite similar" or "very similar", 93% of the 15 - 20 years group rating it at this level and 75% of the 20+ year group rating

TABLE 4.2: Helpfulness of the Model by Experience Level

	Definitely Not	Probably Not	Possibly	Probably Yes	Definitely Yes
1 to 5 Years					1 (100%)
5 to 10 Years		1 (17%)	3 (50%)	2 (33%)	
10 to 15 Years			4 (33%)	5 (42%)	3 (25%)
15 to 20 Years			4 (29%)	5 (36%)	5 (36%)
More than 20 Years		2 (4%)	10 (26%)	23 (43%)	12 (26%)

it in this way. So, all three groups validate quite strongly, but it seems to reflect most strongly how the 15+ year groups work.

The second question moved on to try to establish, whether the respondents thought that the model would be useful in practice. A summary of the answers to this question across all responses is shown in Figure 4.9.

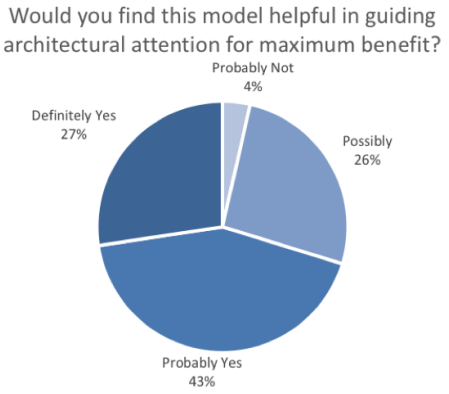


FIGURE 4.9: Helpfulness of the Model

Across all of the respondents, 70% said that it was definitely or probably useful, which we interpret as a strong overall validation of the model. The remaining respondents mainly stated that they might "possibly" find it useful. Only 3 respondents said "probably not" and none said, "definitely not". We were interested in how the utility of the model might vary by the experience of the respondent, so we performed a similar analysis by experience group to that performed for the previous question. The results are shown in Table 4.2.

Again, focusing on the architects with at least 10 years of experience, 67% of the 10 to 15 year experience group believe the model is "probably" or "definitely" useful, 72% of the 15 to 20 year group also rate it at this level, while 69% of the most experienced, 20+ years of experience group, believe it is "probably" or "definitely" useful. A strong majority of respondents in all three groups appear to find the model useful, with the strongest validation coming from the 15 - 20 years of experience group.

TABLE 4.3: Value of Risk Factors by Experience Level

	Definitely Not	Probably Not	Somewhat	Probably Yes	Definitely Yes
1 to 5 Years					1 (100%)
5 to 10 Years		1 (17%)	1 (17%)	2 (33%)	2 (33%)
10 to 15 Years	1 (8%)			5 (42%)	6 (50%)
15 to 20 Years			3 (21%)	6 (43%)	5 (36%)
More than 20 Years		2 (4%)	9 (19%)	17 (36%)	19 (40%)

Finally, we wanted to check that the areas of risk we had identified as important within the "prioritise time according to risks" heuristic were valuable to a practising architect. The results of the corresponding question in the survey across all respondents are shown in Figure 4.10.

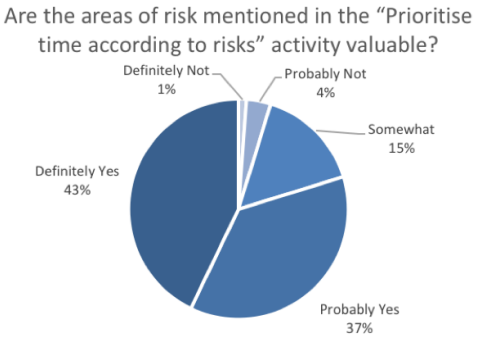


FIGURE 4.10: Validation of Risk Prioritisation Areas

In this case we did have one very strong negative opinion ("definitely not") but this was a single individual (an enterprise architect in the 10 - 15 years of experience group, who commented in the open-ended questions that he did not believe that it was possible to define general software development risks in a useful way).

Beyond this response, 80% of respondents believe that the areas of risk were "definitely" or "probably" valuable, suggesting that this aspect of the model should be of value to many practitioners.

Again, we analysed the responses by the experience level of the respondent, which is shown in Table 4.3.

Looking at the architects with more than 10 years experience, we still see a high degree of validation (92% of 10 to 15 year experience respondents, 79% of 15 to 20 year respondents and 76% of respondents with more than 20 years of experience saying "probably yes" or "definitely yes") but there is a larger range of opinion than before.

TABLE 4.4: Usefulness of the Model by Role Type

	Definitely Not	Probably Not	Possibly	Probably Yes	Definitely Yes
Software Architect		1 (3%)	11 (35%)	13 (42%)	6 (19%)
Software Designer		1 (10%)	2 (20%)	5 (50%)	2 (20%)
Solution Architect			2 (25%)	5 (63%)	1 (12%)
Technical Architect				2 (50%)	2 (50%)
Enterprise Architect		1 (4%)	4 (17%)	9 (39%)	9 (39%)

We had the single individual who strongly disagreed with the risk factors being valuable in the 10 - 15 year group, 3 respondents in the 15 to 20 years of experience group only feeling that they were "somewhat" valuable and 11 respondents with more than 20 years of experience stating that the factors were "somewhat" or "probably not" valuable.

This said, we still feel that the degree of validation that the model received across all of the experience levels indicates that the model has a high possibility of being useful to at least a majority of practitioners.

We were also interested to investigate if the key question of how useful the model was would vary significantly between our different respondent groups, particularly by job family and geography, which might suggest that the model aligned better with certain types of architecture work or practice in certain parts of the world.

We have analysed the responses to the question "Would you find this model helpful in guiding architectural attention for maximum benefit?" by role type and the results are shown in Table 4.4 and Figure 4.11.

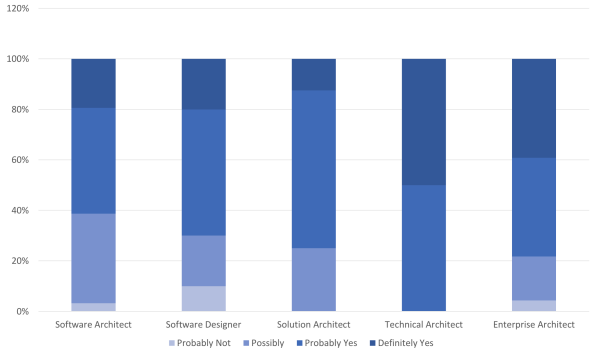


FIGURE 4.11: Usefulness of the Model by Role Type

As can be seen, all of the respondent role groups validated the model as "probably" or "definitely" useful, with the lowest validation (interestingly) coming from the Software Architect group, where 61% of the respondents indicated this level of agreement (although

TABLE 4.5: Usefulness of the Model by Geography

	Definitely Not	Probably Not	Possibly	Probably Yes	Definitely Yes
Americas			6 (23%)	10 (38%)	10 (38%)
Asia-Pacific			3 (50%)	3 (50%)	
Europe (inc. UK)		3 (6%)	11 (23%)	22 (45%)	11 (23%)
Middle-East & Africa			1 (100%)		

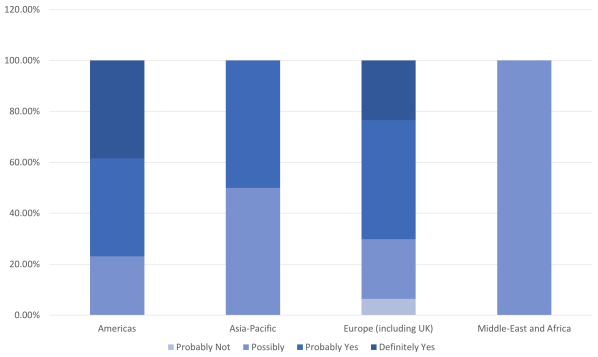


FIGURE 4.12: Usefulness of the Model by Geography

nearly all of the remaining respondents were neutral - "possibly" - rather than negative). In the Software Designer group 70% of respondents, in the Solution Architect group 75%, in the Technical Architect group 100% and in the Enterprise Architect group 78% of respondents considered the model to be "definitely" or "probably" useful.

Turning to the possible influence of geographical location, we analysed the same question as to the usefulness of the model by the respondents, by the geographical location that the respondents told us they were from. The results are summarized in Table 4.5 and Figure 4.12.

In this analysis, we have included all of the regions, but in reality, the data for the Middle-East and Africa and Asia-Pacific is difficult to use with confidence as there were very few respondents from these regions (1 from ME&A and 6 from APAC). Therefore, while recognising the importance of these regions and their contribution to contemporary software engineering, we focus on Europe and the Americas for the purposes of this specific analysis.

We had 26 respondents from the Americas and 69 from Europe (including the UK). Both of these contributions are large enough to be significant, being 31% and 82% of the total responses, respectively.

Of these significant geographical groups, the group from the Americas validated the usefulness of the model more strongly, with 76% of respondents indicating that the model was "definitely" or "probably" useful. For the European group, 68% of the respondents indicated that the model was "definitely" or "probably" useful.

We interpret this data as suggesting that the model validates well in both the Americas and in Europe and should be useful in both regions. The data we have suggests that it should also be useful in Asia-Pacific as 50% of respondents indicated it would be useful, while in contrast, it may not be useful in ME&A as our single respondent from that region indicated it was "probably not" useful; However, we do not have enough respondents from these regions to draw meaningful conclusions. To investigate the usefulness of the model in APAC and the Middle-East and Africa would require a further study.

In summary, having analysed the answers to the closed-ended answers in our survey, we conclude that our model is likely to be credible and useful in Europe and the Americas and broadly aligns with the prioritisation approach used by many experienced architects in those regions. We view this as a successful validation of the model; However, we were also interested in how the model could be improved and so we used the responses to the open-ended questions in the survey to find themes which we could include in a refined model.

4.5.4 Analysing the Open-Ended Responses

As explained earlier, we asked two important open-ended, questions, Q4, to identify missing risk factors from the "prioritise time according to risks" heuristic ("are there other general areas of risk that should be added to "prioritise time according to risks" that would be applicable to most (information) systems and environments?") and Q5, to ask whether we had missed any important aspect of the model entirely ("are there any significant factors missing from the model which you use to focus your architectural work?").

We had 44 responses to Q4, about missing aspects of the model, and 51 responses to Q5, about missing areas of risk.

Given the nature of these responses, we again used grounded theory style analysis to analyse them, coding each one initially using straightforward, descriptive labels, directly

reflecting the language used in the response, then refining this with further coding steps, to identify higher-level categories to allow the responses to be collected into meaningful groups.

For the first question, Q4, we initially coded the responses to 37 distinct categories, plus two null categories for the initial coding of "None" and "General Comment" for those responses which were present but did not specify a new risk area or just made a general comment. The responses suggested a diverse range of possible risk areas, and when we refined the coding to find common concepts, this resulted in 24 higher level categories. We attempted to refine this further but did not find further meaningful refinements as we tried further rounds of coding (we judged that we had reached "theoretical saturation" with the data). In addition, we had a very long "tail" of concepts with only a single mention in the responses, leaving us with 5 categories that had 4 responses or more: Organisational Environment (11 occurrences), Stakeholders (6 occurrences), Cost (6 occurrences), Time (4 occurrences), External Environment (4 occurrences). We chose to focus on categories with at least 4 occurrences as this represents approximately 5% of the total respondents to the survey and we judged this to be high enough to warrant consideration.

In the context of a survey with over 80 responses, none of the categories of response for missing risk factors was universally viewed as important, but we felt that it was important to reflect the fact that these four factors had been independently identified by a number of people. Therefore, we decided to integrate these new factors into the refined version of the model.

For the second open-ended question, Q5, on missing aspects of the model, we initially coded the responses into 43 distinct categories, again plus "None" and "General Comment". As we continued with the process of refining the coding further, we ended up with 26 higher level categories. As with the responses to Q4, many of the categories were only mentioned once and only four were mentioned 4 times or more: Team Effectiveness (10), Benefits (7), Stakeholders (6) and Time (5). Of these factors, "Stakeholders" are already a significant factor in the model and the comments provided in these cases were suggesting a particular emphasis on certain stakeholders or method of dealing with stakeholders. The specific suggestions were all different and stakeholder needs and priorities are already a significant part of the model, so we did not feel that a new model element was

needed, but we simply need to review the detailed comments and ensure that they are reflected in the existing elements of the model.

In this context, we felt that adding a completely new aspect to the model was a significant step and so we only wanted to consider this for aspects which had been identified as important by a significant number of respondents to the survey. On this basis, we decided to add a new element to the model to reflect the "Team Effectiveness" theme as it was the only additional candidate aspect that at least 10% of the respondents had identified as important.

Finally, we also received 37 general comments in the open-ended question at the end of the survey along with another 14 responses to the other open-ended questions which we judged to be general comments rather than specific answers to those earlier questions, making a total of 51 general comments. We do not view these responses as part of the validation of the model, but some of them did provide useful commentary on the work. Again, we used a grounded theory style coding approach to analyse the data and this resulted in 23 categories of comment. Like the other open-ended questions, most of these categories were not judged as significant because less than four respondents mentioned them.

The categories which had four or more respondents were general Positive Comments such as "nice and simple model" (14), comments on the How the Architect Should Work, such as "an architect must help implement what he/she helped to decide" (6) and comments on the Presentation of the Model, such as "this model [...] does appear to be a rather linear, and distinct, in reality it [the process] is quiet iterative and overlapping" (sic) (5).

We found all of the general comments interesting and potentially useful, but most of them did not lead us to conclude that further changes were needed to the model. The exception was the group of comments categorised as Presentation of the Model. These 5 comments suggested that the respondents interpreted our graphical presentation of the model as indicating a linear "upfront" process. In fact, we had meant to communicate exactly the opposite through our diagram, and indicate that the process is iterative and continuous, happening right through the project's lifecycle. We took this as an important

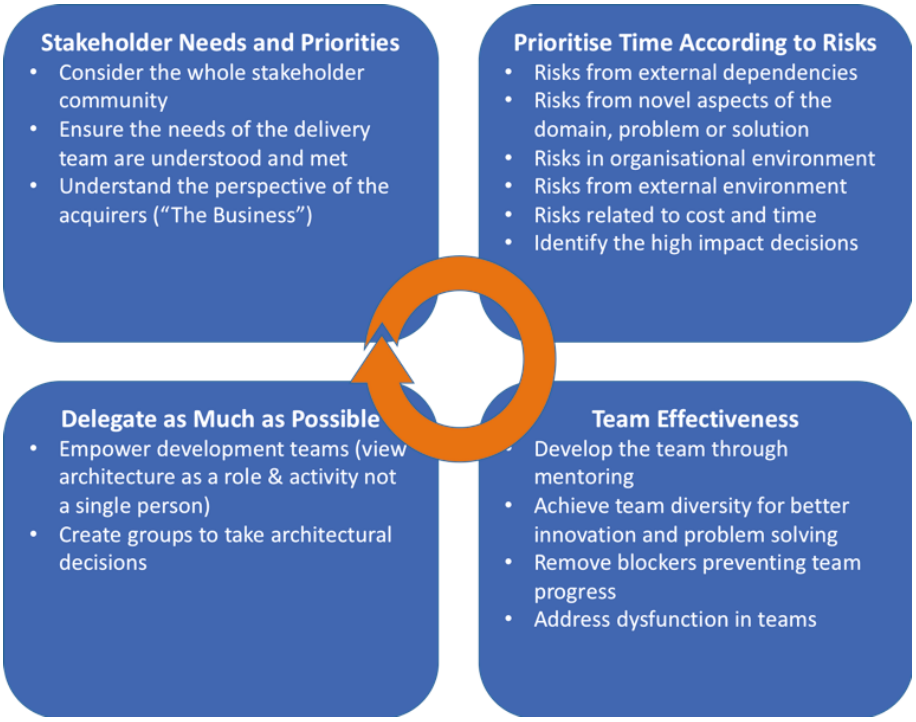


FIGURE 4.13: Refined Model for Prioritising Architectural Attention

indicator that we needed to change the graphical representation of the model and also describe its intended iterative and continuous nature more clearly in the supporting text.

4.6 Stage 4 - Refined Model for Prioritising Architectural Effort

We took the outputs of the open-ended question analysis described in section 4.5.4 and used them to add missing features of the model, improve the list of risks to suggest for time prioritisation and improve the model using the advice provided in the general comment responses to the survey. The result of these additions is a refined model for prioritising architectural effort, with an additional feature of the model, "Team Effectiveness" and a refined list of risks for time prioritisation. As mentioned earlier, we also decided to alter the graphical presentation of the model to try to emphasise that it is not a linear "process" but rather a set of activities to be performed throughout the project lifecycle. The refined model is shown in Figure 4.13.

The model is comprised of 4 aspects, Stakeholder Needs and Priorities, Prioritise Time According to Risks, Delegate as Much as Possible and Team Effectiveness. Each aspect

is a theme which our initial interviewees and the later survey respondents find useful when considering how to prioritise their architectural work. The details of each theme are described in the subsections below.

The idea of the model is to provide a guide for new architects, or an aide-memoir for experienced architects, on how to prioritise their architectural work in order to maximise its effectiveness. It is not a process or a step to be followed in an architectural "method" but rather these are themes for effective effort prioritisation that should be repeatedly considered during the lifecycle of a project. As with any set of heuristics, they can only be a generalised starting point and need to be considered, interpreted and applied in a context-specific way by the architects and teams who use them. However, they have validated well against a reasonably broad survey of experienced, practising architects and so we believe that they are a useful starting point upon which to build a personal approach for prioritisation.

4.6.1 Stakeholder Needs and Priorities

The first theme which emerged strongly in our study was focusing on the needs and priorities of the stakeholders involved in the situation. The principle that architecture work involves working closely with stakeholders is widely agreed [21, 153], and this theme reinforces that. Architects need to focus significant effort to make sure that stakeholder needs and priorities are understood to maximise focus on the critical success factors for a project and maximise the chances of its success. Three specific heuristics to achieve this which emerged from the study are:

- *Consider the whole stakeholder community*. Spend time understanding the different groups in the stakeholder community and avoid the mistake of just considering obvious stakeholder groups like end-users, acquirers and the development team. As the architecture methods referenced above note, ignoring important stakeholders (like operational staff or auditors) can prevent the project from meeting its goals and cause significant problems on the path to production operation.
- *Ensure that the needs of the delivery team are understood and met*. Spend sufficient time to ensure that the delivery team can be effective. What is the team good

at? What does it know? What does it not know? What skill and knowledge gaps does it have? These areas need attention early in the project so that architecture work avoids risks caused by the capabilities of the team and that time is taken to support and develop the team to address significant weaknesses.

- *Understand the perspective and perceptions of the acquirers of the system*. Acquirers are a key stakeholder group who judge its success and usually have strategic and budgetary control, so can halt the project before delivery if they are unhappy. Specifically addressing this group's needs, perceptions and concerns emerged as an important factor for some of the experienced architects in our study. Acquirers are often senior managers and so may be distant from the day-to-day reality of a project and need regular, targeted, clear communication to understand their concerns and ensure that they have a realistic view of the project.

4.6.2 Prioritise Effort According to Risks

During a project, an effective approach to prioritising architectural attention is to use a risk-driven approach to identify the most important tasks. If the significant risks are understood and mitigated, then enough architecture work has probably been completed. If significant risks are open, then more architecture work is needed. The specific heuristics to consider for risk assessment are:

Risks from external dependencies. Understand your external dependencies because you have little control over them and they need architectural attention early in the project and whenever things change.

Risks from novel aspects of the domain, problem or solution. Another useful heuristic, from the experience of our study participants, is to focus on novelty in your project. What is unfamiliar? What problems have you not solved before? Which technology is unproven? The answers to these questions highlight risks and the participants in our study used them to direct their effort to the most important risks to address.

Risks in the organisational environment. Each organisation is different and there are nearly always risks specific to an environment such as the internal political situation, what is possible in the organisational culture, and the maturity of the organisation with respect to architecture, change and risk. Different organisations also have different cultures and capabilities for funding change, which can create risks. The speed which different sorts of risk and difficulties can be addressed can also be affected by organisational factors and so may cause you to change where you focus attention. Participants in our study noted the importance of "situational awareness" [178] that allows the architect to find and address the risks specific to their organisational environment. Risks from the external environment. Nearly all organisations exist in a complex ecosystem of interacting partners, customers regulators, competitors and other actors and they can be a source of risk for many systems, as can general trends and changes in the industry that the organisation exists within (such as changing regulatory environment or industry-wide pressures such as reducing margins on products or services).

Risks related to cost and time. Most architects will report that they are often expected to achieve challenging goals in unrealistic timescales or with unrealistic cost estimations. Many of our study participants reported that they needed to focus significant attention on risks resulting from cost and time.

Identify the high impact decisions. Prioritise architecture work that will help to mitigate risks where many people would be affected by a problem (e.g. problems with the development environment or problems that will prevent effective operation) or where the risk could endanger the programme (e.g. missing regulatory constraints).

4.6.3 Delegate as Much as Possible

Delegation was an unexpected theme that emerged from our study. The architects who mentioned this theme viewed themselves as a potential bottleneck in a project and delegation and empowerment of others was a way to minimise this. Delegation was also seen as a way of freeing the architect to focus on the aspects of the project that they had to focus on rather than all the other aspects that they could possibly get involved in.

The general message of this theme is to delegate as much architecture work as possible to the person or group best suited to perform it, to prevent individuals becoming project bottlenecks, allow architects to spend more time on risk identification and mitigation, and to spread architectural knowledge through the organisation. The heuristics that were identified to help achieve this are:

Empower the development teams. To allow delegation and work sharing, architects need to empower (and trust) the teams that they work with. This allows governance to become a shared responsibility and architecture to be viewed as an activity rather than something that is only performed by one person or a small group. This causes architectural knowledge, effort and accountability to be spread across the organisation, creates shared ownership, reduces the load on any one individual and prevents reliance on a single individual from delaying progress.

Create groups to take architectural responsibilities. A related heuristic is to formalise delegation somewhat and create groups of people to be accountable for specific aspects of architectural work. For example, in a large development programme, an architecture review board can be created to review and approve significant architectural decisions. Such a group can involve a wide range of expertise from across the programme and beyond, so freeing a lead architect from much of the effort involved in gathering and understanding the details of key decisions, while maintaining effective oversight to allow risks to be controlled and technical coherence maintained. Similarly, a specific group of individuals could be responsible for resilience and disaster recovery for a large programme, allowing them to specialise and focus on this complex area, and allowing a lead architect to confidently delegate to them, knowing that they will have the focus and expertise to address this aspect of the architecture.

4.6.4 Team Effectiveness

A theme that emerged when we validated our initial model with a wider group was the need to focus attention on making sure that the overall development team was as effective as possible. The participants who indicated the importance of this factor were concerned

with the need to develop the individuals in the team and to ensure that the team was as diverse as possible, to allow it to use a range of skills and perspectives when innovating and solving problems.

Other aspects of this theme that participants were concerned with were the importance of architecture work being used to quickly unblock the team when it hit difficulties and the importance of technical leaders, like the architect, to step in when needed to make sure that the team was functioning well and to address any dysfunctional behaviour observed.

The heuristics that the participants identified as being important for focusing architectural attention to achieve team effectiveness were:

Develop the team through mentoring. Every team should be on a collective journey towards improvement and hopefully, every individual in a team is on a similar personal journey to be the best that they can be. People doing architecture work tend to be some of the most experienced people in a team and so a valuable and important area to focus attention, in order to achieve a highly effective team, is to spend time developing the individuals in the team, and the team as a whole, through thoughtful, intentional mentoring.

Achieve team diversity for better innovation and problem-solving. In order to innovate and identify good solutions to problems, it is valuable to have a range of experience, perspectives and skills in the team. Our study participants indicated that a valuable use of architectural time is to spend time building diverse teams that can achieve this.

Remove blockers preventing team progress. Development and support teams often tend up blocked by technical or organisational factors, so spending time resolving these problems is a valuable focus for many architects.

Address dysfunction in teams. Sometimes teams do not work well and it requires someone who is close to the team and respected by them, but outside the team structure, to identify the problem and suggest solutions. Some architects work directly in individual teams and are not well placed to do this, but people doing architecture work

are often close to the teams but outside their structure, and have the respect, soft-skills and experience to resolve team problems. This use of architectural time can have huge benefits when dysfunctional behaviour is observed in teams.

4.7 Threats to Validity

We designed and conducted this study carefully to provide us with a useful model and a reliable evaluation of it, avoiding bias as much as possible. Specific steps we took to produce a reliable evaluation of the model included focusing on the practitioner community (as they are the intended users of the model), focusing on experienced respondents who have the experience to evaluate the model, finding a reasonably large, geographically distributed group to validate it for us, structuring the questionnaire to allow disagreement as well as confirmation, and analysing the results in a careful, structured manner to allow the data to lead us to the conclusions, to avoid the danger of us subconsciously using it to validate an opinion we already held. However, we acknowledge that there are potential limitations to any qualitative study, including ours, which can result in threats to our study's validity.

There are four main types of threat to the validity of a study like this, namely construct, internal, external and conclusion validity as defined in [110].

Construct validity is concerned with the relationship between theory and observation. A commonly recurring threat when using questionnaires is the phrasing of the text, the questions and the responses to the closed-ended questions. A second threat is where too many closed-ended questions are used and respondents cannot find suitable responses in the available set. We addressed potential problems with wording by keeping the amount of text in the questionnaire as small as possible and using simple language that directly referenced the model. The model itself was derived from the language and concepts that emerged from the semi-structured interviews. We also tested the questionnaire ourselves and on a small number of other architects that we knew, to ensure that their interpretation of the questionnaire was as we expected and we refined the language slightly as a result. We mitigated the possible problem of participants being unable to express their opinions through the closed-ended questions by limiting the closed-ended

questions to being simple ratings and then providing open-ended questions for the participants to explain, expand or clarify their answers.

Internal validity is concerned with the validity of the causality relationship between the observations and the outcomes of the study. We addressed this by using very straightforward analysis approaches, both for the statistical data analysis and for the analysis of open-ended answers, so the threats to the correctness of the analysis we performed are minor. We also reviewed all of the answers from each respondent to ensure that they formed a credible and consistent set (which all did), so validating that the respondents understood the process and were expressing a coherent opinion. To avoid possible misunderstanding we provided a clear definition of the model, links to additional information, trialled the questionnaire with people we knew, and included open-ended questions to allow respondents to express opinions that could not be easily captured using the closed-ended questions.

External validity is concerned with the generalisability of the results of the study. The key risk we identified relating to external validity is an unrepresentative respondent population or a respondent population who lack competence in software architecture and so cannot validate the model effectively. We mitigated these risks by finding a relatively large respondent population, who are distributed geographically, although as noted earlier, nearly all of the respondents came from Europe and the Americas. So a residual risk we continue to have is the lack of representation from Asia, in particular, countries like India, China and Singapore, with significant software engineering populations and the potential for significant cultural differences from Europe and the Americas. We mitigated the concerns around experience and competence by targeting experienced architects and architects in our extended professional network, whom we knew to be experienced and highly competent. We know a significant percentage of the respondents at least slightly and through some informal sampling of employing organisations and job titles, have a high degree of confidence in the ability of our study participants to validate or critique the model correctly. This leaves us with a residual risk that our extended network may be more likely to think similarly than a truly random sample, but anecdotally we do not believe that they are significantly different to most practitioners we have met over the years and we believe this trade-off to ensure credible study participants is the correct one.

Conclusion validity is concerned with the validity of the relationship between the data obtained in the study and the conclusions that have been drawn from it. The threats that we identified in this area are whether we asked the right questions at each stage of the study, whether we made mistakes in analysing the data, and whether we introduced unconscious bias into the study which could invalidate our conclusions. We mitigated the possibility of asking the wrong questions by using a semi-structured interview in the first stage and providing extensive opportunity for open-ended responses in the third stage. We acknowledge that we could have made mistakes in our analysis and processing of the data, but we mitigated this by reviewing and cross-checking our work and using a simple, repeatable process which was straightforward to follow. The largest risk to conclusion validity is probably the chance of introducing unconscious bias, as much of the study involved interpreting open-ended responses from the interviews and the questionnaire. We attempted to mitigate this risk through the use of the grounded theory process, which helped us to be led by the data rather than trying to fit the data into a pre-existing theory. We also reviewed our conclusions several times and repeated parts of the analysis if we felt that there was any danger of an alternative outcome being more representative. We also did not have any preconceived ideas about the likely outcome of this study at the beginning, so did not have an underlying theory we were trying to validate. Overall, we do not feel that we have been likely to introduce unconscious bias in the study, but we accept that it is hard to be certain that this did not occur at all.

In summary, we have designed and executed the study carefully, but do acknowledge that there are a number of threats to its validity which could threaten the generalisability of our results. Probably the most severe threat to the global applicability of the model is the lack of study participants from Asia. However, this threat does not suggest that the model will not be useful in Europe and North America, which would still be a valuable outcome.

4.8 Conclusions

Our experience and informal discussion with architects over many years suggested that they find it difficult to decide how to focus their effort to maximise their effectiveness and allow time to focus on architectural qualities, such as energy effiency, that were not

immediate priorities of key commercial stakeholders but were clearly important to the long-term success of the system they were working on. We were interested in how experienced practitioners solved this problem and whether they used common heuristics. To investigate this, we used a four-step process of investigation.

We started with a semi-structured interview process with eight experienced practitioners. The conclusion of the initial study was that there are some shared heuristics which practitioners use, but that the community of practising architects is not aware that the heuristics are common and shared. We found that the heuristics clustered into three groups: focus the architect's attention on stakeholders, use their time to address specific risks and delegate as much as possible, in order to give them as much time for architecture work as possible.

We then created a simple structured model to capture and explain the heuristics that emerged from the initial study and we published this via Internet social media channels. In the next step, we asked practitioners to complete a survey to comment on the usefulness of the model and whether anything had been missed. 84 responses were received to the survey, mainly from European and North American software, solution and enterprise architects with over 10 years of professional experience.

When we analysed the survey responses we found that the model validated well, as 70% of the practitioners who responded to the survey think it would probably or definitely be useful, but we found that we had missed several important risk factors which are commonly used for prioritisation and we had missed an entire element of prioritising effort, which is the need to spend time to ensure overall team effectiveness. We added these missing elements to the model.

These findings are not completely unexpected and many of the heuristics in the model are familiar. However, neither the participants or ourselves knew that these were the important and shared heuristics before we undertook the study, so we believe that the model we have created will have value as a teaching aid and as an aide memoir for experienced practitioners.

Given the validation that we have achieved, the model should also be an effective technique to help architecture practitioners understand how to organise their priorities in a way

that allows them to address quality properties, like energy efficiency and security, that key stakeholders often ignore when they prioritise the work for the development team.

In summary, this work contributes a useful, validated, model to help architecture practitioners to prioritise their effort effectively, but more specifically, it has the potential to guide practicing software architects to prioritise their workload such that they can address energy efficiency as a key system quality property.

Chapter 5

Design Principles for Energy–Efficient Applications

5.1 Introduction

Digital-transformation initiatives have led to major efficiencies and cost savings, including the transition from paper-based processing to electronic documents and the use of traffic-routing algorithms for vehicle navigation. However, the software performing this new computerised work consumes nearly 10 percent of the world's electricity [136]. Today's cloud-based applications span multiple continents, consuming energy in servers, networks, cooling and power facilities, storage, and user devices.

As discussed in Section 2.4, over the past decade researchers have been studying IT infrastructure energy consumption, working to increase data centre, network, and hardware efficiency. A number of significant research projects (such as DC4Cities [48] and the European Commission's work on data centre efficiency [2]) have brought promising research together with interested practitioners in order to find practical solutions to the problem of ever-increasing energy consumption for the world's ongoing digital transformation. As a result, data centre energy efficiency has improved considerably. For example, in the US,

public-sector data centres now often operate at a power usage effectiveness (PUE) of less than 1.5, whereas a PUE of 2 was considered normal only a few years ago. [1]

Individual hardware devices have experienced a similar trend and computations per joule of energy have doubled roughly every 18 months over the past two decades [91].

However, an aspect of energy efficiency that has been largely neglected is the efficiency of the software applications underpinning this digital revolution, and specifically how to provide the software architecture practitioners designing them with practical guidance on how to take energy properties into account as they design their systems. In this chapter, we start to address this situation and present three simple design principles that software architects can use to address system-level energy efficiency. We also present a case study that inspired the creation of the principles and illustrates the energy savings possible with a holistic architecture-led approach.

5.2 Research Method

The aim of this aspect of the research was to identify design guidance that could assist architects in treating energy efficiency as an architectural concern as part of their normal architecting work. Achieving this would allow us to answer research question RQ3.

The work began with a narrative literature review to survey the state of research in this area, described earlier in section Section 2.3, which identified some useful ideas in the research literature, notably an architectural perspective for energy efficiency [78] containing some useful guidance for the architecture practitioner. We found relatively little design guidance relevant to the software architect working on application software.

We wanted to identify some useful design guidance to guide architects when considering energy efficiency and identified an industrial case study which had successfully improved energy efficiency to learn from. We found a case study from eBay who had successfully reduced the energy consumption of key application services as part of their Digital Service Efficiency (DSE) initiative [47]. We analysed this scenario and synthesised

[1] PUE is a data centre's total energy consumption divided by its IT energy consumption, usually measured over one year. A PUE of 1.5 indicates that for every 1 KWh of IT load, a data centre requires an additional 0.5 KWh.

key principles from it and described them. These principles can now be integrated into practitioner-oriented literature such as the architectural perspective.

5.3 The Challenge for Software Architects

We suspected that software architects might find it difficult to prioritize energy efficiency for three main reasons.

First, we have little practical understanding of how design decisions affect energy efficiency or other system qualities such as user experience, reliability, and performance. Without this knowledge, analyzing tradeoffs to elucidate the benefits or costs of improving energy efficiency is difficult.

Second, to achieve meaningful improvements in energy efficiency, architects must gain new technical knowledge beyond traditional design boundaries. This will require that people from different specializations and departments work together. Such collaboration is challenging given the structure of many organizations that inadvertently prevent such collaboration through competing objectives, human dynamics and organisational barriers.

Finally, system acquirers and users rarely list energy efficiency as a major concern. This is partly because of split incentives. System operators such as administrators or data centre managers don't pay for the energy bill directly - the budget for energy tends to be included in a separate facilities budget owned by a different manager. This means that they would see little or no direct personal benefit from any energy savings that they achieve.

A previous survey attempted to understand architects' perspectives on energy efficiency [20] and surveyed 12 representative, experienced architects from various organization types. They were asked whether they had encountered energy efficiency as an architectural concern in the previous five years and whether they believed that they had the right tools to address energy-related challenges. The survey also asked whether the participants believed that energy would be an important architectural concern over the next five years.

The survey, while small, did include participants from a number of relevant sectors including 7 from technology consultancies, 1 from an Internet company, 2 from banking and finance and 2 from other industries. Of these respondents, most of them (83%) hadn't had to deal explicitly with energy concerns during the previous 5 years although interestingly 66% of them thought that energy was an important concern that they would need to deal with over the next 5 years. Given the state of the art, predictably, only 25% of the respondents thought that they had the right tools to deal with energy as an architectural concern. These findings are consistent with our industrial experience, where energy is rarely discussed as an architectural concern, when it is discussed it is usually seen as a hardware and data centre concern, and when application architects are concerned with it, they lack the methods and tools to allow them to understand and compare the energy implications of their architectural decisions.

5.4 State of the Art

To increase efficiency, we must be able to measure it. That is, we must be able to measure the useful work our software applications produce and the amount of energy this takes and then optimize the ratio between the two. However, although the data centre world has metrics such as PUE, no comparable, generally agreed, metrics exist for software.

Optimization must also consider key quality properties such as resilience (because redundancy in system designs is usually a major contributor to energy consumption), usability, and performance. However, architecture practitioners don't generally have access to such tools and techniques today [20].

Despite these challenges, energy efficiency has been gaining traction in software engineering. Much of the early research focused on measuring applications' energy consumption [75] and tried to define useful work so as to allow the creation of useful metrics (for example, the DC4Cities project we mentioned earlier [48]). In parallel, other researchers have explored compiler optimization to decrease energy consumption or have evaluated design patterns' energy efficiency.

All these efforts have helped us begin to understand and optimize software applications. However, improving today's Internet-scale systems will require a more holistic approach

that considers the whole system. Software architects are well placed to lead such an approach.

5.5 The Three Principles

On the basis of early industrial experience in successful projects to reduce energy consumption we propose three simple architecture principles for achieving energy-efficient systems:

- *Principle 1*. Energy efficiency metrics must relate business transactions to energy consumption in a meaningful way to key system stakeholders.
- *Principle 2*. Identifying sources of energy waste at the system level produces the biggest savings.
- *Principle 3*. Addressing the energy optimization problem requires a cross-disciplinary team.

These principles are discussed in more detail in the following subsections.

5.5.1 Relating Business Transactions to Energy Consumption

Architectural priorities are set by the system's stakeholders, whom the architects interact with to understand their needs and goals, trading these off to reach an acceptable set of architectural properties for the combined stakeholder group [153]. In order for energy to be viewed as an important system quality, it must be explained, measured and its impact assessed in a way that significant system stakeholders can understand and relate to their own goals. Ultimately, identifying suitable metrics and relating them to stakeholder goals can help to achieve a holistic approach to system quality property tradeoffs and hence drive revenue and cost optimization.

Like many other quality properties, we have observed that energy properties tend to be viewed as a purely technical concern, only of interest to stakeholders involved in system

operation. This means that today energy properties rarely become a concern of senior business stakeholders and so often do not have sufficient executive sponsorship to gain significant attention, when in competition with core concerns like functional coverage, performance or scalability. Until recently we have also seen this effect in regards to security, which only recently has started to be a concern of key decision makers [32].

The key to achieving alignment with the goals of senior decision makers is to relate energy characteristics of the system to the operational cost of the system and hence to characterise energy usage as part of the cost of executing business transactions. The cost of a business transaction is quite simply the reduction in margin (profit) on that transaction and for many businesses, operating on slim single-digit percentage margins, small improvements in per-transaction costs are attractive targets for engineering effort, as they return more and more benefit as the volume of business increases.

Of course, executive decision makers are only one type of stakeholder and an important part of the architect's job is to make the right tradeoff between the needs of different types of stakeholder. The executive decision makers are normally the *acquirers* of systems, being the budget holder and so operational cost and development cost are significant concerns for them. Most systems have one or more type of *assessor* in the compliance, audit, security or quality management groups. The assessors are often interested in energy consumption from an environmental management compliance perspective. While not directly a financial concern, compliance or audit assessors may well have environmental impact, or direct energy efficiency goals that they are interested in achieving. The *systems administration* stakeholders encompass all those involved in operating system system (from data centre managers to individual administrators) and this group are often goaled with reducing their overall site (data centre) energy consumption and so will be motivated to work with application architects who are energy aware and can help to achieve application level reductions to contribute towards this goal. The *developers* of the software often aren't aware of their contribution towards a system's energy characteristics and can be difficult to motivate to prioritise energy concerns unless there is a direct impact on user experience (such as IoT device software or mobile application developers who often have to be very aware of energy consumption to avoid exhausting battery based power sources). This means that we are quite dependent on the *architects* (who

are themselves stakeholders in the application) to understand the wider context for application energy concerns and translate these into comprehensible goals for developers to incorporate into their work.

While different stakeholders have different perspectives on the energy properties of an application, in reality, energy can affect an organisational in a number of ways including ethical (how to minimise the resources required to run the organisation), environmental (how to reduce or "green" energy consumption to reduce the organisation's environmental impact), reputational (to avoid negative associations with careless use of natural resources or creation of pollution), cost (where reducing energy consumption will make a business more efficient) and agility (where energy efficiency can reduce the need to increase an organisation's physical data centre footprint in order to change or expand the organisation's business). As more and more enterprises move to public cloud environments [57] these organisations need to focus on the direct ethical and reputational impacts of application energy consumption, and work with their cloud providers to address environmental and cost impact of their energy consumption (the cloud providers largely hide the possible impact on agility due to their huge scale).

In summary, if the energy characteristics of an application are considered simply as an organisational level cost concern (as, for example, environmental emissions were a few years ago) then it is unlikely that they will ever get the attention required for architects to prioritise them as an architectural concern. The role of application architects is to make the broad organisational impact clear and this involves transparency of usage, impact and cost for the different parts of the organisation that can have an effect on energy consumption. Hence the architect needs to translate the technical energy metrics for different stakeholder groups to relate them to their goals in order to illustrate their impact on these goals and motivate different stakeholder groups to address energy concerns in a cross-organisational way.

5.5.2 Energy as a System Level Concern

The second principle we have identified is that energy needs to be seen as a system level concern, and so be addressed at the application architecture level. However, our experience and review of the research literature suggests that many of the most practical

existing approaches to dealing with energy efficiency and usage are at a micro level, measuring the energy usage of individual code procedures, or at the large-scale macro level, analysing the energy usage of an entire data centre.

While valuable in different situations, the problem is that neither the micro or macro view is useful when trying to understand and improve the energy efficiency of mainstream software applications. If we measure energy efficiency at the level of individual code blocks, this is too fine-grained a level of detail for an application architect to use at any scale. Improving measurements taken at this level can rarely have a big impact on system energy consumption as dozens or hundreds of such micro-level components combine to process a particular system usage scenario and it is very difficult to see which of the micro-level improvements have the biggest impact at system level. On the other hand, the data-centre level measurements, while useful for understanding the energy characteristics of the entire IT estate, also don't help the application architect, as the only metrics available are maximums, minimums and averages on sections of the infrastructure estate and these measurements do not allow an application architect to understand the energy implications of their architectural decisions.

It seems clear then that the most effective level of abstraction at which to consider application energy efficiency is at the architectural level of the application itself. This allows us to consider the application as a complete system and to analyse usage patterns (scenarios) rather than individual components. Scenarios describe how systems are used and so understanding their energy characteristics reveals the real runtime energy characteristics of an application. Once a system's key usage scenarios are understood, then their energy characteristics can be analysed and the architect can use this information to allow the architect to see where to focus their effort for the most impact and the effect of their architectural decisions.

An example of how considering energy at an application level can help to focus effort where it will be the most effective is to consider how redundancy is used within an application. Many applications use redundancy of system elements in order to achieve qualities like scalability and resilience. However, redundancy is a commonly overlooked source of energy consumption as it is often applied at all levels, including facilities, hardware, and software, due to the application not being considered as a holistic system. A

system-level evaluation of resilience requirements allows the architect to identify where redundancy is unnecessary, which is a huge opportunity to achieve energy savings that would be difficult with local optimizations.

5.5.3 Employing Cross-Disciplinary Teams

Energy optimization requires design work across traditional design boundaries. For example, optimizing the design of resilience requires collaboration among infrastructure engineering, application development, and business teams. Without a collaborative approach, improvements will be restricted to local optimizations, which often miss the bigger opportunities for savings.

This problem is related to the problem of most energy efficiency work being performed at the micro (code block) or macro (data centre) level, as we described above. These may be considered to be vertically separated areas of focus. However, for this principle, we are concerned with the tendency for organisations to partition work between "horizontal" organisational groups, from the business organisation, through the application delivery teams, through the operational teams, to the data centre infrastructure organisation at the end. This organisational separation of people and work encourages a tendency to believe that energy consumption is "someone else's problem". For example, production operations groups view it as a problem with inefficient applications, whereas application development groups view it as a problem both with the lack of focus on energy by their business stakeholders and the "arbitrary overheads" they believe are imposed by the production environment that their application runs within.

This organisational fragmentation can have many negative effects, but it is particularly serious when we are trying to address energy consumption concerns, as it results in the information needed to address energy concerns being scattered across organisational boundaries. It is common to find hardware energy consumption metrics in one place, operating system counters in another, application tracing and monitoring somewhere else again, and data centre efficiency and overheads being yet another group.

The knowledge and skills needed to understand and address energy concerns also tend to be found in different teams, separated by organisational boundaries, as does the

authority needed to make meaningful change and effectively evaluate the impact of a change, as this will be at multiple levels of abstraction and ranges of authority.

All of these factors mean that for us to address energy concerns effectively, we need organisations to form cross-functional teams to take shared accountability for driving down energy usage by finding the most effective places to focus their attention, identify solutions to those problems, and then to gain the consensus and commitment to addressing them.

5.6 Case Study: Online Auction Site

The energy management principles we identified above were inspired by the work performed by architects at eBay, the Internet auction site, who identified a number of significant architectural design decisions to improve energy efficiency and from which we identified these principles.

In order to reduce its environmental impact, while also reducing cost, eBay introduced a programme known as the Digital Service Efficiency (DSE) initiative [47] which aimed to increase the efficiency of the eBay platform. DSE relates business metrics such as volumes and value of customer transactions to their energy consumption and environmental impact. The approach they defined designed a set of metrics that were meaningful to key stakeholders across the business, which helped the organisation to understand, analyse and reduce its energy consumption, while understanding the architectural tradeoffs that this involved (which is the basis of Principle 1). Some of the metrics that the DSE programme defined were the number of buy transactions per kWh of energy consumed, the number of sell transactions per kWh of energy consumed, the revenue generated per MW of power consumed, and the volume of CO2 emissions per million active eBay users.

The organisation identified that reducing infrastructure redundancy was as one of its key opportunities to save energy at an acceptable level of cost and effort. In order to achieve this, eBay considered energy consumption at a system architecture level and made significant architectural changes that would result in large improvements in energy efficiency. They achieved this by reconsidering the assumptions and tradeoffs between business requirements for resilience and the cost of introducing redundancy in the architecture (this

was the basis of Principle 2). The key architectural insight that allowed them to make this step was the realisation that no matter how resilient its data-centre application services were, if the client application (a mobile app or web browser application) failed then the whole session failed and the client applications failed a lot more frequently than the data-centre services. Through their analysis they came to realise that in many cases, making the data-centre application services more resilient would not increase resilience as experienced by their end users and so a high degree of redundancy in the data-centre services was unnecessary in many cases. This meant that responsibility for system resilience could be moved to the weakest link: the client application (i.e. the eBay web user interface and the eBay mobile applications).

To capitalise on this opportunity, they defined a new architectural pattern for client access to application services and introduced a service request proxy component in the client applications. The client applications call the proxy, rather than the services, and the role of the proxy is to implement a timeout feature so that when a service request has exceeded a reasonable period of time, the proxy will automatically reissue the request, which will be routed to another service instance. Hence if a service fails during a client's service request then the client is unaware of this provided that the proxy's duplicate request can be routed to another surviving service instance. While not necessarily suitable for transactional services (like payments) due to complications with idempotency, this approach works very well for non-transactional or less-critical services like search requests or user profile updates. This allows the non-critical or non-transactional services to be significantly simplified and data-centre side redundancy and retry to be eliminated because the proxy's operation can mask their failures.

Using their existing business analytics, eBay estimated that only about 10 percent of service requests needed to be considered as critical transactions, such as payments (which has specific regulatory needs and is normally expected to be executed on a highly available infrastructure). This insight allowed eBay to process critical transactions at a separate, smaller, highly resilient data centre that has a fraction of the capacity of the main data centres [123] while processing most of the service invocation workload using a cheaper, less resilient infrastructure. This provided a significant improvement in energy efficiency and required collaborative work across eBay's engineering, operational, and business teams (which led to Principle 3).

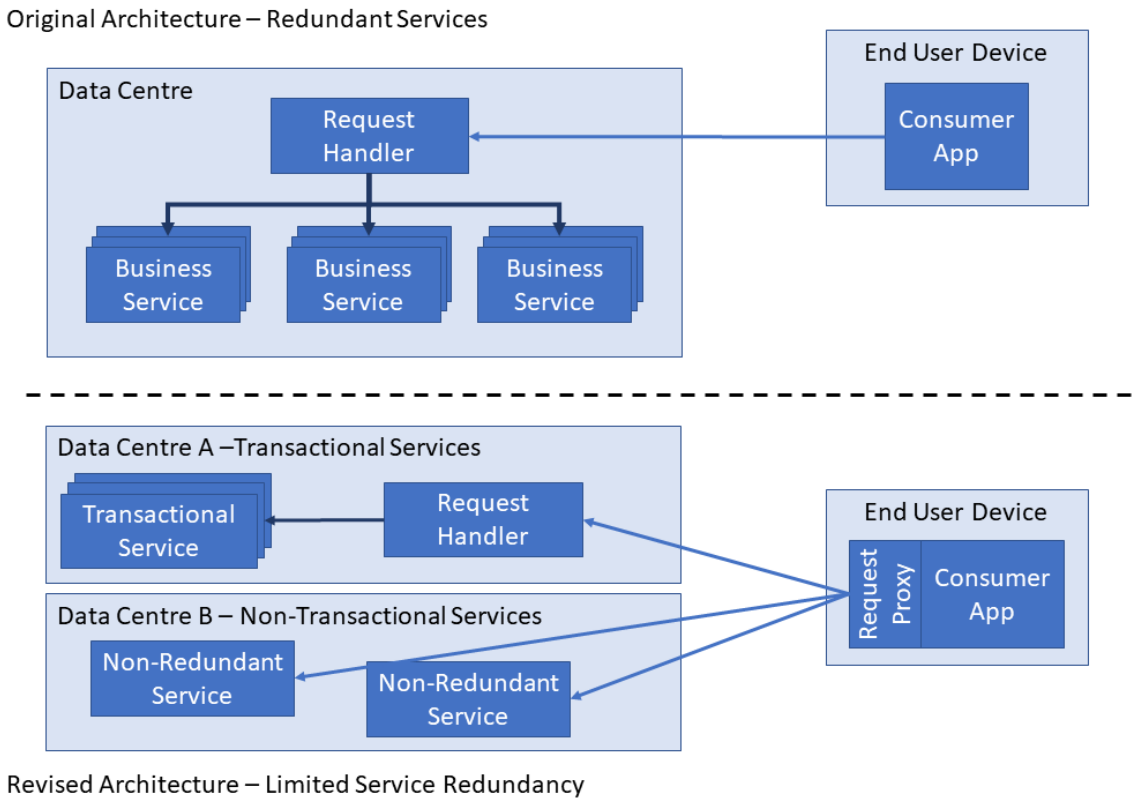


FIGURE 5.1: eBay's Architectural Evolution for Energy Efficiency

Figure 5.1 depicts eBay's original and new more energy-efficient architectures. The new architecture allows for reduced data centre redundancy while maintaining overall system performance and resilience.

This new architecture allowed eBay to achieve major capital and operational expenditure savings and well as reducing their energy consumption significantly. This was due to the simpler design of the low-redundancy services, which have substantially decreased the amount and complexity of the infrastructure needed. As a consequence, they have achieved a significant reduction in data centre build-out and fit-out costs and timescales. The underlying factor that allowed this improvement was that redundancy costs are significant. For example, according to Steven Shapiro, the cost of building a Tier III data centre is double that of a Tier II data centre [157]. (The Tier Classification System is a widely used rating system for data centre availability, with Tier I being the least available or redundant facility and Tier IV the highest [72].)

Even more important (from our perspective), eBay has reduced energy consumption by approximately 50 percent because the low-redundancy site requires fewer infrastructure

components (for example, N + 1 rather than 2N + 1 redundancy). This has resulted in significant energy cost savings and also a reduction in maintenance and hardware refresh requirements, further lowering environmental costs.

5.7 Conclusions

There has been increased interest in reducing the significant energy costs of running large IT systems. However, little attention has been given to addressing energy efficiency at an application level, which prevents software architects from considering energy as a first-class architectural concern. When software architects do attempt to understand the energy property implications of their decisions, they find that they lack suitable tools and methods to address energy concerns when designing systems. With this challenge in mind, we've investigated how a large organisation solved this challenge and from that experience suggested three practical principles to guide architectural decision making, which architects can use to guide energy-related tradeoffs during system design even with today's limited knowledge and technology.

Despite these principles' simplicity, the eBay experience shows that they can yield significant cost and energy savings when applied to large-scale production systems. Savings of this scale are difficult to achieve through local optimizations, so we must address the problem at a system level, and ensure we allow software architects to work across stakeholder groups and organisational boundaries in order to achieve meaningful results.

Chapter 6

Monitoring Application Energy Usage During Operation

6.1 Introduction

As we discussed in Chapter 1, the energy usage of information and communications technology systems is starting to receive much more attention due to its sharp increase in recent years and the predicted continued growth for the foreseeable future. Researchers from a number of domains have been working on the problem of ICT energy efficiency for some time and have made good progress in a number of areas including data-centre level efficiency [40] and micro-level code analysis, allowing more energy efficient implementation options to be identified [131].

However, as we also discussed, the problem for the software architect is that their interest sits between these two extremes as they need to understand the energy consumption at the overall application system level rather than at the individual software module level or at the data centre level where many applications are consolidated. As we saw during our literature review of this area Section 2.4, some progress has been made in providing tools and techniques for application energy monitoring (such as [131] or [138]), but much of the technology developed is immature, relatively unproven, and unavailable for general use.

The other limitation, from the software architect's perspective, that we identified in the existing research is that it monitors application energy consumption in terms of the energy properties of architectural components, operating system processes or even server machines. An architect cannot gain immediate insight from these measurements as they are not related to the workload that was running on the system at the time. Therefore the architect has to run careful benchmark exercises in highly controlled conditions to allow them to understand the energy properties of the different types of workload that their system processes. This information is needed in order to allow them to decide which usage scenarios to focus their attention on and how their architectural decisions affect the energy consumption of each one.

The modern trend towards microservice-based systems [179] makes things more complicated. In principle, it is possible to use or adapt the code-level energy estimation approaches to be useful with monolithic applications. But this quickly becomes overwhelming with a microservice-based system, due to the number of system components and the complex ways in which they combine to process an incoming request.

In this chapter, we present the logical (i.e. technology independent) aspects of the design for a piece of technology that we have designed to address this problem by fairly allocating the energy usage of a host machine to the application elements running on it. Using modern microservice and operating system technology including containers, tracing and resource monitoring, combined with energy statistics for a machine, we can provide the software architect, and also the platform operator, with reliable estimates of the fair energy allocation of a machine's total consumption required to process requests for an application running on it. This allows cost estimation but, more importantly from the software architect's perspective, the evaluation and exploration of architectural alternatives to minimise energy consumption.

The logical design in this chapter then forms the specification for our implementation of the ideas which we describe in Chapter 7.

6.2 Motivation

There are several reasons to seek a method of estimating energy usage by software applications, but two immediate motivating examples in our case are cost estimation and architectural evaluation.

Today, the energy consumption of an application is not taken into account when considering its cost to operate and so there is little motivation for the software architect to understand and minimize their application's energy footprint. This prevents large possible reductions in energy usage and environmental impact and cost.

If the architect is interested in the energy usage of their application there are limited options for estimating the energy usage of software at the application level. This means that it is difficult for architects to evaluate the energy usage qualities of different architectural options that they are considering.

The specific advance achieved by this piece of work is the novel combination of scenario (application request) level resource usage statistics, total host resource utilisation statistics, and host energy consumption to create an approach to fairly allocate the energy consumption of the host to the workload that ran on it at particular points in time. This allows a fair, reliable and useful estimate of the energy usage that should be allocated to specific requests made to the application.

The goal of this is to provide a practical approach for software architects to estimate the energy impact of their applications and to evaluate different architectural design and deployment options in terms of their likely energy usage.

6.3 Research Method

The research reported in Chapters 6, 7 and 8 was undertaken in order to answer research question RQ4 (*How can we make architects aware of the runtime energy characteristics of their systems?*) by designing and building a proof-of-concept implementation of a technical solution to provide architects with visibility of the energy consumption of their

systems. This was an *Improvement Problem to Develop a Useful Artefact* in TAR terminology [177], which involved problem investigation, the design of the solution (a piece of software) and the validation of it using realistic testing. We excluded the further TAR steps of *real-world implementation* and *validation* from the scope of the work reported in this thesis due to the scale of the work required.

The work began with a narrative literature review to survey the state of research in this area, reported in Section 2.4. This exercise discovered that some application energy measurement systems have been designed and reported in the literature, but most are not available for general use. The other problem with the measurement systems found was that they do not allow the architect to understand the energy consumption of application execution scenarios, just of application components over a period of time. From the architect's perspective, this limits the value of the information the tools produce and means that the tools are difficult to use in production environments with a synthetic workload.

In response to these limitations, an approach to capturing representative energy usage for application execution scenarios was designed in a largely technology independent manner, as reported in Chapter 6. Our solution to the problem approaches it in a slightly different way to the existing systems and allocates estimated server energy usage to the applications running on that server, rather than trying to calculate an absolute energy consumption for each. This sidesteps a number of problems with the existing approaches, as we will illustrate later. We also performed the measurements and calculation from the perspective of the architect, using execution scenarios (tracing requests through the distributed application components), rather than measuring the energy usage of operating system processes. This design was then implemented using a specific set of technologies, including Linux, Docker and Java, as described in Chapter 7. Once implemented, a set of practical tests was designed and executed to validate the implementation's internal consistency and external correctness with respect to its runtime environment, as described in Chapter 8.

This work enabled us to answer research question RQ4.

6.4 Microservice-Based Systems

The microservice architectural style [151] is rapidly becoming a mainstream approach for building industrial software systems and it is systems built using this style that we are specifically targeting in this work.

A microservice-based system is made up of many small, encapsulated, network-connected services, rather than the more traditional approach of having a small number of large servers that aggregated many services (a so-called "monolith" [111]).

For our purposes, the important characteristics of a microservice-based system are:

- The business logic in the system is implemented as a group of small, focused services, each performing one task, typically implementing a "bounded context" in Domain-Driven Design terms [50].
- The services are as independent as possible and have well-defined service interfaces and only interact through these interfaces. Resources such as databases are owned by a specific service and are not accessed by other services (hence a microservice-based system will have many independent data stores rather than a single consolidated database used by many services).
- Handling an incoming request for a system is likely to involve a set of cooperating services, with one handling the initial request and then calling other services in order to fulfil the request and provide a response. Microservice-based systems often separate request-handling and domain services but we do not assume that this is necessarily the case.

Well designed microservice-based systems typically share other important characteristics [125], such as independent build, test and deploy for each service, and interfaces defined using machine-readable formats such as OpenAPI [168] and RAML [163] but these other characteristics are not significant in the context of this work, and so we do not assume that they are present.

6.5 Estimating Energy Usage

As we investigated the problem of how to provide software architects, and other interested parties like platform operators, with estimates of application-level energy usage we identified a number of possible approaches, as described in Section 2.4.

- Other researchers have tried to create entirely *model-based approaches* (such as [156]) which can allow relative energy usage between different architectural structures to be estimated, but sidestep the problem of calculating energy values, do not provide runtime measurement and require significant amounts of effort to create the models for non-trivial applications.
- Other research projects have attempted to estimate architectural characteristics through *event-based simulation* [65], but creating event based simulation models is an unfamiliar process to many practitioners, and again does not provide runtime monitoring and is significant amounts of effort for non-trivial systems. It is also the case that existing research in this area has not yet investigated how to estimate energy consumption, but rather has focused on other architectural qualities such as performance.
- Some researchers have used *regression models* to provide an estimation of runtime energy consumption based on a number of resource consumption measurements for an operating system process and power measurement statistics [138]. As noted in Section 2.4 these approaches have had little validation beyond small laboratory experiments and we have concerns about the amount of training and re-training that regression models need to retain their predictive power in real environments.
- Other researchers working on runtime energy estimation have used *cost-based models* to provide an estimation of energy consumption based on the specifications of the hardware in use and key resource consumption measurements for an operating system process (typically CPU utilisation and in some cases network or memory utilisation) [1, 131]. Most of these research projects have been focused on micro-level measurement rather than architectural measurement, but the practical and useful approaches resulting from a number of these projects, along with their open source implementations, influenced our approach significantly.

However, as we mentioned earlier, all of these existing approaches suffered from the same limitation from our perspective, namely that they measure or estimate energy at the architectural component or operating system process level. Of course, this can be useful, but it means that to get meaningful results the architect has to run tightly controlled benchmark scenarios and perform the measurement during the time period bounded by the start and end of the scenario. Without this discipline, the energy usage of the components is not particularly useful, as the architect has no context as to the workload that caused the energy consumption.

That said, some of the research projects investigating code level measurement at the *micro-measurement level* have created sophisticated approaches and tools to estimate energy consumption for individual algorithms or programs using cost-based models with a high degree of success [128]. However, we discovered that these approaches rely on a highly controlled execution environment for the code being measured and in some cases access to CPU-specific low-level hardware state metrics, such as processor frequency statistics, as a result of the fine-grained level at which they attempt to measure power consumption.

We explored whether we could combine the results of several of these projects who open sourced their tools [24, 128] with application-level resource usage measurements to produce meaningful energy estimates for the application-level workload. However, we found that the assumptions caused by their fine-grained approach to measurement (such as access to CPU-specific low-level hardware state metrics) meant that they aren't practical to use in large distributed execution environments such as those used by microservice-based systems.

Having considered these options and realised that each of them had practical limitations, we shifted our focus slightly and reconsidered the goal of the work. The problem we aimed to solve was to provide software architects and other interested parties with information on how to improve the energy efficiency of their applications. The solution we identified to this problem was to shift our goal from precisely estimating the *actual* energy usage of a distributed application to estimating the *fair allocation* of energy consumption to each of the service components of an application. This is a useful goal because it allows service providers (such as platform operators) to understand how to fairly allocate the cost of

energy consumed by their infrastructure and it allows software architects to understand the energy consumption implications of their design decisions as a proportion of real energy cost, taking into account their deployment decisions as well as software design decisions.

Our approach assumes that the energy consumption of the execution platform hosting the application is available to the energy calculation process and uses this, along with the total resource usage consumption of the execution platform and the resource usage of the application components, to allocate the platform energy consumption to the different application elements running on it. By tracing the execution of inbound requests across application elements, this allows us to allocate energy usage to specific workload scenarios and so compare the energy efficiency of different parts of a system, different workloads and different system design options.

Allocating energy usage at the application level is a complex process and so it is important to be clear how the calculation will be performed before trying to design an implementation of it.

There are five quite distinct parts to the problem:

- *Identification of service elements involved in processing a request*. Processing a request in a modern distributed system will often involve a chain of service calls between the services that comprise the system logic. We assume that the implementation of the services is under the control of those wishing to estimate energy consumption.
- *Identification of the processing periods attributable to a request*. Once we know the system elements involved in processing a particular request, we need to identify when the element was performing work on behalf of the request.
- *Resource usage of the request*. Given the system elements involved in processing a request and the periods when they were active on behalf of the request, we need to estimate the runtime resources consumed by the system elements during these periods. The resource consumption we need to estimate is primarily the amount of CPU consumed (for reasons we will explain later) although the amount of memory in use, the number of network i/o bytes sent and received, and the number of disk

i/o bytes read and written are also typically available from modern operating system and container platforms.

- *Estimation of resource and energy usage of the underlying host machine*. Our aim is to fairly allocate the energy used by a machine during a time window across the requests that were active during that period. Hence we need to estimate the overall resource usage of the machine and the energy that it consumed as a result.

- *Energy allocation of the request*. Once we have the resource consumption metrics for a particular request, we then need to translate these into an estimate of the share of energy consumption that they imply. We do this by establishing the percentage of overall machine resource utilisation that can be attributed to the request and then allocating the same percentage of energy usage of the underlying host to the request.

Each of these aspects of the problem can be solved largely independently, and once resolved, combined to achieve a reliable and fair energy allocation for each inbound request for a microservice-based system. In the following sections of this chapter, we discuss the "logical" design for a solution to each of these aspects of a problem. This provides us with a largely technology independent design of the solution, which we then use as a specification for the technology-specific design that we implemented, as described in Chapter 7.

6.6 Logical Design of an Energy Allocation System

The functional design of a system to estimate energy allocation for application requests (which we have dubbed "Apollo" - the Greek God of Prophesy, amongst other things) is shown in the informal block diagram in Figure 6.1.

The elements in the diagram with solid fill are the underlying runtime platform and services, the diagonally hatched elements are the application under study, the densely-dotted elements are data, while the sparsely dotted element is the Apollo energy estimation system.

The elements of the system are briefly described below:

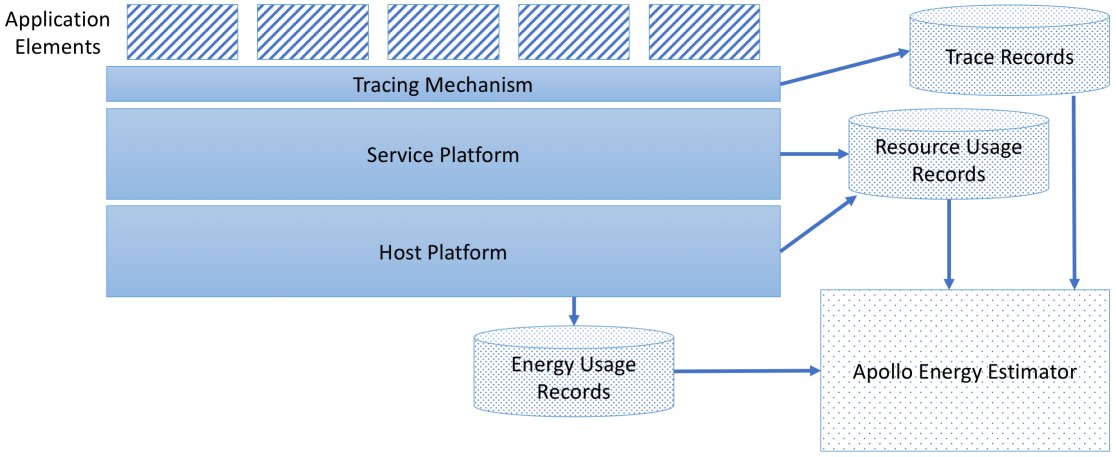


FIGURE 6.1: Logical Design of the Energy Estimation System 'Apollo'

- *Application Elements* are the architectural elements of the application which is being studied for their energy consumption characteristics. These are the main functional processing elements of the system, running in their own address space, providing a network interface to their services, invoked as part of request processing, with the implementation being under the control of the system owner who wants to estimate energy usage. An example would be a request handling microservice to create an order.

- *Service Platform* is the system software which hosts the application components and can provide detailed measurements of their resource usage. An example might be a PaaS platform like Cloud Foundry [33], a virtual machine, a modern operating system like Linux or a container platform like Docker [45] or Rkt [150].

- *Host Platform* is the hardware and operating system platform that provides the general computing platform that hosts the service platform and provides runtime statistics on the host's execution and energy consumption. This is typically an Intel-based server machine running a Linux or Windows operating system, or a virtualised version of them.

- *Tracing Mechanism*, which is key to our approach. We require the ability to reliably and efficiently trace the architectural elements that were involved in handling a particular request for the system. The tracing mechanism provides that ability, reporting the sequence of invocations of architectural elements and the time and duration of each. Technology to allow this transparently for microservices originated at Google

[160] and has been implemented in open source projects such as Zipkin [132] and Jaeger [77].

- *Resource Usage Records* is a database of resource usage statistics for all of the functional elements of the system, fed from the Service Platform and the Host Platform. An example could be a regular database or a more specialised time-series database like Prometheus [148] or InfluxDB [70]. The database is typically populated using a specialised statistics collection server like Telegraf [71] or cAdvisor [63].

- *Trace Records* is a database containing the request invocation traces from the Tracing Mechanism, which is likely to be a simple relational or document-oriented database, which the Tracing Mechanism writes its trace records to.

- *Apollo Energy Estimator* provides the energy allocation element of the energy estimation process, a calculator that works through the Trace Records and for each one, calculates which elements were active over which time periods and uses the resource usage records to estimate the resource consumption of each request across the elements that were involved in processing the request. These values are then compared with the overall host platform resource usage and the ratio of the values used to allocate the energy consumption of the host platform during the time period that the requests were active.

There are three key data structures in the design, which are critical to achieving the energy calculation, namely Trace Records, Resource Usage Records and Energy Usage Records.

- *Trace Records* are created by the tracing mechanism and are used to record the invocation of system elements in processing a request. There are two types of trace records, commonly referred to as Traces and Spans [34]. Common features of the two are the identity of the runtime element writing the record, the start time of the record and the end time of the record. A "trace" record is written by the first element to handle an inbound request and its start time is when the request is received and its end time is when the response was dispatched. A "span" record is written to record the invocation of another system element (i.e. service) by the

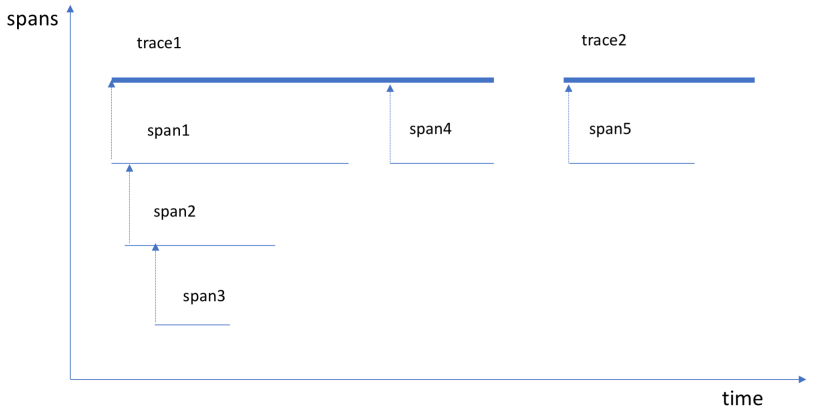


FIGURE 6.2: Example Execution Trace

original element handling the request and it has a "parent" attribute which indicates the trace it is part of. Should an element *e1* handle an inbound request and as part of processing it cause another element *e2* to be invoked, then a trace record is written to record the execution of *e1* and a span record is written, recording the execution of *e2*, with the trace record as its parent. Hence traces and spans are organized into a tree that mirrors the invocation structure of the request handling. An example trace is shown in Figure 6.2.

The figure shows two traces, representing requests, one after the other. The traces are distinguished by not having parent spans, both traces have child spans, as indicated by the dotted arrows. The y axis indicates invocation from top to bottom, the x axis indicates the passage of time. The first request has two child spans, indicating that it invokes two other services. Child span "span1" in turn invokes a third service, which invokes a fourth. The second request invokes a single service.

- *Resource Usage Records* are generated by the runtime platform and the host platform and stored in a timeseries style database to allow the metrics for a particular runtime element during a specified period to be retrieved (e.g. metrics for process *p1* for the 3 second period from 20171105T084512.500 to 20171105T084515.500 or metrics for the host *h1* for the 12.5 second period from 20171105T084512.500 to 20171105T084525.000). The metrics will be generated by the underlying platforms at regular but arbitrary intervals and so this data store will need to interpolate between the available measurement points to get the resource usage for the requested time periods. The metrics that both the runtime platform and the host

platform will be assumed to create are CPU usage (in a hardware-based measure such as "ticks" or a time-based measure such as milliseconds), memory usage (in KB), disk IO (in KB) and network IO (in KB). Absolute or cumulative measurements are both usable for our purposes, but we need to process absolute measurements into cumulative values to make our queries on the dataset tractable. At this stage, we assume that we can obtain a reliable measure of the resource utilisation of each architectural element of interest, without being concerned with how this is provided. Mechanisms which could provide this data include a container system like Docker, an operating system's process statistics, an internal application monitoring system or even a virtual machine manager with each architectural element in a separate virtual machine.

- *Energy Usage Records* provide a history of the estimated energy usage of the host platform (that is the virtual or physical hardware and its operating system). There are two varieties of energy usage record source that could be available to us depending on the situation. In some cases, there will be physical energy usage records available from data centre infrastructure sources such as DCIM data collection platforms, like Sunbird [165] or Nylte [126], which extract data from enterprise-class hardware devices through a protocol like IPMI [73] (and increasingly its more secure and capable replacement, Redfish [44]). These records can be fed into a timeseries database (similar to the one used for resource usage records) and queried directly. In other cases, a model-based approach can be used to simulate real power readings. A model-based approach can be used to produce workable estimates of energy usage at different points in time by using accurate power ratings for different hardware devices such as those reported through the use of the SPEC Power benchmark [96]. These benchmark results provide accurate power consumption rates for specific pieces of hardware at different degrees of utilisation. If we know what host platform is in use and have accurate host platform resource usage records (for CPU usage specifically) then these can be combined with the benchmark results to create a usable model-based estimate of energy usage of the underlying host at a point in time.

6.7 Utilising Resource Usage Records

Modern computing platforms like Linux and Windows provide detailed and comprehensive resource utilisation statistics that describe CPU utilisation (typically in terms of milliseconds or nanoseconds of CPU time used), disk i/o (in terms of read and write requests performed and blocks and bytes read and written), memory utilisation (as KB of memory used over time) and network i/o (in terms of receive and transmit requests and bytes) [62, 121].

When estimating energy characteristics of a workload, it is possible to measure the power consumption of a server via DCIM monitoring equipment or alternatively estimate power consumption using a model-based on reliable benchmarking. However, it is difficult to attribute the overall energy consumption of a server to the specific components within in. While it is possible to obtain representative power specifications for individual components of enterprise-class computing hardware [68], interpreting such specifications to produce accurate power consumption estimates is extremely difficult as they require a detailed understanding of the physical operation of a specific hardware element during the period of interest (such as the frequency of the CPU or the amount of time that disks spend spinning at full speed during a particular measurement period).

As a result, we concluded that combining CPU, disk, network and memory utilisation statistics in a cost-based model was not a practical approach for anything other than the smallest laboratory examples and so was not suitable for the large scale enterprise use that we aspired to apply our work to. While a statistical regression model might have been able to perform this task, given the right training set, as outlined in Section 2.4 we had significant concerns about the usefulness of such regression models in practice beyond the laboratory and the investigation of them was beyond the scope of this work.

Therefore in common with a number of other researchers [9, 84, 116, 187] we decided to simplify our approach and focus our attention on CPU utilisation as a proxy for overall server power consumption. Previous research [15] has shown that CPU utilisation is directly proportional to the power consumption of the server as a whole and supporting this assertion for memory usage, Chen and his colleagues found that "research on the power consumption of memory reports that the power consumption of memory remains constant

regardless of the workloads" [31]. From practical experience, we are also confident that CPU varies roughly linearly with network IO given the large amount of serialisation and deserialisation work that characterises modern RPC protocols and libraries. We investigated the relationship between disk IO work and CPU and in our testing, we found that a linear relationship holds for that resource type too (see Chapter 8). Hence we have a high degree of confidence that this simplifying assumption is correct and that CPU utilisation is a good proxy value for the relative power consumption of a server computer.

This assumption allows us to use CPU utilisation of the server host as a whole, compared to the CPU utilisation of the architectural elements used to process application requests, as a reliable proxy for the ratio of the energy consumption of the server to the energy consumption required to process application requests.

In the next section of this chapter, we explain how we calculate estimations of energy allocation for application requests.

6.8 Calculating an Energy Allocation Estimate

6.8.1 Our Approach for Estimating Energy Allocation

The key novel element in the system is clearly the Energy Estimator, which implements the processing required to fulfil the purpose of the system. This element is a numerical calculator which combines data from the trace records, resource usage records and energy usage records in order to produce a reliable estimate of the amount of the energy of the underlying host which should be allocated to a specific request which the application under consideration has processed.

We illustrate the approach to creating an energy allocation estimate using the graph in Figure 6.3, which shows the cumulative CPU usage over time for two architectural elements *E1* and *E2* along with the corresponding cumulative CPU usage for the host they are executing on. The x-axis of the graph is time and the y-axis is cumulative CPU usage.

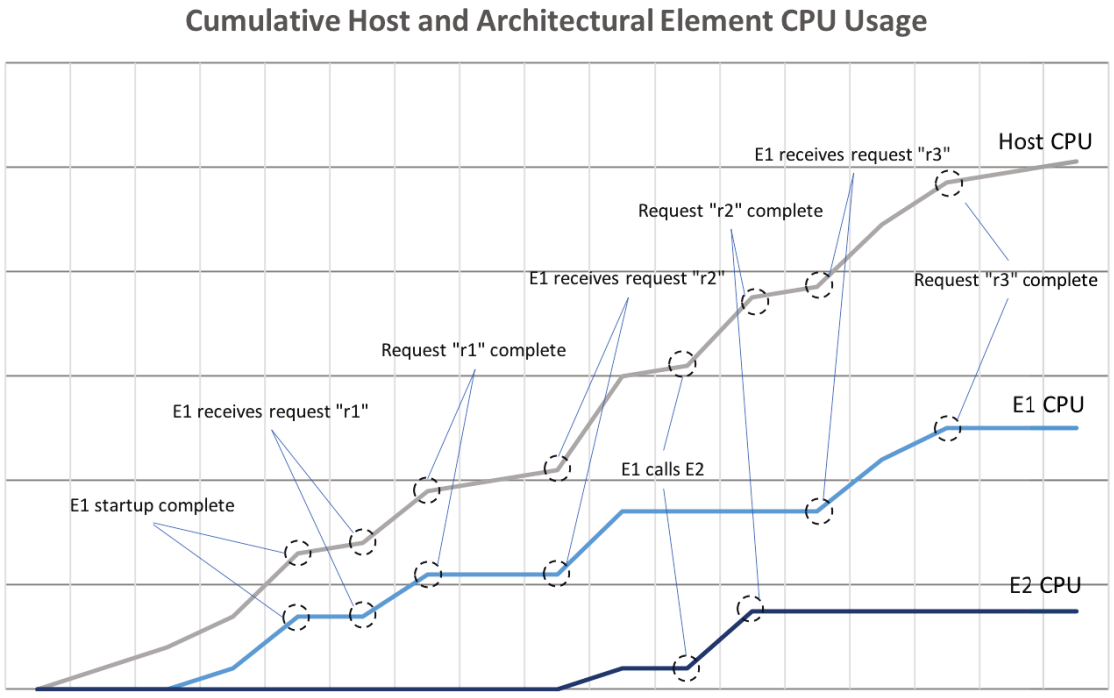


FIGURE 6.3: Cumulative CPU for Request Handling

The graph shows how CPU usage accumulates for an individual architectural element when it handles requests and how the CPU usage of the individual elements combine, along with other workload on the machine, to affect the host's CPU usage.

Starting from the left, the host and element *E1* start up and use CPU during this process before becoming largely idle for a short period and so their CPU usage curves flatten.

Next element *E1* receives a request, which we name *r1*. As can be seen, processing this request causes *E1* to consume CPU rapidly, hence the CPU consumption curve becomes much steeper for a short period and the host's CPU usage curve follows a similar shape, reflecting the effect of *E1*'s CPU usage on the host. After request *r1* has been processed, the CPU utilisation curves flatten again, reflecting the idle state that the application elements and the host are in.

The next part of the graph shows the arrival of another request, which we name *r2*, processed by element *E1*. Similar to the previous request, this results in a short burst of CPU consumption by *E1*. However, in this case, *E1* calls a second architectural element, *E2* as part of the processing of the request. At this point, this CPU consumption curve for

E1 flattens as it is no longer consuming CPU, but *E2*'s CPU consumption starts increasing rapidly for a short period. Once *E2* has completed processing, the host CPU curve flattens, indicating that request *r2* has been processed.

Finally, element *E1* receives a third request, which we name *r3* and its CPU usage graph again becomes steep for a period as the request is processed. Note that the CPU usage for processing request *r3* is significantly greater than that which was recorded for processing *r1* (and in fact *r2*). The host's CPU usage can be seen to increase in step, reflecting the impact of processing the request on the CPU usage of the host.

By comparing the shape of the host CPU usage graph to the usage graphs of the application elements it is possible to assess the amount of the host's CPU consumption which is attributable to the application elements. In this example, the host CPU usage graph has quite a similar shape to application elements beneath it. However, careful inspection reveals that its slope is nearly always steeper than that of the application elements, reflecting the other workload that the host is undertaking in addition to that of the architectural elements.

In this case, the fairly close match of the host usage curve to the application elements suggests that relatively little other workload is active on the host. In the most extreme case, the host utilisation curve would be flat at the top of the graph, indicating close to 100% utilisation and a very significant amount of other work being executed on the host in parallel with the application workload which we are studying.

It is this host CPU to application element CPU ratio which is key to our approach to allocating energy usage. As explained above, we view CPU usage as a good proxy for the energy consumption of a host. Therefore if an application's elements are consuming (say) 65% of the host's CPU usage for a period, then we know that it is consuming roughly 65% of the energy of the host during that period.

We use this insight as the key to designing an algorithm to allow us to estimate energy allocation for individual application requests, as explained in the next section.

6.8.2 A Specification for Energy Allocation Calculation

We can express the specification for this algorithm as a series of simple equations as shown below.

As explained earlier usage records are *Traces* which contain *Spans*. A *Trace* represents an inbound request to the system and can be considered to be a tuple of the form

$$Trace :: (tid, st, et, S) \tag{6.1}$$

where tid is a unique identifier for the trace record, st is the start time of the trace, et is the end time of the trace and S is the set of spans that the trace contains.

A span is a single invocation of an internal system element and so every trace contains at least one span (i.e. $\forall t : Trace \cdot t.S \neq \emptyset$) but most traces will contain a number of spans, related to each other, as illustrated in Figure 6.2. A span can be considered to be a tuple of the form

$$Span :: (tid, sid, st, et, a, pid) \tag{6.2}$$

where tid is the identifier of the trace this span is contained within, sid is the unique identifier of this span, st is the start time of the span, et is the end time of the span, a is the address of the architectural element that the span is invoking and pid is the unique identifier of the parent span of this span, if there is one (this is an optional element in the tuple).

For both traces and spans, times are assumed to be recorded to millisecond precision and are conventionally represented as the number of milliseconds since an agreed point in time, by convention 00:00:00 on the 1st January 1970 [83]. The approach described here does not require a particular granularity of timestamp for start and end times, but we make this assumption of milliseconds in the equations below as some of the calculations require the length of the period between the two.

The address of the architectural element can be any unique identifier which can unambiguously identify an architectural element and can be captured reliably by the tracing system in use. In practice, this is usually a network address of the form $ipaddr : port$ where $ipaddr$ is an IP version 4 address (e.g. 156.76.45.32) and $port$ is an IP network

port (e.g. 9745). However, our approach does not require addresses of this form and other addressing and identification schemes could be accommodated if required. We discuss this important detail further in Chapter 7.

Given these structural definitions, we can now define how the energy estimation process should work.

Given a trace t we first define our period of interest p to be the period between $t.st$ and $t.et$ and the set of spans of interest S to te $t.S$.

Next, we define the set R of runtime application elements which were involved in processing the requests described by the set of spans in S, which are identified by their network addresses. By "runtime application element" we mean any execution container for which resource usage statistics can be reliably obtained, such as a container, a process or a thread.

Let us assume we have a function $ator : (A, \mathbb{Z}) \rightarrow R$ which can map a network address and a point in time from set A to its corresponding runtime element at that point in time, represented by an integer timestamp. Such a mapping can be assumed to be available from the runtime platform in use.

We can now define the set R to be the set of runtime elements corresponding to the network addresses found in the spans within the trace:

$$R = \{ator(s.a) \mid s \in t.S\} \tag{6.3}$$

Let us now consider how we estimate the CPU usage of a runtime element during period p.

We make a key, but reasonable, simplifying assumption that a runtime element will only process workload on behalf of a single trace during a trace's execution period. While this would be a significant constraint with traditional monolithic architectures, it is common practice in microservice-based deployments to deploy instances of microservices for the purpose of monitoring and testing in the production environment [67] and microservice

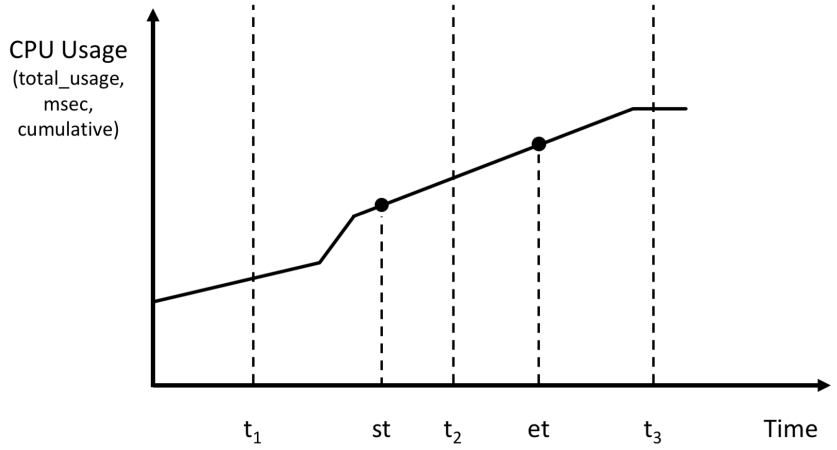


FIGURE 6.4: Cumulative CPU Usage Graph

infrastructure, such as Istio is specifically designed to allow this [76]. Hence, temporarily dedicating a small part of the deployed application estate to processing requests for energy testing is not difficult or onerous in most modern microservice based systems.

For each runtime element, we have a series of resource usage samples at fixed intervals and we need to use interpolation between these points to estimate the CPU usage of runtime elements at arbitrary points in time. The process is illustrated in Figure 6.4. The figure shows an example CPU usage statistic for a particular operating system process, captured as a cumulative usage value over time. Points in time t_1, t_2 and t_3 are sample points when the CPU usage statistic is reported. Points st and et are the start and end times of the period p we are interested in. We indicate the value of the CPU usage statistic at a point in time using subscript notation, where C_{t_1} is the value at point t_1 and C_{st} is the value at point st.

Given r is a runtime element then we can describe the CPU usage of the element using the equations below, where $C_{r,t}$ indicates the cumulative CPU usage for the element r at time t.

Let $C_{r,st}$ be the cumulative CPU usage for the element r at point st defined by

$$C_{r,st} = C_{r,t1} + \left(\frac{C_{r,t2} - C_{r,t1}}{t2 - t1} \times (st - t1) \right) \tag{6.4}$$

Let C_{et} be the cumulative CPU usage at point et defined by

$$C_{r,et} = C_{r,t2} + \left(\frac{C_{r,t3} - C_{r,t2}}{t3 - t2} \times (et - t2) \right) \tag{6.5}$$

Then the CPU usage of runtime element r can be defined as

$$C_r = C_{r,et} - C_{r,st} \tag{6.6}$$

Given h which is the host that a runtime element was executed on, and st and et, which are the start and end times of a period respectively then we can describe the CPU usage of the host in a similar manner.

Let $C_{h,st}$ be the cumulative CPU usage for the host at point st defined by

$$C_{h,st} = C_{h,t1} + \left(\frac{C_{h,t2} - C_{h,t1}}{t2 - t1} \times (st - t1) \right) \tag{6.7}$$

Let C_{et} be the cumulative CPU usage at point et defined by

$$C_{s,et} = C_{h,t2} + \left(\frac{C_{h,t3} - C_{h,t2}}{t3 - t2} \times (et - t2) \right) \tag{6.8}$$

Then the CPU usage of host s between times st and et can be defined as

$$C_{h,st,et} = C_{h,et} - C_{h,st} \tag{6.9}$$

Having estimated the CPU usage of the runtime element during its execution period and the CPU usage of the host during the same period, we can use the ratio of the two, combined with the model-based or physical measurement estimates of the host's energy consumption over the same period. Power indicates the rate of energy consumption at a moment in time and is a constantly and rapidly fluctuating value. We rely on averages of its value over time in order to find a representative value to use for our calculations.

In the situation where we have physical power metrics from the data centre infrastructure, then it is normal to find a Data Centre Infrastructure Management (DCIM) software platform deployed and we can extract power consumption metrics from this platform directly,

using the API that such products expose. Different DCIM platforms offer different facilities and, in some cases, we might need to perform our own aggregation of device power readings in order to produce power consumption metrics suitable for our needs. In other cases, the DCIM platform will have performed this aggregation and normalisation of the data already and we will be able to retrieve it via the API. In either case, the approach would be to produce a cumulative power usage metric for the hosts of interest in the managed environment, to allow a power usage average to be calculated in a similar way to the treatment of the cumulative CPU usage metric.

If a model-based approach needs to be used then a slightly different calculation approach is required, as shown below:

Let $T_{h,st,et}$ be the total possible CPU consumption of host h between start time st and end time et. Let N_h be the number of CPUs that the host contains.

$$T_{h,st,et} = N_h * (et - st) \tag{6.10}$$

Let $U_{h,st,et}$ be the percentage CPU consumption of host h between start time st and end time et.

$$U_{h,st,et} = \frac{C_{h,st,et}}{T_{h,st,et}} \tag{6.11}$$

The power consumption estimate in watts, P_h, can now be extracted from the relevant SPECPower results for the model type of the host machine using $U_{h,st,et}$.

Let $E_{h,st}$ be the energy consumption (in joules) for the host between start time st and end time et (assumed to be expressed in milliseconds, as explained above).

$$E_{h,st,et} = (et - st) \times 1000 \times P_h \tag{6.12}$$

Given these supporting definitions, we can now define the specification for the energy estimation process to be

$$C_t = \sum_{r \in R} C_{r,t.st,t.et} \tag{6.13}$$

$$E_t = \left(\frac{C_t}{C_{h,t.st,t.et}} \right) \times E_{h,t.st,t.et} \tag{6.14}$$

The energy allocation for a trace is the sum of the CPU usage for the runtime elements that processed its workload during its execution time, as a percentage of host resource usage during the trace execution period, multiplied by the estimated host energy usage during the trace execution period.

An algorithm which implements this specification will, therefore, allocate the energy used by the underlying host fairly across the application execution traces that it is executed for, so allowing fair comparisons to be made between different application requests to motivate design for energy efficiency and allowing recharging of energy costs in a fair and systematic manner.

6.9 Conclusions

In this chapter, we have explained the problem of providing energy estimates to a software architect and explored how this can be achieved in a realistic enterprise-computing environment through the use of a combination of resource consumption statistics, host energy consumption statistics and data centre efficiency factors.

So far our discussion has been largely implementation agnostic. We have assumed the availability of the information we need, which presupposes a reasonably modern, mainstream enterprise application development and deployment environment (such as .NET on Windows or Java on Linux), but we have not limited or constrained our design by choosing specific technologies to use for data collection and analysis. This means we have a specification for the software we need to build, which we could now implement using a choice of technologies for each part of the solution.

Realistically, when we choose technologies for each part of the solution they will solve parts of the problem for us and also constrain the solution and bring limitations and difficulties. This is why presenting the implementation independent form of the design is important so that the underlying ideas can be explored and applied in a range of technology environments. However, we are aware that the design, and possibly its effectiveness, will be altered by the specific implementation choices we make when we build it.

In the next chapter, we explain how we built a proof-of-concept version of Apollo, the technology choices we made, the problems we encountered and how we solved the problems to create a realistic and viable application energy calculator.

Chapter 7

Implementation of Application Energy Monitoring

7.1 Introduction

The logical design of the Apollo energy allocation system was presented in Chapter 6 and showed how, in principle, we can build a useful calculator to fairly allocate energy usage of a collection of host machines, on the basis of the individual application requests processed in a period of time (rather than just the total resource usage of application elements over a period). Such a calculator can provide a tool for a software architect to understand the energy consumption implications of their architectural decisions and can guide them towards higher energy efficiency for the applications.

The logical design of the calculator is independent of specific technologies and does not specify the details of the implementation, simply the operations that must be performed. Hence, as we are now to implement the calculator, the logical design acts as our specification. Our proof of concept implementation can be implemented in a number of ways using a number of different technology choices and detailed design decisions.

In this chapter, we explain how we went about the task of implementing our proof-of-concept calculator, the problems we set out to solve, the choices we made, some of the problems we had to solve and some of the tradeoffs that were necessary.

7.2 Defining the Context

In order to allow an effective solution to be created in a reasonable amount of time, we needed to narrow the possible design space for our problem, to allow us to focus on the key decisions specific to our research and avoid being distracted by the generic decisions that all software projects need to make.

These basic context-setting decisions constrain the complexity of the solution. The decisions that we made, and the justification for each, are explained below:

- As mentioned in the previous chapter, We will focus on the domain of *microservice-based information systems*, such as those found in large enterprises, Internet-facing systems and Internet-oriented startup companies. We will not specifically exclude the approach being used with other architectural styles, but where a design decision is required, we will assume that the approach will be used with a microservice-based system. A microservice based approach makes resource monitoring easier because resources can be monitored at operating system process, rather than thread, level, which is simpler and in some cases, more accurate. This architectural style is rapidly becoming a mainstream choice and so we do not view this choice as a significant limitation for industrial application. In principle it could be relaxed later, although we recognise that this would complicate our implementation significantly and could require some rethinking of the approach.

- We will only consider our application's *microservice elements and only implement monitoring of microservices that communicate via RPC calls* (typically JSON over HTTP). This is a simplifying assumption that could be relaxed at a later point, but reduces the scope of our initial implementation. We make this assumption because it makes implementation and validation of the initial solution more tractable and allows its usefulness to be validated more quickly that would otherwise be the case. The restriction is also easy to relax at a later date because the implementation approach that we chose allows tracing over any application elements that communicate over network connections including those that use gRPC, HTTP, B3, Thrift, application messaging, Kafka and others. This means that while the initial implementation does

have a limitation, it is one that does not affect the research results and can be relaxed relatively easily later without changing any of the fundamental characteristics of the solution.

- We will assume that the primary technology "stack" used to implement the system will be *enterprise Java*, meaning software written in Java, running on the JVM, using common open source frameworks and libraries like Spring Boot, Hibernate, Dropwizard, Spring Data, Apache Commons and so on. Our approach is not specific to these technologies, but they simplify implementation considerably, allowing us to focus on the research problem. While most of the technologies that we use in our solution (such as Zipkin and MySQL) would support applications written in many technologies (including Java, .NET, Go, Ruby and Python) supporting a "polyglot" multi-stack application adds significant amounts of implementation work and makes the validation process more complex. However all of this additional work is unrelated to the fundamentals of the approach and does not add to the validity of the research. We chose the enterprise Java stack because of its very wide use in many different industrial settings and the very wide support it enjoys in the open source community, meaning that there is a rich open source ecosystem for us to use.

- We will assume that the primary *execution platform is Linux on Intel*, as this is something of a defacto standard for running large-scale enterprise Java systems. Where it is possible we will try to make the approach execution platform agnostic (in particular to allow for Windows, another common server execution platform in the enterprise) but again, where a decision needs to be made, we will assume a Linux on Intel host. With today's virtual machine based technology, the operating system is the least important decision in our design space. We choose Linux because it is widely supported, widely used for microservice Java applications in industry and has the best command-line (and so scriptable) tools for performance and resource usage monitoring, which are central to energy efficiency monitoring. The use of Linux is a simplifying limitation, which would be straightforward to relax later with some relatively simple porting and scripting work.

All of these decisions align us with mainstream industry practice and provide practical options for applying our approach to industrial systems while narrowing our design decisions

to those related to our research problem, rather than generic concerns.

We discuss each of the more specific design problems we needed to solve and the decisions we made to solve each one in the following sections of this chapter.

7.3 Tracing Application Execution

Our aim is to provide the application architect with insight into the energy consumption implications of their design decisions, and so simply measuring the energy of the infrastructure hosting the application does not provide enough information for this purpose. The architect needs to understand the energy implications of the execution of different parts of their application, and most importantly, the energy consumption of certain types of workload. This will allow them to understand the energy-intensive parts of their application and workload and focus on improving the energy characteristics of these application elements. Comparing the energy characteristics of different elements and workloads will also allow them to see the implications of particular design choices.

This requirement means that we need some way of tracing the execution of workload through the application to produce data equivalent to the traces and spans that we saw in Chapter 6. We considered a number of ways of achieving this.

Application specific tracing could be provided by an application as part of its implementation and write special-purpose log files or database entries to record how an application request is processed through the application's elements. While straightforward from our perspective, this is a complex and potentially time-consuming feature to add to an application and would be quite expensive to add to an existing system. We think that this approach would be unlikely to be adopted in practice.

Application Performance Management (APM) tools, such as AppDynamics, New Relic and Dynatrace [11, 46, 124] already perform application request tracing to allow them to measure and estimate application performance characteristics. Initially using such a tool as the basis of our approach was our preferred option. In practice though, while attractive to practitioners who were already using the particular tool we would choose, it is a significant barrier to everyone else due to the cost and complexity of deploying these

tools. While we see our work as a potential extension of APM tools, perhaps providing them with a new dimension to their facilities, we decided against basing our approach on one of them.

Microservice tracing systems like the open source Zipkin and Jaeger [77, 132] projects were a third alternative that we considered. These systems are used by application developers to provide standardised trace data about the execution of their applications and provide collection and analysis infrastructure to allow the trace data to be easily used. When we initially investigated them, they appeared to solve part of the problem of implementing application specific tracing, but still left the application developer with significant work to do. As we investigated these open source products further we found that they are supported by or integrated into many commonly used application frameworks (for example Zipkin is already integrated into libraries for about 10 languages, including Java, Python, C# and JavaScript, and just considering Java, it is integrated into more than 15 well-known application frameworks including Spring Boot, Dropwizard, Google RPC, Apache HTTP Client and Jersey). When utilising a pre-integrated framework, using these tracing systems is very straightforward from an application developer's perspective and just involves starting the data server to receive the trace data and setting some configuration parameters in the framework configuration.

After some experimentation we found that the Zipkin tracing system worked very well for a set of Java microservices and its database was easy to query to extract the trace information we needed. We concluded that the reliability, easy availability, low implementation overhead, ease of use and its large number of existing integrations with widely used application frameworks made Zipkin a good choice for our work.

7.4 Estimating Resource Usage of Application Workload

Once we can reliably trace the execution of application requests through the application elements involved in processing them, we can move to consider how to estimate the resource usage of those application elements in order to work out the resource consumption of the requests processed by the application. The key requirement here is to be able to collect reliable samples of the resource usage of the application elements on a very

frequent and predictable basis (e.g. every couple of seconds). Ideally, the samples will be in terms of cumulative usage rather than usage at that point in time, as these are much easier to use for our purposes.

The only practical source of application resource usage statistics is the application execution platform. There are a number of sources of statistics that we could use, each with slightly different characteristics.

The simplest option is to use *native operating system tools* such as `sar` and `pidstat` on Linux and `procmon` and `perfmon` on Windows. In principle, these tools can collect resource usage statistics for application processes on the machine. However, in practice, our industrial experience, and recent investigation for this work, suggests that they are really intended for collecting host-level statistics or for interactive investigation of a performance problem on a machine by a skilled administrator. They do not provide an easy way to get a reliable stream of samples of cumulative usage over time written into an accessible form.

An alternative is to bypass the tools and access the *operating system performance counters* directly. Like most modern operating systems, Windows and Linux both implement a set of performance counters in their kernels, which are used for monitoring performance and throughput of workload executed by the machine. These counters are used by the operating system tools to provide the data they need to operate and so by accessing the counters directly, we can avoid any limitations in the tools and still achieve consistent results. Linux provides access to its performance counters via the very convenient `/proc` file system, which exposes all of the kernel's counters for global and process-specific metrics, via a pseudo file system interface (which can be read using standard text processing tools or through the standard file system API). Windows provides access to its performance counters via the `perfmon` tool or through an API which is accessible via PowerShell (the modern Windows scripting language) or a conventional programming language. In principle we can build any collection tool we want using these interfaces or we can use metrics collection servers such as `telegraf` or `collectd` [56, 71] to read them automatically and store them in a suitable database for us. During initial research, this approach was our preferred option. However, in practice, we found it quite difficult to get

a usable set of statistics for our application. The main problem we faced was that the collector does not know which workload on the machine belongs to a particular application. Therefore we had to collect everything at operating system process level, which potentially generates huge amounts of unnecessary data. We also found that the business of building a reliable collector was more difficult than initially assumed and that linking the trace data to the dataset we could collect easily from operating system counters was quite difficult to do reliably. These difficulties were not insurmountable but led us to consider whether there were other options we could consider.

The third option we investigated was to use *Docker* [45] as a packaging technology for the application elements and to allow us to collect resource usage statistics. Docker is an operating system virtualisation technology which uses operating system mechanisms (such as "cgroups" and "kernel namespaces" on Linux) to isolate processes from each other, providing the illusion that each is running on a separate machine. Docker also provides a packaging convention that allows software to be packaged into reusable packages called "images" which are combined at runtime to form runtime environment known as a "container". Docker also provides a set of management APIs and tools to allow containers to be interrogated, managed and controlled. Docker is rapidly becoming a de-facto standard in the industry to package, deploy and manage microservices both on general host computers (where Docker becomes an execution environment on top of the operating system) and more abstract, container-based platforms like Kubernetes [95] where Docker forms part of a sophisticated platform providing quality properties like scalability and resilience, across a cluster of host computers.

Our interest in Docker lies in its ability to provide a low-overhead, isolated environment for running application elements (in our case, microservices specifically). If we can assume that all of the microservices within our system are packaged and then run as Docker containers then it provides us with the ability to extract accurate resource utilisation statistics for the microservice running in the container. Another benefit of using Docker is that each container has its own network (IP) address which can be found via the runtime metadata available via Docker's management API. This greatly simplifies the process of relating the resource usage data to the request traces from Zipkin as the network address is a shared piece of information between the two.

We believe that requiring application elements (the microservices) to be packaged and run as Docker containers is realistic and reasonable, given its wide and growing adoption in industry, particularly for microservice-based systems. Therefore this combination of factors resulted in us deciding to use Docker as the basis for collecting application resource utilisation statistics.

A useful side effect of packaging the application as a set of microservices in containers is that it makes the utilisation of the trace data simpler. The trace data identifies the application elements by network address (typically IP address and port number). Each container in a Docker deployment (strictly a Docker network) has its own set of one or more IP addresses. Therefore, by having each application element in a separate container, we can rely on them listening on different IP addresses and that the container to IP address mapping is available from the Docker metadata. Hence this approach to application packaging and deployment makes the mapping of trace data to application elements straightforward.

The second part of collecting utilisation statistics is how they are extracted from the execution platform and stored. In our case, our experimentation with Docker quickly revealed that a number of open source projects, including *cAdvisor* [63] and *Telegraf* [71], provide close integration with Docker and can extract and store the utilisation statistics data in different database systems.

After some experimentation, we chose to use Docker with Telegraf, storing utilisation statistics in the *InfluxDB* [70] open source time-series database to provide us with reliable application element resource utilisation statistics gathering.

7.5 Estimating Resource Usage of the Host Platform

Estimating resource usage of the underlying host platform is simpler than gathering the same information for the application elements, as host-level statistics gathering is a mature and widely used technology, provided by all major operating system platforms.

In our case, we need to achieve reliable resource utilisation sampling of the host platform - the underlying Linux operating system - and have the resulting statistics stored in a

database that allows us to extract them through a query interface to support the calculation process. Ideally, if the statistics are in a similar form to the statistics for the application level resource utilisation (e.g. the same timestamp convention and data types used), then this is likely to make the implementation of the calculator easier.

Our earlier investigation of the *Telegraf* data collection server to extract and store application-level utilisation statistics revealed that it can be configured to extract and store host-level utilisation statistics too and that this facility is provided as part of the standard distribution of the (open source) product. When we tested the host resource utilisation statistics feature of Telegraf we found that it was straightforward to use and reliably stored accurate statistics in the same database as the application-level statistics. The host-level statistics used the same basic conventions as the application-level statistics (e.g. they were both cumulative utilisation statistics using the same timestamp conventions).

Therefore we were able to solve the host-level resource utilisation statistics gathering problem by simply extending the configuration of the Telegraf server to extend the dataset it stores, to include host-level statistics.

7.6 Estimating Energy Usage of the Host Platform

As explained when we discussed the motivation for this work in Chapter 1 the field of energy estimation is relatively young and reliable approaches for energy estimation of individual devices and applications are only just emerging. Our work does not intend to address the problem of estimating energy consumption for the underlying computers, but rather requires a reliable energy consumption metric to be available.

As explained in Chapter 6 there are two main approaches available to us that can provide energy usage of our underlying host platform, physical energy consumption metrics made available via a DCIM platform and model-based energy consumption estimation using machine utilisation levels and published benchmark results.

In many industrial situations, a DCIM platform will be available and energy consumption metrics for all or a significant subset of host machines will be available through it. However, in our research environment we did not have access to such a platform and in some

TABLE 7.1: SPECPower 2008 Benchmark Results for Dell R730 PowerEdge

Machine Load	Power Consumption (W)	W / %
Active Idle	44.6	-
10%	84.8	8.48
20%	102.0	5.10
30%	120.0	4.00
40%	136.0	3.40
50%	150.0	3.33
60%	163.0	2.64
70%	181.0	2.58
80%	205.0	2.56
90%	238.0	2.64
100%	272.0	2.72

industrial situations, this will be the case too. While the state-of-the-art in organisation design is to integrate development and operations groups [87], they are frequently still separate and so even when a DCIM platform is available, software architects may well not be able to access it easily.

Therefore we decided to use a model-based approach to estimate the energy consumption of the host, but to ensure that it was easily replaceable with alternative models or with queries to a DCIM platform if one was available.

The approach used to estimate the energy usage of a host was explained in section 6.8.2 and relies upon published power consumption benchmark results associated with the SPECpower_ssj 2008 benchmarks [96]. An example dataset for power consumption for a specific model of server host is shown in 7.1.

We use this style of benchmark data combined with the machine type executing our application workload and the utilisation level metrics of the server executing the load to estimate host server energy consumption at a particular point in time, as explained in section 6.8.2.

The third column in the table is a derived value we have added to show the power consumption per percentage point of server utilisation at each level of utilisation. This illustrates the need to keep servers busy from an energy efficiency perspective as it can be seen that 1% utilisation when the machine is quiet is three times more expensive in energy consumption terms than 1% utilisation when the machine is busy. We investigate this observation further in Chapter 8 when we describe how we validated the implementation of Apollo.

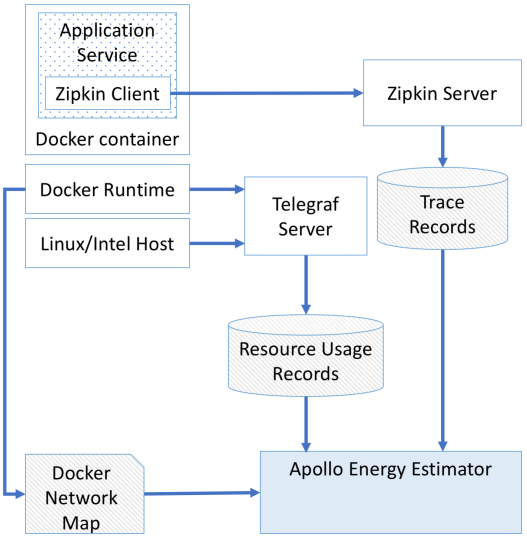


FIGURE 7.1: Design of the Apollo Energy Calculator

7.7 Implementing the Calculator

7.7.1 Design of the Calculator

The design of the Apollo calculator is shown in the simple block diagram in Figure 7.1. The system element filled with the fine dotted pattern represents the architectural elements of the application (i.e. an application microservice), the system elements filled with the fine cross-hatching are data elements, while the Apollo Energy Estimator is filled with the light solid fill. The unshaded elements are the third party technologies which are reused unchanged as part of the implementation.

The design elements of the calculator and their responsibilities are summarised in Table 7.2

Most of the system described here is open source software (Docker, Telegraf, Zipkin, InfluxDB, MySQL and Linux) and so the only significant piece of custom software that had to be developed for this investigation was the Apollo Energy Estimator module. The other significant software development effort was configuration and scripting to combine the different pieces of software into a single system. We describe the software design of the Estimator module in the next section.

TABLE 7.2: Apollo Energy Calculator Design Elements

Design Element	Responsibilities
Application Service	This element represents the regular microservices that comprise the application under investigation. The implementation of these services are under the control of the development team and they have the responsibility to generate Zipkin trace records (via the Zipkin Client library) to record their activity (although this will usually be achieved automatically through use of an application framework like Spring Boot).
Zipkin Client	A trace of the invocations to and between application elements is required and as explained above, the Zipkin tracing system is used to achieve this. The Ziplin Client is a client programming library used by Application Services to generate trace records and forward them to the Zipkin Server for storage. The application code may invoke this library directly or it may be invoked automatically by an application framework like Spring Boot or Drop Wizard.
Zipkin Server	The Zipkin server receives and stores the trace records from the Application Services. One Zipkin Server is used for all of the Application Services in a monitoring context.
Trace Records	The Zipkin Server persists the trace records in a well defined schema in a database. In our case we used MySQL as the database for the trace records.
Docker Container	All application elements need to run within Docker containers. This allows metadata about the elements to be retrieved and resource usage statistics to be gathered. This container contains the Application Service (each service is packaged in a separate container to allow it to be monitored separately).
Docker Runtime	The Docker Runtime is part of the Docker system software package and provides the control and monitoring of the Docker containers in the application and provides runtime statistics for the containers, which in our situation are streamed to the Telegraf Server for storage.
Linux/Intel Host	The application runs within Docker on the underlying host machine(s) and we have chosen to use an Intel host running the Linux operating system, due to the maturity of both Java and Docker on this platform.
Telegraf Server	The Docker platform produces a stream of resource usage statistics for the containers that it is executing. The Telegraf open source metrics collection agent collects these metrics and stores them in a timeseries database for easy retrieval.
Resource Usage Records	The Telegraf Server generates a stream of resource utilisation statistics by constantly querying the Docker Runtime and the Linux Host. These statistics records are persisted to a database for later use. The datastore used for the Resource Usage Records is InfluxDB, an open source timeseries database.
Docker Network Map	The Zipkin traces and the resource usage records identify the runtime elements of the system in different ways; the Zipkin traces are collected at the network level and so identify elements by IP address and port number, while the Docker resource usage statistics are identified by Docker container ID. Hence metadata is needed to link the two together and this is the purpose of the Docker Network Map which is metadata available from the Docker Runtime (through the `docker network inspect` command) which allows us to find the IP address(es) in use by each container during the execution of the application.
Apollo Energy Estimator	The Apollo module collects data and implements the energy allocation algorithm described in Chapter 6. This module is the primary software that we have implemented as part of this research (along with an example application we use for validation, which we describe in Chapter 8).

7.7.2 Technology Choices

In order to implement our solution efficiently, in a manner that would allow application in a mainstream industrial environment, we have chosen a number of pieces of reusable open source technology to base it upon. These technologies and the reasons for choosing each are listed below:

- **Docker** - the Docker container system is used to simplify the process of collecting resource statistics for our application elements. By packaging our application elements in containers we can use the statistics collection features of the Docker subsystem to collect accurate component-level statistics, while also aligning with mainstream industrial practice. While there are other container systems with similar features to Docker, they do not have particular technical advantages and do not have the degree of acceptance and standarisation that makes Docker a defacto mainstream standard.
- **InfluxDB** - the InfluxDB database is an open source timeseries database, used in our solution to collect resource usage statistics. A timeseries database is important as we need to perform efficient temporal queries and this style of database provides strong support for this requirement. While there are other timeseries databases availabel, we chose InfluxDB because it integrates closely with the Telegraf statistics collection server (see below).
- **Linux** - as explained in Section 7.2 we chose Linux as our operating system to align with mainstream industrial practice and because it provided the best command line (and so scriptable) resource usage collection facilities.
- **MySQL** - the MySQL relational database is used by our solution to store the Zipkin execution traces. Zipkin supports a limited number of databases for trace storage (Cassandra, ElasticSearch and MySQL) and we chose MySQL due to the relative ease of running the server and running complex queries against the data, which is necessary for our application.
- **Telegraf** - our solution needs a mechanism for collecting resource usage statistics for both the application under investigation and the server machines that it is executing on. There are a number of ways of achieving this including custom scripting,

operating system tools and resource monitoring servers such as Telegraf, DD-Agent and Collectd. After a short period of experimentation we quickly concluded that we needed to use a resource monitoring server given the complexity and effort required to collect statistics from both Docker and the operating system ourselves. There are a number of such open source products available but after a brief survey of those available, we concluded that Telegraf (and InfluxDB) were widely used, reliable and suitable for our needs and so we decided to use it.

- **Zipkin** - a key part of our approach is using execution trace records to identify the application elements involved in processing an application request. It is possible to create such a tracing system from scratch, but this would be a significant amount of work which would not contribute towards our research goals. Therefore we needed to find a tracing system to use. As we outlined in Section 7.2 there are a small number of possible approaches but we quickly found that the practical solutions available to us were two open source systems (Zipkin and Jaeger). We chose Zipkin simply because we found that it was reliable, accessible and had a friendly support community. Almost certainly Jaeger would have been a workable solution too, but Zipkin was an effective solution for our needs.

While many of these technology decisions could be changed fairly easily, without significantly affecting our implementation, the choices we have made are pragmatic, allow efficient implementation and align with mainstream industrial practice and so support our research goals.

7.7.3 Data Design

In the earlier subsections, we explained how we would solve each of the data gathering problems in order to create the datasets that the energy estimator requires in order to perform its calculations. We now need to define how that data is used by the calculation code.

The data model for the data from the different sources is shown using the UML class diagram in Figure 7.2 and the data elements are described in Table 7.3.

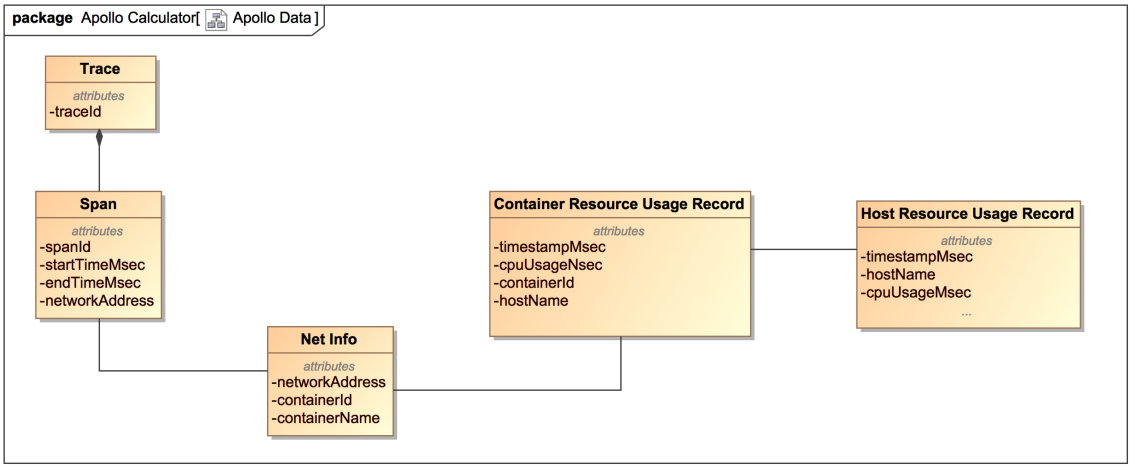


FIGURE 7.2: Data Structure for Apollo Energy Estimator

TABLE 7.3: Apollo Energy Calculator Data Elements

Data Element	Description
Trace	The *Trace* is just a simple container for a set of *Spans*. While the Zipkin data item does contain more information, from our perspective, we are only interested in it having a unique ID. This is sourced from the Zipkin dataset.
Span	Contains the information recording a specific invocation of an application element. One *Span* is written for each application element invoked as part of a trace. Identifies the application element by *networkAddress* (an IP address in our case) and records the start and end time of the invocation in milliseconds. Composed into exactly one *Trace* and so contains its ID to record the relationship. This is sourced from the Zipkin dataset.
Net Info	Provides a mapping between network addresses, (Docker) container IDs and container names. Any network address maps to exactly one container ID and one container name. This is sourced from the Docker Network Map dataset.
Container Resource Usage Record	Records the resource usage of a specific container at a point in time. Contains the sample time as a millisecond timestamp, the cumulative CPU usage in nano-seconds measured at that point, the ID of the container being measured and the hostname of the computer it was executing on. This is sourced from the Docker resource utilisation statistics collected by Telegraf (and stored in InfluxDB).
Host Resource Usage Record	Records the resource usage of a host computer at a point in time. Contains the sample time as a millisecond timestamp, the cumulative CPU usage in milliseconds measured at that point and the hostname of the machine. This is sourced from the operating system resource utilisation statistics collected by Telegraf (and stored in InfluxDB).

The basic data access path required is to start with the *Trace* and iterate over the *Spans* that it contains. For each span, use the *networkAddress* to identify the service endpoint and use the *Net Info* data to map this to the *containerId* that executed the span's request. This allows the resource usage records for the container to be identified using the container ID and the start and end time of the trace. Then, to establish the resource utilisation of the host during the same period, the *hostName* attribute of the *Container Resource Usage Record* and the start and end times can be used to identify the correct *Host Resource Usage Records* to use.

This data stucture allows us to navigate through the datasets to find container and host resource utilisation data for each of the spans in a trace.

There is one remaining problem, however, which is the fact that the spans are periods of time, but the usage records are samples at a point in time, and the span start and end times will rarely coincide with the sample times. We need to solve this problem to use the data correctly.

In our specification, in Section 6.8.2 we specified the use of interpolation to estimate the CPU usage between the two points surrounding the start and end of the trace and then calculating the difference between the two cumulative CPU usage values for each container involved in handling the request.

This approach works well when the trace period crosses a number of sample intervals and we confirmed this early in the process using practical testing. However, when using the technology it is important to be aware that when the trace period is very short compared to the sample interval, and so all of the execution occurs within one sample interval, the interpolation estimation is too coarse to produce accurate results for the CPU usage. This is simply a limitation of using sample intervals for resource usage estimation and is a familiar problem in performance monitoring and analysis work.

As we investigated the technologies that could provide resource utilisation metrics, we discovered that sample intervals for real resource utilisation statistics, such as those provided by operating systems or Docker, are actually very long when compared to request durations. A typical sample interval for Docker is 10 seconds and an operating system might be 1 minute, whereas the duration of a simple request may only be tens to hundreds of milliseconds. Sample intervals can be reduced in duration (at the cost of monitoring

overhead and data volume) but realistically they can only be reduced to a few seconds, not the lengths that match request durations.

The solution to this is simply to run the workload multiple times so that the measurement is made over a number of sample intervals, at which point the error from the interpolation is insignificant in the overall estimate. With standard Docker and Linux monitoring through Telegraf we found that 5 or more sample intervals was sufficient (so 10 seconds with a sample interval of 2 seconds). While real application traces would not be this long, it is a common constraint for performance monitoring work. Given that we are running dedicated microservice instances for energy estimation and assuming synthetic testing workload run in the production environment (as explained in Section 6.8.2) then this constraint is unlikely to be a problem in practice.

7.7.4 Software Design and Implementation

The Apollo estimator does two primary things, it gathers data from the resource utilisation statistics, the trace records and the Docker network metadata and it uses the information to perform the energy allocation processing required to establish the energy characteristics of each of the inbound requests to the application, described in the trace records.

The implementation of the module is described by the UML class diagram in Figure 7.3 and the class descriptions in Table 7.4.

The interaction between the classes is fairly straightforward. Some of the code within the classes (such as that needed to retrieve and transform data into a suitable set of types to make the calculations straightforward) is intricate, but none of it is particularly complex algorithmically.

The flow of control between the modules is quite straightforward and follows the layout of the classes on the diagram; it can be summarised as:

- The Application is invoked, it loads its configuration settings from a configuration file, creates instances of the other objects, "wires" them together and invokes the Energy Calculator.

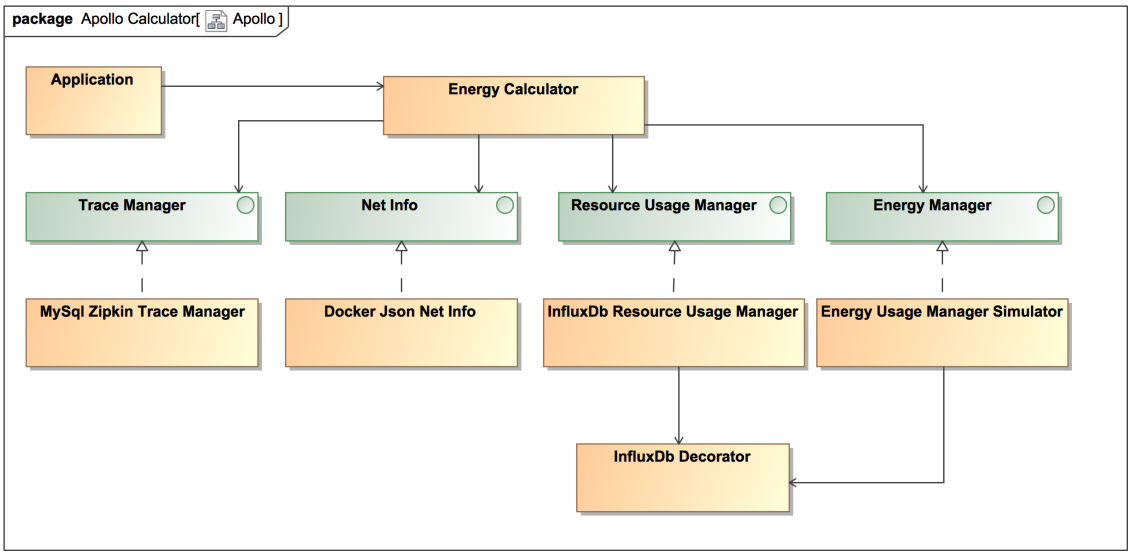


FIGURE 7.3: Implementation of the Apollo Energy Estimator Module

TABLE 7.4: Apollo Energy Estimator Module Structure

Design Element	Description
Application	This is a small "bootstrap" class with the responsibility of loading configuration settings, creating the other elements and providing an entry point for the runtime system.
Energy Calculator	Provides the main processing loop to retrieve traces from the `Trace Manager`, retrieve the corresponding data for each trace from the other elements and then perform the required calculation.
Trace Manager	An interface defining a simple API to retrieve trace records.
MySql Zipkin Trace Manager	An implementation of the `Trace Manager` interface that accesses the Zipkin Server's MySQL database to retrieve the trace records held within it.
Net Info	An interface defining a simple API to allow container IDs to be mapped to network addresses and vice versa and container names to be mapped to container IDs and vice versa.
Docker Json Net Info	An implementation of `Net Info` to provide the container name, ID and network address mappings from the JSON file produced by the Docker `docker network inspect` command.
Resource Usage Manager	An interface describing the API to retrieve resource usage information during a period of time for a Docker container, for a host machine or for a host machine where a specific container ran.
InfluxDb Resource Usage Manager	An implementation of `Resource Usage Manager` that retrieves the information from an InfluxDB that has been populated by the `Telegraf` metrics capture server.
Energy Usage Manager	An interface defining the API that provides the ability to retrieve energy usage information for a host machine by name for a period of time, or a host machine that a particular container ran on during a period of time.
Energy Usage Manager Simulator	An implementation of the `Energy Usage Manager` interface that uses a simulation based approach to provide realistic energy consumption information.
InfluxDb Decorator	A key utility class that implements the Decorator pattern and adds functions and ease-of-use features to the standard `InfluxDB` Java client class for InfluxDB. This allows all of the schema navigation and query language specifics to be isolated and hidden in this class.

- The Energy Calculator calls the Trace Manager to retrieve all of the traces in the current dataset and for each one:
  - calls the Net Info class to find the set of containers that were invoked (by mapping the network address from a span record to the container ID);
  - calls the Resource Usage Manager to retrieve the resource usage data for the containers;
  - calls the Resource Usage Manager to retrieve the resource usage data for the hosts the containers were executed on;
  - calls the Energy Usage Manager to retrieve the energy usage of the hosts during the time period of the trace; and
  - calculates an estimate of the energy allocation of the trace, based on the estimates of the energy allocation of each container that it can calculate using this information.
- The Application then reports the energy allocation of each trace using the information returned from the Energy Calculator.

We discuss how we implemented the algorithm in more detail in section 7.7.5.

The Energy Estimator module was implemented using mainstream, modern, development technology from the Java Virtual Machine (JVM) ecosystem. The primary implementation language was Kotlin [81], which was chosen for its ease of adoption, its strong support for interoperability with existing Java libraries and its functional language features. Much of the core calculation involved sets, lists and maps and functional programming features made this code significantly shorter than would have been the case using a traditional imperative style in Java.

The code was developed in a largely test-driven style with unit and integration tests being developed with the main code and an automated build, implemented using Gradle, running all of the tests before creating the delivered binary package.

A significant amount of open source software was used in the implementation, which dramatically reduced the amount of software that needed to be written. The key external libraries used in the main code were *Spring JDBC* to access the MySQL database,

InfluxDB-Java to access the InfluxDB database, *Klaxon* to provide JSON binding and *Konfig* to handle configuration settings.

The Energy Estimator module is about 1500 lines of code with about 1250 lines of associated test code. Kotlin's brevity and functional programming features contributed significantly to keeping the number of lines of code needed low. There are also about 1500 lines of associated script and resource definition code for automation of environment setup and execution.

7.7.5 The Calculation Algorithm

In Section 6.8.2 we presented the specification of how to perform the energy allocation process as a series of equations. Having designed a practical approach to collecting and estimating the input data for this process, we now had to implement an algorithm that would perform the calculation processed as specified. In this section, we highlight the significant steps that we took in this process.

The first step was to implement *Trace* and *Span* abstract data types, along with a set of types to store resource usage measurements. These class were all implemented using Kotlin's data class construct. The key decisions were to use a string representation for network addresses (to allow different sorts of network address without code change) and for all times and durations to be stored as millisecond precision timestamps since the Unix "epoch".

The next stage in implementation was to implement the *Trace Manager*, and *Resource Usage Manager* interfaces and their implementing classes. These classes provide access to the MySQL database containing the Zipkin trace records and the InfluxDB database containing the Docker and host resource usage statistics records respectively. Both of these classes are relatively sophisticated in the sense that they encapsulate the complexity of navigating the database models and mapping the data returned by the databases in order to provide a simple interface to calling code, using the abstract datatypes implemented in the previous step. The MySQL implementation of the Trace Manager relies heavily on Spring Data and the MySQL Java Connector library classes to simplify query

execution and result set handling. The InfluxDB implementation of the Resource Usage Manager uses the standard InfluxDB Java client to access the database.

Once we had reliable, tested, access to the databases, the *Net Info* interface and its Docker specific implementation was implemented. It reads a JSON file generated by the `docker network inspect` command and extracts a network address to container ID to hostname mapping from it. It uses the Klaxon open source library to provide idiomatic access to JSON data from Kotlin, which made the implementation fairly straightforward.

It was now possible to embark on the initial implementation of the *Energy Calculator* which was implemented as two classes, the main Energy Calculator class that retrieves the traces from the Trace Manager and iterates over them, and a helper *Trace Calculator* class that performs the energy allocation calculation for one trace.

The Trace Calculator makes extensive use of Kotlin's functional programming features and related data types (maps, sets and lists). The data is extracted from an SQL database, a JSON file and a No-SQL timeseries database. Therefore it can't simply be accessed via a single database query, which might have been an option if it had all been in a single relational database. Instead, this class reads subsets of the data of interest from the data stores and converts them to maps and lists structured to make the required computation straightforward (for example the Net Info data is transformed from a JSON structure to a set of maps from networkAddress to containerId, containerId to networkAddress and containerId to containerName). This then allows extensive use of functional programming constructs like `map`, `reduce` and `fold`, which resulted in compact code that reflected the essentials of the algorithm well.

At this point, the software could run reliably against a dataset and calculate CPU usage and the percentage of total host CPU that the trace represented during its execution.

The next stage was to implement the *Energy Usage Manager* interface and its concrete implementation the *Energy Usage Manager Simulator* class, which provides estimates of energy consumption for a specified machine during a specified period. This implementation uses a model-based approach, utilising SPEC Power benchmark energy values for the machine type of the host. The class queries InfluxDB to find the host CPU utilisation during the specified time period for the specified host and then uses a lookup table to find

an estimate of the power consumption for the host type of the machine at the average level of utilisation during the period.

This completed the implementation of the calculation module and its automated tests, allowing us to move on to the validation stage and use it to perform energy allocation calculations for real test cases for a sample application, which we describe in Chapter 8.

7.8 Limitations of the Apollo Energy Estimator

Our implementation of the Energy Estimator is a practical proof of concept version of our approach. We have used it to run extensive tests to validate its implementation and our general approach for energy usage allocation of an application's request processing. There are however a number of known limitations in the current implementation, some of which are fundamental to our approach and some of which are the result of simplifying decisions to make the proof-of-concept implementation tractable, and which could easily be addressed through further work.

Firstly the estimator does not take *infrastructure energy consumption* into account as part of its calculation. A data centre's energy efficiency is normally measured through its *Power Usage Effectiveness - PUE* ratio [74], which measures the ratio between the energy used for IT equipment and the overall energy consumption of the data centre. The difference between the two is the data centre's infrastructure energy consumption due to items like lighting and cooling. For simplicity of initial implementation, the estimator does not take PUE into account and so ignores the infrastructure energy overhead. This would be reasonably straightforward to add, given a mapping from our execution hosts to data centre environments, and a source of data to define the PUE ratio at different times of day for those data centre environments.

As we noted at the start of this chapter, we have made a simplifying implementation decision to only monitor the *energy consumption of our application microservices that communicate using RPCs* (usually JSON over HTTP, although this is not, in fact, a limitation of the implementation - using Zipkin we can trace over other transports such as gRPC too). This assumption reduced the number of cases we need to consider in the proof-of-concept implementation and could be relaxed at a later date. Zipkin is capable

of tracing service invocations over other transports (like messaging) and we could extend the approach to allocate energy correctly to other elements of the architecture (like database servers dedicated to our microservices), by extending the sophistication of our implementation. What we can't easily estimate is energy consumed by "side effects" of the execution of our system, such as the energy consumed by another system as the result of us sending it a message. If the approach is implemented in all of the systems then all of the energy will be allocated somewhere, but in reality, this is unlikely ever to be the case, hence we exclude the energy impact of external side effects.

Next, the approach we use has a *reliance on dedicated microservices* ("runtime elements") for testing purposes. This is because we cannot separate the resource usage by a microservice for our traced requests from the overall resource usage of the microservice if it is used for other work during the period of the trace. This is a reasonably fundamental limitation of using resource utilisation metrics at the microservice level. On some operating systems (such as Linux) it is possible to obtain CPU utilisation metrics at thread level, which would allow more flexibility in this regard. However, it would require us to make quite a sophisticated extension to a tracing library like Zipkin or Jaeger to map requests to execution threads as well as network addresses. We did not attempt this during the proof-of-concept implementation but will consider it as a possible future direction for the work. In practice, as explained in Section 6.8.2, we do not think that this will be a significant barrier to the usefulness of the approach, given the ability of most microservice architectures to support test workload in production and the fact that this limitation is common with many types of performance-based testing.

Our proof-of-concept implementation is *batch based* and we use it by running tests in a specific environment, collecting all of the metrics we need and then loading these metrics into a separate environment and running Apollo against them. This is a fairly slow process and is better suited to offline analysis than interactive use in a test or production environment, where immediate feedback would be more valuable. However, the calculation process does not require the databases to contain only the data for the current calculation, as it extracts what it needs from the database, rather than trying to process everything that is there. Hence, it is perfectly possible to use the calculator in a "mini-batch" mode, where it is triggered for a specific trace ID once the trace's records have

been written to the tracing database. In this way, the calculator could be used to produce a stream of energy allocation results, almost as soon as traces of interest complete. While we have not done the scripting required to use the calculator in this way, we do not view this as a significant limitation as it is a relatively straightforward mode of operation to implement.

As was discussed in Section 7.7.3, the use of sample-based resource utilisation metrics means that we rely on *interpolation between the sample points to estimate resource utilisation* at the arbitrary start and end points of execution traces. This is an inherent limitation of using sample-based metrics and as explained in the previous section, we have an effective mitigation, by measuring repeated execution of test workload, to minimise the impact of the inevitable inaccuracy that creeps in through this process. This is a common problem in a lot of performance testing work given the common use of sample intervals for performance metrics and so we do not believe that it is likely to be a significant problem. However, it is inconvenient for the user to have to run longer test workloads than they would ideally use. We believe that a fruitful area of future work will be to investigate the practicality and tradeoffs of event-based sampling, such as the use of request interceptors to record the values of real-time operating system statistics to avoid this limitation.

Finally, the current implementation does not provide *visualisation or analytics* for the software architect to use to investigate the results and draw insights from them. While not the focus of this work, there are a number of commonly used open source tools (such as Grafana) that could provide a visual representation of the results and architects also commonly use analysis tools such as Microsoft Excel to perform this sort of analysis. This is clearly an area that future work could investigate to find out what types of visualisation and analysis would best support an architect's investigation into the energy characteristics of their application.

7.9 Conclusions

In this chapter, we have described how we implemented a practical and reliable proof-of-concept version of the Apollo Energy Allocator using a mix of open source software and a custom software calculator.

The specification was interpreted to identify concrete pieces of software to perform most of the tasks required (such as application packaging and data collection and storage) and this software was configured to integrate it into a working system.

The custom software module reads data from the data stores (relational database, NoSQL database and JSON file) that describes the application workload processed during a period of interest, the application elements involved in processing it, and the resource usage and energy consumption metrics of the application elements and the environment. It transforms and normalises this data to support the calculation process and then implements the energy allocation calculation specified in the previous chapter to allocate energy fairly to the application requests described in the trace data.

The primary third-party software used was Docker, Linux, MySQL, InfluxDB and Telegraf, while the custom software implementation was performed (mainly) in Kotlin, using key third-party libraries such as Klaxon and Konfig to simplify the implementation process as much as possible.

We performed extensive unit and integration testing during the configuration and development of the system in order to provide confidence in its correctness, reliability and usefulness as it was constructed.

With this implementation process complete, the software could now be applied to realistic problems, using a test microservices application, which we describe in the next chapter.

Chapter 8

Validation of Application Energy Monitoring

8.1 Introduction

As described in Chapter 7 we have implemented a proof-of-concept version of the Apollo energy allocation system, to prove the usefulness of our approach for allocating the energy consumed by underlying host systems to individual application requests executing on them. The software was tested during development to ensure correctness with respect to expected results through unit and integration tests but now needs to be validated by using it in realistic test cases. This process is described in this chapter.

In order to validate Apollo with realistic test cases, we need to define the kind of validation we are interested in achieving. There are four validation goals that we wish to achieve, as listed below.

Validation Goal 1 We need to validate the *calculation correctness* of Apollo's results, by running the calculator in one or more controlled scenarios where we can also gather additional runtime statistics that allow a separate independent calculation of a fair energy consumption and manually perform these calculations and use them to check the correctness of Apollo's results in the same scenarios.

Validation Goal 2 We also need to validate the *calculation consistency* of Apollo's results across a range of scenarios, to ensure that the same result is calculated consistently for equivalent but different workloads (e.g. 5 tasks of 1 CPU second workload produce the same result as 1 task containing 5 CPU seconds of workload, when all other factors remain constant).

Validation Goal 3 We need to validate the *energy allocation algorithm* when an application scenario is run on a host machine with different amounts of competing workload. Specifically, as competing workload on a machine rises and other factors are held constant, the energy allocation for an application scenario should fall, to reflect it's declining proportional usage of the machine.

Validation Goal 4 We want to validate *CPU usage as a proxy for resource usage* when performing energy allocation calculations. For this validation, we focus on how CPU usage varies for disk IO intensive workloads.

Throughout this validation testing, we aim to achieve consistency of results to within 5% tolerance. Long experience has taught us that a high degree of consistency is very difficult to achieve in any performance or resource utilisation testing due to the number of factors that can affect the results of a test on a modern multi-cpu, multi-core server machine. Our experience in previous testing work is that a 5% tolerance in most cases is the lowest degree of variation we are likely to achieve, being equivalent to equality plus the variation caused by factors outside our control.

8.2 Testing Approach

8.2.1 The Test Application

To allow us to test Apollo, we needed to reliably and reproducibly generate application workload that we could monitor, collect data for and run Apollo on the results. We briefly considered using a real application such as an existing enterprise application or an open source business domain application. However, it quickly became clear that using a real application would make the validation of the Apollo calculator almost impossible as it

would be very difficult to make the application workload highly predictable and repeatable, to allow validation of the calculations performed.

Therefore, we decided to create a simple application, specifically designed to allow predictable application workloads to be generated and reproduced on demand. The application contains four microservices implemented in Java:

- A *Gateway Service* that provides a simple entry point to the application for a benchmarking client to call and acts as an API gateway [10] for the application. The Gateway Service contains configuration to define repeatable application scenarios that can be invoked by name and this service then calls the other microservices as required for a particular scenario.
- A *CPU Hog Service* that will consume a specified amount of CPU time, specified by a URL parameter to its service call. The service can consume CPU time constantly for a timed period (e.g. 5 seconds) or it can consume a specified number of milliseconds of CPU time before returning. It times itself for a timed period using the system clock (via the `java.lang.System.currentTimeMillis()` method) and measures its CPU time using the Java JMX monitoring facilities.
- A *Memory Hog Service* that will consume a specified number of megabytes of (heap) memory, then wait for a specified number of seconds before attempting to release the memory. The amount of memory and holding period are specified as URL parameters to its service call. Given the design of the Java Virtual Machine (JVM) care must be taken when using a program to consume and release memory reliably, as the combination of JVM garbage collection and virtualisation of memory access means that the operating system process often does not increase or decrease its memory usage predictably in response to Java code increasing or decreasing memory usage. The service is useful though for applying increased memory pressure on the machine at specific times.
- A *File IO Hog Service* that will perform a specified amount of file IO when its service entry point is called. The amount of IO to perform is specified in megabytes as a URL parameter. This service works in a very specific way, to avoid possible IO system optimisations reducing the real IO performed, while also avoiding unnecessary

CPU consumption. The problem we could have is that if we repeatedly write blocks of predictable data (e.g. blocks containing the same value throughout or repeating short data sequences repeatedly) then there is a danger that the IO subsystem will avoid performing some of the IO operations by performing some sort of write compression. On the other hand, if we generate a large number of blocks of truly random data, this will involve quite a lot of CPU usage, which distorts the operation of the service and could make it CPU rather than IO intensive. Therefore, the service generates a set of resusable random blocks of data when it starts up and selects a random sequence of these blocks, using a cheap pseudo-random number generator (`java.util.Random`) when writing a file. This minimises CPU usage during the writing of the file, while also ensuring enough randomness to avoid obvious IO system optimisations.

These services were implemented using Java 1.8 and version 1.5.7 of the widely-used Spring Boot application framework (to provide application services such as HTTP request handling, configuration, database access abstraction and dependency injection). This minimised the amount of application code that had to be written, allowing us to focus on the code needed to implement the core purpose of each service. The microservices were organised into separate source trees, built using the Gradle build utility.

Once developed, and unit and integration tested, the services were packaged into Docker containers to allow easy versioning, deployment and monitoring via the Docker runtime system, which was required by our proof-of-concept version of Apollo.

8.2.2 The Test Software

In order to reliably run a large number of validation tests for Apollo, the process needed to be automated so that the execution of the tests and collection of the results was standardised, efficient and reliable. This involved a number of different types of automation and tools.

We started with a Linux server machine as the test environment. The specification of the machine and the specific version of Linux are not very important, but we selected a 4 CPU machine with Intel Xeon 2.3GHz CPUs, 16GB of memory and 50GB of SSD storage

running Ubuntu 16.4.04, one of the Ubuntu stable, long-term support releases. This is a reasonably large machine, but we selected this specification as it is representative of the sort of mid-range server class machine that forms the backbone of the server fleet of many large organisations today. Microservice systems often utilise many smaller machines, but we quickly realised that while the Apollo system runs equally well on many smaller machines (due to the features of the open source tools it uses) consistency, repeatability and control were going to be reduced when using a number of hosts. Therefore it was decided to do most of the testing using a single host machine that had sufficient capacity to run all of our services without resource contention.

The *machine configuration* was automated using the open source Ansible system configuration tool, with a number of custom configuration files (or "playbooks" as they are known in the Ansible ecosystem) and some custom shell scrips defining the basic system configuration needed for the test environment, including updating the operating system and installing security patches, creating users and installing public keys, installing basic tools like Python and Git, installing Docker, and setting up some basic security mechanisms.

Once the machine configuration was complete, *application configuration* and *service configuration* were both automated using the Docker Compose tool, which is part of the Docker toolset. Docker Compose allows a group of cooperating services to be defined in terms of the Docker container images they utilise, the specific configuration that each service applies to the base image and the dependencies between the services (such as one service needing to call another). We defined a single Docker Compose configuration that included the application services and the data collection services we needed to provide the data for Apollo (Zipkin, MySQL, Telegraf and InfluxDB as discussed in Section 7.4). This approach allowed a simple operating system script to be used to call Docker Compose with this configuration to reliably start or stop all of the services via a single command.

The process of *running tests and collecting outputs* was automated using a range of operating system utilities and custom shell scripts. The process can run one or more test scenarios, which were defined in the configuration of the Gateway Service as explained in the previous section. The process of running a test scenario involved:

- Clearing the MySQL and InfluxDB metrics databases so that the results of the test could easily be exported to files.
- Starting the `mpstat` and `pidstat` operating system utilities as "background" processes, logging their outputs to files, so that operating system and service process resource utilisation could be analysed after the test had executed.
- Invoking the Gateway Service to run the requested scenario (and in turn, it would call the other services as the scenario definition defined).
- Stopping the operating system utilities cleanly to terminate resource utilisation statistics collection at a defined point.
- Exporting the execution traces, host and application resource utilisation metrics and Docker network metadata from the MySQL and InfluxDB databases and Docker, into files that could be used to re-populate databases at a later point in time for Apollo to use.
- Exporting the host level resource utilisation statistics from the `sar` utility into a text file to allow host level resource utilisation to be analysed after the test had executed.
- Gathering all of the outputs of the test into a `tar` archive file for easy storage and transfer.

When a particular test scenario needed competing workload on the host machine, the `OpenSSL` package's `speed` command was used to generate load on one or more of the host's CPUs. This was a scripted process, invoked manually before test cases that required predictable competing workload on the server host.

Once test cases were available, the output archive data files were transferred to local machines and a scripted process to *run Apollo on the test scenario output* was executed. This cleared the local MySQL and InfluxDB databases, loaded the trace and resource utilisation data from the scenario into these databases, unpacked the other data files from the scenario archive, and ran Apollo on these data sets. The output of the Apollo calculator was then inspected to analyse the characteristics of the scenario and the energy allocation that it had been allotted.

8.3 Validation Goal 1 - Validating the Allocation Calculation

Our first validation goal is to ensure that the allocation calculation performed by Apollo can be reproduced by hand, following the algorithm presented in the previous chapter, with an acceptable level of consistency between the two approaches.

We chose one of the simpler standard scenarios that we had used during the testing process, the `simple-cpu-x50` scenario, which repeatedly makes short calls to the CPU intensive service. We decided to use a scenario based on just two services, the Gateway service and the CPU intensive service, both running on a single machine. This was to keep the manual calculation process tractable and avoid inconsistencies and mistakes complicating the process. The Apollo calculation algorithm is simply a process of aggregating results from individual application elements and so we were confident that if the result of calculation for a small number of application services was correct then this would result in a correct calculation process for more complex cases that were simply aggregations of the lower level calculations.

To perform the calculation process, we decided to use a different set of metrics data sources from the set used by Apollo, to validate the approach to data collection as well as the implementation of the algorithm. The set of data that the calculations were based on was:

- the *Zipkin traces* from the scenario being studied, as there was no credible alternative to this data;
- the *Docker network configuration* for the Docker subsystem that the application elements were running within, to allow the IP address to container mapping to be retrieved, and again this was the only source of this information;
- a set of `pidstat(1)` metrics for the application, showing the CPU utilisation every second for the application processes;
- a set of `mpstat(1)` metrics for the host machine which allowed us to assess how much CPU was being used by the host during the duration of the test trace;
- the *SPECPower benchmark results* for a representative server model, similar to the host in use; and

- the output of the `ps(1)` command from the host machine with all of the application elements running, to allow the process IDs for the application elements to be retrieved.

The important point about the data sources we used for this calculation is that the key data sources for resource utilisation metrics were different to those used by Apollo, to allow a broader degree of validation than a simple recalculation using identical data would have allowed. While Apollo gets its resource utilisation metrics from the Docker subsystem's statistics mechanism (gathered via the Telegraf metrics server), for our calculations we used the metrics from mpstat(1) and pidstat(1) as explained above.

The first step was to inspect the Zipkin trace data to allow the start and end points of the request to be identified. The Zipkin trace data had been saved from the execution of the test scenario and was loaded into a MySQL database to allow it to be queried easily.

The root traces were extracted from the Zipkin database, by selecting all of those rows in `zipkin_spans` where the `trace_id` column was the same as the `span_id` column:

```
SELECt * FROM zipkin_spans WHERE id=trace_id
```

From the row returned, we extracted a trace ID of `5896569591426873811`, a start time of `1534185668229000` (nano seconds since the "Epoch", equating to 20180813T184108.229Z), and a duration of `126292740` (nano seconds - 126.292 seconds) which imply an end time of `1534185794521740` (being equivalent to 20180813T184314.522Z).

Using the trace ID we could then run a more complex query to extract the trace's spans and their attributes from the Zipkin database:

```
SELECT hex(s.trace_id) as trace_id, hex(s.id) as span_id,
       hex(s.parent_id) as parent_id,
       start_ts as start_time_usec,
       start_ts+duration as end_time_usec,
       inet_ntoa(endpoint_ipv4 & conv("ffffffff", 16, 10)) as ipv4_address,
       endpoint_port
FROM zipkin_spans s, zipkin_annotations a
WHERE s.trace_id = 5896569591426873811
```

```
AND s.trace_id = a.trace_id
AND s.id = a.span_id
AND a_key = 'sr'
ORDER BY start_ts
```

This allowed us to identify the spans making up the processing of the request and the IP addresses of the two application elements that handled the invocations as being `172.18.0.7` and `172.18.0.8`, which were the IP addresses of the Docker containers containing the microservices involved in handling the request.

The next step was a simple lookup of the IP addresses in the Docker network configuration which resulted in the container IDs of the two Docker containers that our application elements ran within. The short form IDs returned were `c42b3d3bf4b2` for the Gateway container and `b75d29577eb9` for the CPU intensive container.

Having the container IDs allowed us to reliably identify the Linux process IDs of the Java virtual machines running our application elements. By using the `ps` output we saved during the test execution, we could use the container IDs to find the Docker container daemon processes running the JVMs for our services, with CPU intensive service and the Gateway service.

Having the process IDs allowed us to inspect the `pidstat` output that we had collected during test execution. This utility captured CPU percentage usage statistics for our processes every second during the test period.

For the CPU intensive service, we found that its average CPU consumption was 25.42% with a very low standard deviation of 0.26 across the test set.

For the Gateway process, we found that its average CPU consumption was 0.3% of the machine (although this did have a maximum of 0.75% for short periods, leading to a standard deviation of 0.11 for the test set).

Therefore, we decided to use 25.75% as the container CPU usage during the test period.

The next step in the process was to estimate the energy usage of the underlying host machine. We collected `sar` and `mpstat` resource utilisation metrics for the machine during the test period to provide host CPU utilisation metrics. The `sar` statistics were of less use

as this utility collects statistics every minute and so we had a limited number of samples but it appeared to indicate a total machine CPU usage of about 25% during the test period.

The `mpstat` statistics were more useful as they contained a metric measurement for CPU usage percentage per CPU every second during the test period. When we analysed this dataset we found that the machine's utilisation was a very constant 27% right through the test period, apart from one second (10 seconds into the test) when it jumped to 75% usage and then back down to 27% (for reasons we weren't able to identify).

The CPU usage percentage allowed us to estimate the energy consumption of the machine using a representative set of SPEC Power benchmark results for a similar server model. The benchmark results were 102W at 20% and 120W at 30%. Therefore interpolation resulted in the value $((120 - 102) \times (7/10)) + 102 = 12.6 + 102 = 114.6W$. That is, the machine's power consumption during the test period was about 114 J/second.

Given that the test period was 126.30 seconds, this resulted in a power consumption value for the host machine of $126.30 \times 114 = 14473.98J$, which we rounded to 14474J.

Given the host CPU usage, the CPU usage of our containers and the host power consumption we could now calculate the power allocation to our application trace.

Given a test length of 126.3 seconds and a 4 CPU host machine this means a total possible CPU resource available of:

$$126.3 \times 4 = 505.2\ CPU seconds \tag{8.1}$$

Our test data indicates that the host was busy for 27% of the time during the test execution, therefore our total host CPU time consumed during the test period was:

$$505.2 \times 0.27 = 136.40\ CPU seconds \tag{8.2}$$

Our application processes (containers) consumed 25.75% of the host's CPU time during the test period therefore their CPU usage is calculated as:

TABLE 8.1: Manual Calculation Compared to Apollo Calculation

Value	Manual Calculation	Apollo Calculation	Difference
Trace CPU msec	130090	124257	4.5%
Host CPU msec	136404	132891	2.5%
Host Energy J	14474	14213	1.8%
Application Energy J	13804	13290	3.9%

$$136.40 \times 0.2575 = 130.09\ CPU seconds \tag{8.3}$$

Given our CPU usage and the host energy consumption we can then calculate the energy allocation to our application request as being:

$$130.09/136.40 \times 14474 = 13804J \tag{8.4}$$

We then ran the Apollo proof-of-concept implementation on the test scenario data set to compare our results, which are shown in Table 8.1

As can be seen from the results in the table, using a manual calculation technique that attempts to mirror the algorithm used in Apollo, while using different data sources for the critical resource utilisation metrics, has resulted in a very close match between the results, with the manual calculation of the energy allocation for the application request trace being within 4% of the automated Apollo value.

The difference in the results will be familiar to anyone who has been involved in performance testing, where repeatability of results is often very difficult to obtain (as evidenced by the subtle but constant inconsistencies in the numerical results from tools such as `sar`, `mpstat` and `pidstat`). In our situation, as well as the normal difficulty of repeatability, we believe that most of the difference in these results is likely to be the result of Apollo using the finer grained Docker metrics, while the manual process relied on the coarser grained metrics from the standard system administration tools.

This result is well within our target result tolerance and so validates the Apollo proof-of-concept implementation's accuracy and achieves validation goal 1.

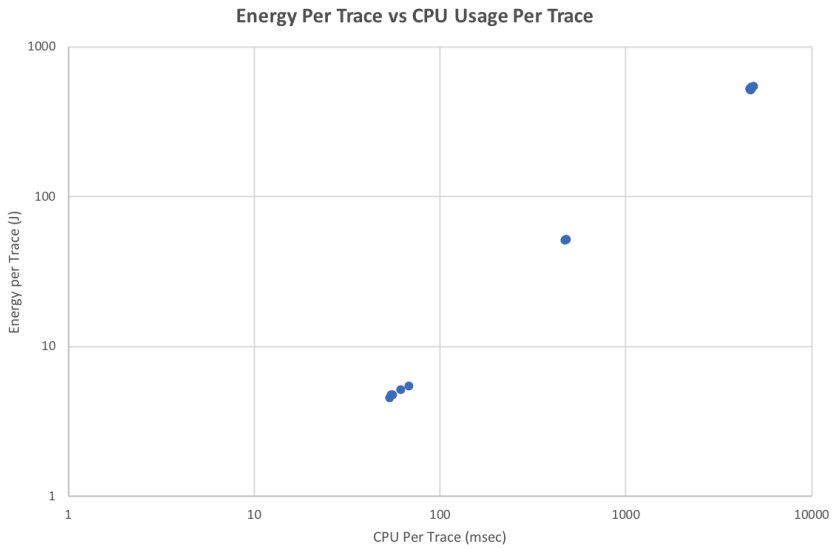


FIGURE 8.1: Energy Allocation per Request for Small, Medium and Large Services

8.4 Validation Goal 2 - Validating Allocation Consistency

Our second validation goal, is to validate consistency of energy allocation. To achieve this, our strategy is to run a known control workload under fixed host utilisation conditions (no other workload being the simplest case) and to run a range of other workloads that we know contain an equivalent amount of computational work but are structured differently. The energy allocation should be the same for each case.

In our first test, we structured a workload into three cases, each of which involved the same amount of CPU workload but in three different scenarios. The first scenario invoked a short service call (involving 50msec of CPU work) 1000 times, the second scenario invoked a longer service call (500msec of CPU work) 100 times and the third ran a long service call (5000msec of CPU work) 10 times. Each scenario was called 6 times to ensure a consistent result.

Our first question was whether the energy allocations per request were consistent across the three cases. Our analysis of this question is shown in the scatter graph in Figure 8.1, which plots the energy usage against CPU workload for each of the scenarios executed, using logarithmic scales.

The graph shows the three sets of scenarios (short, medium and long) clearly clustering very closely, showing that the energy allocation per trace is extremely consistent

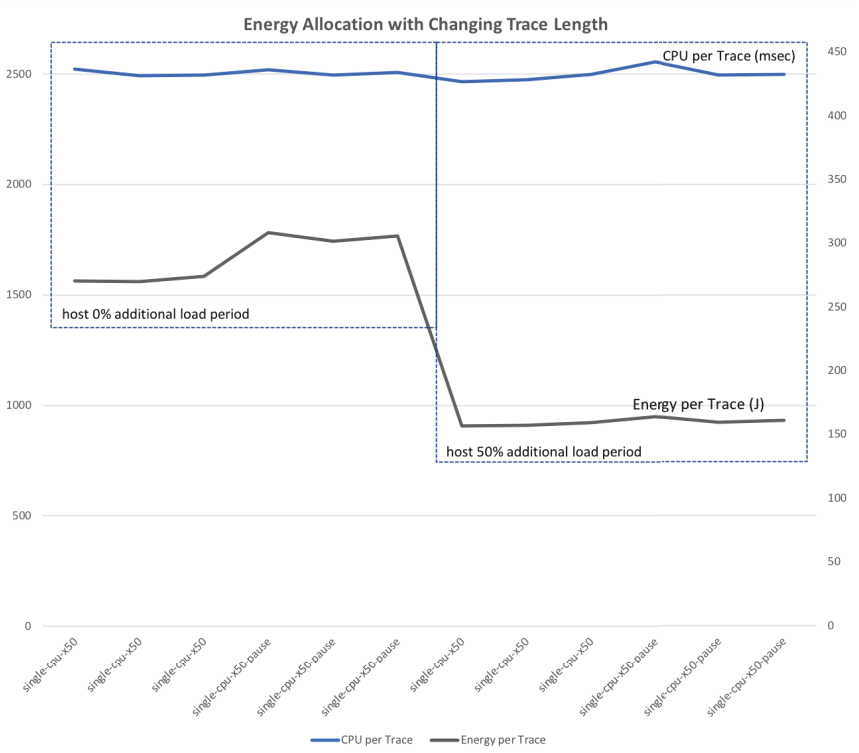


FIGURE 8.2: Energy Allocation Across Different Scenario Lengths

across the three sets, suggesting that the allocation calculation is working consistently as designed. More precisely, the correlation coefficient between the two sets of values is 0.999, indicating a very high degree of correlation between CPU consumed and energy allocation performed.

Our second test involved investigating how energy allocation was affected by a constant amount of CPU workload but in scenarios of different lengths. To test this we repeatedly ran two scenarios, both of which contained the same amount of CPU workload (a trace that took 2,500 msec, run 50 times), with one scenario having pauses inserted into it, to cause the scenario to take longer to execute but consume no more CPU resource during the extended execution time. In addition, for reasons which we explain below, we ran the first set of scenario tests with no additional load on the machine and the second set with synthetic workload consuming 50% of the machine's CPU. The results of this experiment are shown in the line graph in Figure 8.2

This graph plots the CPU usage per trace (the top line) and the energy usage per trace (the bottom line) for sample executions of the scenarios. As indicated by the dashed

boxes annotating the graph, the first 6 executions were performed with no additional workload on the machine, the second 6 executions were performed with the machine having 50% of its CPU capacity consumed by other synthetic workload.

As can be seen from the graph, the estimate of CPU usage per trace is consistent, within a maximum of 2% variation from the mean (min 2467, max 2556, mean 2502, with a standard deviation of 23.09, which is less than 1% of the mean). This is well within our target consistency.

When we analysed the power allocation by trace, we saw an interesting development which was the higher power allocation for the longer scenarios when no additional load was on the machine. In contrast, there was a more constant power allocation when 50% additional load was running on the machine. While unintuitive initially, when we investigated the data, as explained below, we found that this is exactly as expected and is an important energy usage insight for the software architect investigating the power characteristics of their software in production.

When no additional load is executing on the host, there is no other workload apart from our traces to allocate power consumption to. In which case if the scenario takes longer, you would expect a higher energy allocation even with constant resource usage, as the host machine is consuming energy, even when not actively running our workload and if there is no other workload to allocate this "background" energy consumption to, then it will be allocated to our workload. The graph shows that this is exactly what happens; when no other workload is executing on the machine, our longer scenarios (the "single-cpu-x50-pause" scenarios) are allocated more power than the shorter scenarios (the "single-cpu-x50" ones) even though they all consume very similar amounts of CPU time.

In contrast, when there is additional workload on the machine, the energy allocated to each trace is much more even, within a maximum of 4% variation from the mean (min 157, max 164, mean 160, with a standard deviation of 2.66 which is 1.6% of the mean). This is also an expected result as, during the execution of our test workload, there is other workload running on the machine which shares the allocation of the host's energy consumption. As our workload runs longer but is not using CPU during part of the period, it is allocated correspondingly less of the host's energy as there is other active workload on the machine which is allocated more of it. This is the allocation we would expect

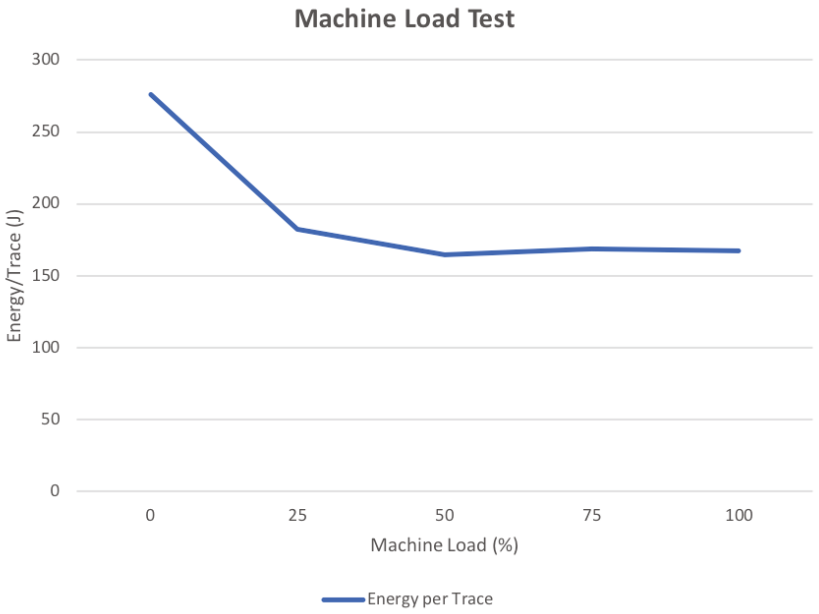


FIGURE 8.3: Energy Allocation Under Different Host Load Conditions

and it is within our target consistency and again suggests that this is a consistent energy allocation mechanism, so achieving validation goal 2.

8.5 Validation Goal 3 - Validating Allocation Scenarios

Validation goal 3 involves validating that a fixed workload is allocated energy consistently when the underlying host has varying amounts of other workload running on it concurrently. We tested this aspect of allocation by running a single workload type under 5 different host utilisation conditions, namely when there was no other workload on the host, and when the host was 25%, 50%, 75% and 100% utilised before our workload started. The results of this test are shown in the line graph in Figure 8.3.

As with some of our other testing, the results initially look somewhat counter-intuitive, but on further analysis are validation of consistent energy allocation by workload.

The first point in the graph shows our workload running on an otherwise idle host and it is allocated quite a large amount of energy per trace (of 276J) because the host is relatively inefficient at lower levels of utilisation and there is no other workload on the host to allocate its energy to.

The second point in the graph shows a sharp reduction in energy per trace (to 183J), which is caused by the host's utilisation, which is now about 50%, which is considerably more efficient than 25% and the fact that the host's energy is being split between two roughly equivalent workloads.

The third point on the graph shows a further reduction in energy per trace (to 164J), which is a considerably smaller reduction than the previous step. This is due to our share of the machine workload falling less significantly than in the previous step (from 49% to 33% whereas the previous step was from 96% to 49%).

The fourth point on the graph, at 75% of other utilisation, actually goes up slightly (to 169J). This is due to two factors. Firstly, once again, our utilisation percentage drop decreases, this time from 33% to 25% (only 8%) but secondly, the underlying machine becomes less efficient as utilisation moves beyond 75% and so there is more energy to allocate between the different workload items. This is an important insight for the application architect so that they consider the potentially non-linear energy consumption curve of the underlying host.

Finally at the fifth point on the graph, with 100% other utilisation, our workload is competing with the existing workload to be scheduled for execution. This results in our execution duration extending slightly (by 5%), and our CPU utilisation percentage to drop slightly (to 23%) with the result being a slight reduction in energy utilisation (to 167J). This is the result of a relatively small increase in the host's energy utilisation (as it was already running close to 100% utilisation at the previous sample) and there is now further workload to allocate the energy of the host across, so reducing our workload's allocation slightly.

This phase of testing was an interesting process because it illustrated the usefulness of investigating energy allocation using practical testing and a quantitative data-based allocation mechanism like Apollo. It would be quite possible to make a simplistic assumption that fair energy allocation would keep falling linearly as load increased, but our tool can be used to provide a more sophisticated analysis that reveals how a complex interaction of a number of factors (including load, scenario length and host energy characteristics) can result in a correct allocation that is more complex. This is a useful insight for the application architect as they investigate the energy characteristics of their application and this process allowed us to achieve validation goal 3.

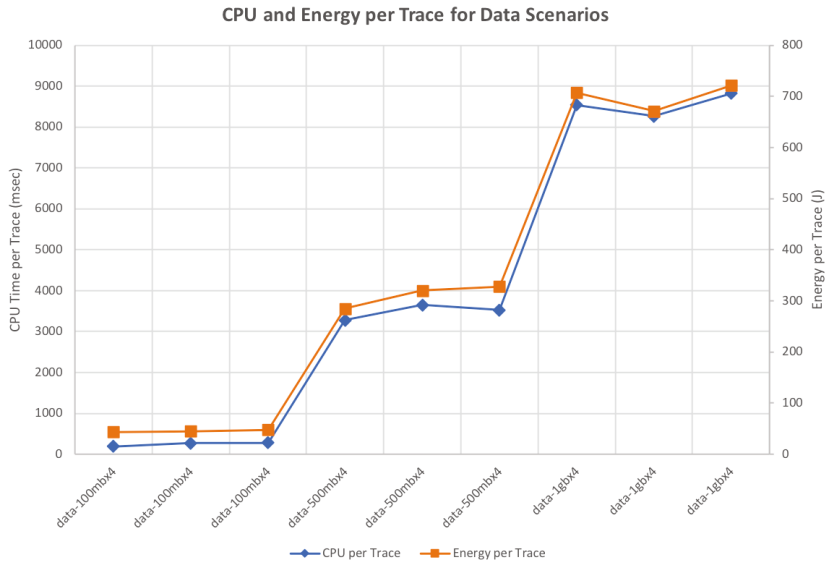


FIGURE 8.4: CPU Utilisation and Energy Allocation Scenarios

8.6 Validation Goal 4 - Validating CPU as a Resource Usage Proxy

Validation goal V4 involves ensuring that CPU usage is a good proxy for overall resource usage of a piece of application software.

When we explained how the energy allocation process for an application's elements was to work, in Chapter 6, part of the design of the allocation approach was to make the simplifying assumption that CPU utilisation is a good proxy for overall resource consumption (Section 6.7). This allowed the approach to rely on CPU usage to allocate energy fairly. While this assumption is based on previous research work [15], we were interested to test this assumption for ourselves by comparing IO activity with CPU utilisation.

To test the assumption that CPU utilisation acts as a good proxy for IO activity, we wanted to find whether the two values correlated well during an application workload. To test this we ran IO intensive workloads of varying known sizes and measured the CPU utilisation of each one. The results of this exercise are shown in the line graph in Figure 8.4. The x-axis of this graph shows the test scenarios, the left-hand y-axis is the amount of CPU measured for each scenario, the right-hand y-axis is the amount of energy allocated to each scenario.

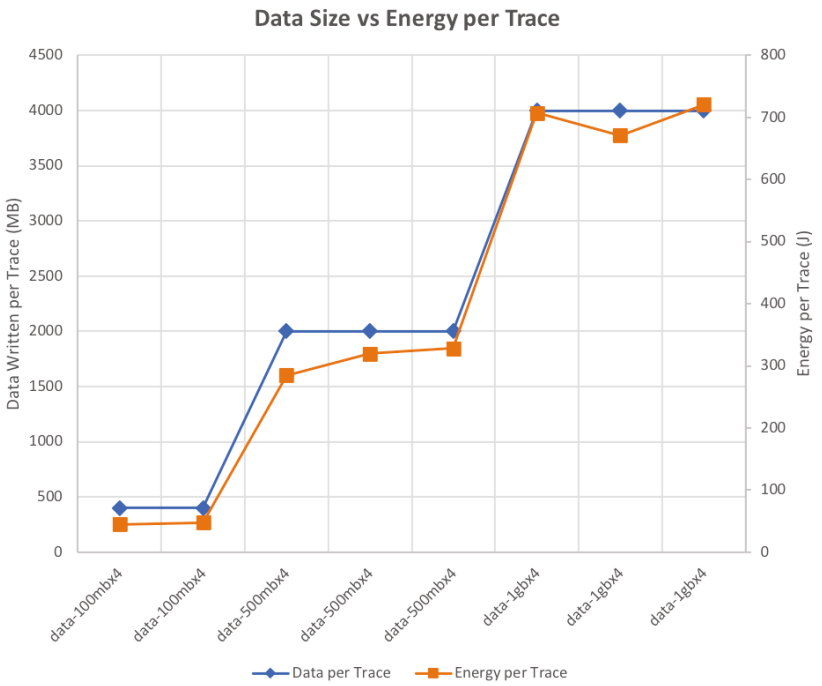


FIGURE 8.5: Energy Allocation by Data Size

The graph shows the results of running a number of data-intensive application scenarios, one group that called a service to write 100MB of data to file 4 times, one that called a service to write 500MB of data to file 4 times, and one that called a service to write 1GB of data to file 4 times. Hence the total data written by the first group of scenarios was 400MB, the second group 2GB and the third 4GB.

When we plot the CPU usage and the Apollo energy allocation for the other test scenarios on the graph, we can clearly see that CPU usage is directly related to the amount of data written and the correlation coefficient between the values for data written and CPU utilised is 0.9965, indicating that CPU is a good proxy for the amount of data written by a process.

For completeness, we also plotted the data-size of each scenario compared to the energy allocated by Apollo to each, which is shown in Figure 8.5.

This graph looks similar to Figure 8.4 but is illustrating a slightly different point in that while the x-axis (test scenarios) and right-hand y-axis (energy per trace) are the same, the left-hand y-axis shows the amount of data written by each scenario, rather than the CPU usage. This can be seen in the flat horizontal sections of the graph for each type of scenario.

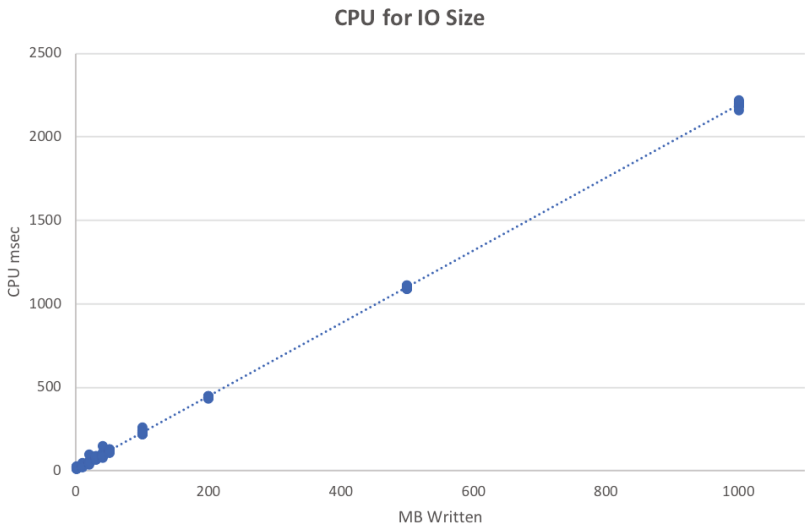


FIGURE 8.6: Energy Allocation by Data Size

The strong grouping of points around scenarios shows that this graph validates that the energy allocation for these data-intensive services correlates well with the amount of data each is writing, albeit with some variation in some test cases. When we calculated the correlation coefficient between energy allocation and data written, it was found to be 0.9967, again showing a very high degree of correlation between energy allocation and the amount of data written by the application elements within the scenario.

Finally, to provide some additional validation of the relationship of IO workload to CPU usage independent of Apollo, we performed some detailed manual tests, calling the microservice directly and measuring CPU usage before and after service invocations for different amounts of IO. The CPU usage was read directly from the `/proc/PID/stat` operating system statistics.

The result of these manual tests is shown in the scatter plot and trend line shown in Figure 8.6. The y-axis shows the amount of CPU used by the service per request, the x-axis the amount of data written by each request and each point on the graph is a single service invocation. As the graph shows there is a strong correlation between the amount of data written and the CPU consumed for the service call and when we calculated the correlation coefficient, it was found to be 0.9998, confirming this finding. By using the difference in data size and CPU utilisation between samples, we were also able to calculate that 1 MB of file IO write activity appears to require approximately 2ms of CPU time on our test machine.

The combination of these test results shows that we were correct in our assumption that CPU is a good proxy for other resource utilisation by a process and achieves validation goal 4.

8.7 Conclusions

In order to validate our proof-of-concept implementation of the Apollo Energy Allocator system, we identified four validation goals that we needed to achieve.

- *Validation Goal 1 - Calculation Correctness* which involves running the calculator in a realistic, but reasonably simple, scenario, under controlled conditions and replicating its calculation process as independently as possible.
- *Validation Goal 2 - Calculation Consistency* which involves running a number of test scenarios with different characteristics and comparing the energy allocation results provided by Apollo, to ensure consistency between different types of scenario.
- *Validation Goal 3 - Energy Allocation Algorithm* which requires us to run identical scenarios in situations with different amounts of controlled competing workload on the host machine(s) to allow us to validate that Apollo allocates energy fairly across these workload profiles.
- *Validation Goal 4 - CPU Usage as a Proxy for Resource Usage* which confirms that the previous research result that suggested that CPU usage was a valid proxy measure for overall resource utilisation, and in particular for disk IO activity.

As stated earlier, our aim was to achieve consistency of results to within 5% tolerance as practical experience has taught us that results within this tolerance level are effectively equal in performance and resource usage testing, given the number of dynamic factors outside our control.

The testing presented in this chapter has specifically addressed each of these aspects of validation.

We performed a manual data collection and calculation exercise in order to validate goal 1, *Calculation Correctness*, as described in Section 8.3. The result of this exercise, based on independent resource utilisation data sources, separate from the data used by Apollo, was an energy allocation value within 4% of the value calculated by Apollo.

We then investigated goal 2, *Calculation Consistency*, by performing a series of CPU intensive workload tests, as described in Section 8.4, that ran controlled application workloads for different levels of resource consumption, in order to confirm that Apollo allocated energy correctly and consistently for all of the test cases. These test cases proved that there was a very high degree of correlation between our different levels of application workload and the energy allocation that each received, so proving that the energy allocation was consistent across varying workload.

The next step was to validate goal 3, *Energy Allocation Algorithm*, as described in Section 8.5, by running controlled test workloads on host machines that had carefully controlled competing workload running on them already. These tests proved that when competing workload was present on a host machine, energy was allocated correctly between our test workload and the competing workload and was well within the 5% level of consistency that we were aiming for.

Finally, we investigated whether validation goal 4, *CPU Usage as a Proxy for Resource Usage*, was correct for an application trace. We were aware of previous research results suggesting that this was the case, but we decided to investigate it empirically in our specific situation too. We focused on file IO during this part of the investigation and ran a number of tests to investigate its relationship to CPU usage. These tests confirmed that CPU usage is a very good proxy for file IO usage, with a correlation coefficient greater than 0.9 for the two values.

In summary, we have investigated four different aspects of the correctness and utility of the Apollo Energy Allocation System's proof-of-concept implementation of our energy allocation approach for application-level energy monitoring. All four of the testing exercises confirmed a different aspect of the correctness of the implementation. Therefore, we conclude that the proof-of-concept implementation is sufficiently consistent and correct to validate the application energy allocation approach that we propose.

Chapter 9

Conclusions and Future Directions

9.1 Summary and Conclusions

The research presented in this thesis has been a journey from abstract design languages to practical runtime tools with the goal of providing software architecture practitioners with better support for considering energy as an architectural concern than they have today. On the way, it has involved tools, design guidance and the working practices of effective architects.

This journey has resulted in a number of research contributions to the fields of software architecture research and energy efficiency research.

Firstly, we have performed a comprehensive systematic survey of 25 years of research in the field of architectural description languages, resulting in a thorough characterisation of the field. This then led to a published case study [180, 181] that reported the experience of creating a large-scale industrial architectural description and the shortcomings of existing architectural description languages in such environments.

The question of how architects can prioritise energy efficiency work led to an interview based investigation of how expert architects prioritise their effort and the development of a model that distils the common advice into an accessible form that can be used to guide less experienced practitioners [183]. This was then validated and refined through a large-scale survey of software architecture practitioners from across the world.

We considered what tangible advice was available to software architects who want to improve the energy characteristics of their systems and found little in the research literature that most architects could directly apply. This led us to identify a small number of architectural design principles [19] based on a successful industrial case study of a large organisation that improved the energy characteristics of some of their application services, through architectural changes.

Finally, we identified the need for a practical tool that architects could use to measure the energy characteristics of their applications when running different scenarios and designed an approach to achieve this. We implemented a working proof-of-concept version of the tool, validated it with practical test cases and made it available as open source software.

This research was undertaken in the context of the four research questions that we introduced in Chapter 1 and now, having completed the work, we can provide answers to them.

9.1.1 RQ1 - Use of Architectural Description Languages

RQ1 *What architecture description languages exist and can they be used to reason about the energy properties of a system?*

To answer this research question, we performed a thorough review of the research literature over the last 25 years, presented in Chapter 2, and considered whether the ADLs we found could be used in an industrial context. We performed a significant case study projectand created a large architectural description of an industrial system, which was then used for a variety of purposes, as described in Chapter 3. However the conclusion we reached during this work (see Section 3.9.3) was that existing architectural description languages are not suitable for mainstream adoption due to their narrow focus on functional structure, the lack of industrial validation, the high adoption cost of most of the languages, and the lack of mature tool support available. This led us to define a lightweight graphical notation, supported by graphical and textual templates for documentation creation, which was ultimately successful in the case study project.

Many of the ADLs we surveyed are extensible, although relatively few of them (4, 7% of those surveyed) provide direct support for capturing system qualities in the language,

we judged that half of the ADLs we analysed could capture system qualities via some mechanism provided by the language. Therefore, in principle, it should be possible to use these languages as the basis of a system to allow reasoning about a system's energy qualities. However, the practical adoption problems we encountered mean that we do not believe that they can be used to support reasoning about energy properties in practice.

We make a number of constructive suggestions for how to make architecture description languages a more practical proposition for practitioners in Section 3.11.

9.1.2 RQ2 - Prioritisation of Architectural Effort

RQ2 *How can architects prioritise energy efficiency as an architectural concern?*

When considering how our work might be used by practitioners we realised that the first challenge was how to persuade architects to prioritise the energy characteristics of their systems. Architects have a very wide range of concerns to address and often complain that it is difficult to know where to focus their attention. Anecdotally, this seems to be particularly acute for less experienced practitioners.

We noted that many experienced practitioners manage to focus their effort very effectively and seem to be able to deal with a wide range of concerns during the lifetime of a project. When we asked people informally, we did not find anyone using a formal approach for this, it just seemed to be something they knew how to do.

An initial literature review did not find any generally applicable approaches that provided sufficient guidance to make a focus on energy properties likely, so we investigated how experienced software architecture practitioners focus their attention, as reported in Chapter 4. We found that some strong themes emerged, which we used to create a preliminary model to guide less experienced practitioners (see Section 4.4.2). This model was validated via an online survey questionnaire, completed by over 80 practitioners from all over the world and reported in Section 4.5. The results of the survey validated the model strongly and also provided input to allow it to be refined into a more effective model, which is described in Section 4.6.

We found that there are four aspects to the approach that experienced practitioners use to focus their attention, namely:

1. Stakeholder needs and priorities
2. Prioritising time according to risks
3. Delegating as much work as possible
4. Ensuring team effectiveness

As detailed in Section 4.8 this work provided an answer to our research question. From these four areas, the aspect of prioritisation that will cause architects to focus on the energy qualities of their systems is to ensure that the energy efficiency of the system is high in the list of stakeholder needs and priorities.

9.1.3 RQ3 - Design Guidelines for Energy Efficiency

RQ3 *What design guidelines can we provide to assist architects to improve the energy efficiency of their systems?*

When we considered what software architecture practitioners needed to allow them to confidently address the energy properties of their systems, we quickly identified the importance of accessible and reliable technical guidance. The two forms of guidance that architecture practitioners are already familiar with are architecture principles and architectural tactics, so we investigated the principles and tactics available to them.

The initial literature review (Section 2.3) revealed that while this field is relatively immature, there was material in the research literature that could be of use to architecture practitioners, notably an architectural perspective for energy efficiency. However, we found a lack of generally applicable tactics and principles. There were several sets of architectural tactics in the literature [99, 143] but one is aimed more at those building cloud platforms than applications and the other is specifically aimed at architects building applications utilising cyber-foraging to offload work from mobile devices.

In response, we decided to try to identify some architectural principles that could be generally applied by software architecture practitioners who were trying to improve the

energy efficiency of their applications. As described in Chapter 5, this was achieved by studying a published industrial case study of a large organisation who improved the energy efficiency of a number of their application services through software architecture changes, and extracting and generalising the principles that had guided their work.

This resulted in a set of three initial principles, introduced in Section 5.5, which had proved of value in the organisation that performed the case study. The principles we identified were:

1. Energy efficiency metrics must relate business transactions to energy consumption in a way that is meaningful to key system stakeholders.
2. Identifying sources of energy waste at the system level produces the biggest savings.
3. Addressing the energy optimisation problem requires a cross-disciplinary team.

This work allowed us to answer research question RQ3 with this initial set of energy-related architecture principles which we believe can be extended further in the future through the study of other successful industrial energy efficiency improvement projects.

9.1.4 RQ4 - Architect Awareness of System Energy Characteristics

RQ4 *How can we make architects aware of the runtime energy characteristics of their systems?*

Having considered how to enable software architects to focus attention on the energy properties of their system and identified some initial principles that could guide the development of more energy efficient systems, it became clear that architects also need to be able to measure the energy properties of their systems.

A literature review (see Section 2.4) revealed that there have been a number of attempts to create software systems that can measure the energy characteristics of software applications. However, as we reported in Section 2.4.3 there were a number of limitations with most of the work that had been reported in these publications.

Firstly a number of the projects used linear regression models to establish the relationship between resource consumption and energy consumption but had not validated how robust or reusable these models would be without constant re-training. Training these models in an industrial setting is complicated to achieve and models that require retraining for different workloads or after every change to the environment would not be a practical proposition.

The other concern with the existing research is the focus on measuring the energy consumption of operating system processes (or individual pieces of code) rather than execution scenarios. This means that the architect needs to set up very specific benchmark scenarios under controlled conditions in order to gain any insight from the results, which again is difficult to do in a real project. Instead, we wanted a scenario-based approach that measured the energy consumption of a single execution scenario, as this would allow the approach to be used with synthetic workload in existing test or production environments.

Finally, we also found that most of the research projects have not made their prototype systems available for inspection or use, meaning that many of the details of the work are unclear and there is no scope for reuse by other researchers.[1]

In order to progress this area of research, we designed a model (presented in Chapter 6) for estimating the energy characteristics of individual architectural scenarios (inbound requests) to a microservice based system. We dubbed our approach "Apollo" and implemented a proof-of-concept version of it (presented in Chapter 7) and then validated this with practical testing (reported in Chapter 8). The result is a reliable and practical tool for calculating the resource utilisation of a specific inbound request to a microservice system and using this to allocate the energy consumption of the server machines to the workload running on them. This encourages the architect to minimise the resource utilisation of their software and also to consider the most efficient deployment options for it.

This work provides us with an answer to our research question, discussed in Section 8.7, which is that we can provide architects with tools that calculate a context-specific energy consumption estimate for their software application executing different scenarios. This

[1] The E-Surgeon researchers [131] are a notable exception as they have helpfully open sourced all of their tools, which we investigated for insight and inspiration.

will allow architecture practitioners to use the tool with synthetic workload in suitably configured production and test environments, alongside other workload. This will allow them to monitor the application energy consumption over time and understand the energy implications of their architectural design decisions, so allowing energy to be treated as a first-class architectural concern.

9.2 Future Directions

Much of the work reported in this thesis has promising future directions to further increase the scope, applicability or sophistication of the research results reported here.

9.2.1 Architectural Description

An interesting observation during our industrial project to create a large architectural description was how easy or difficult people found the process of creating models of their software. As we reported, many software engineers appeared to find it very difficult to create effective models of their software, even when they understood it very well. In contrast, some others found it very straightforward and produced useful models with little or no guidance. When we did some initial investigation we could find no common factors that suggested how people separated into the two groups. We speculate that it could be related to people's learning styles (and in particular how easily they find abstract versus concrete thinking). However, this is just speculation and so an interesting further research direction could be to investigate this observation further.

Another area of possible research would be to investigate the applicability of the ADL we created to other situations. We did not create an ADL with a view to reuse and so some elements are quite specific to the environment that it was created for. However many of the key abstractions, such as message-driven servers, different sorts of user interface, messaging and RPC based connectors are common in many environments. Hence a possible future area of work is to attempt to apply the ADL in other environments to establish if it could be reused without major changes. In order to provide tool support

for the ADL, another related area of investigation would be to apply existing reusable ADL tooling such as that created for ACME [60] or ByADL [43].

During our research we also identified the possibility of using an ADL in conjunction with the Apollo energy estimator or a similar tool. While, as already outlined, we have some reservations about many existing ADLs, we do recognise the possibilities that the added formality of a machine readable ADL provides. The ADL could provide the estimation process with more accurate information about the structure of the system, while the estimation tool could feed energy estimates to an ADL-based model, so allowing simulation-based approaches to energy consumption analysis. This would allow such analysis at design time, before the software is available. If successful, then research in this direction might improve the abilities of the energy estimation tools and also augment an ADL to make it a useful design-time energy analysis tool.

While our work here has focused on energy estimation through runtime monitoring, as we noted earlier (see Section 2.4) other work (such as [65]) focuses on model-based energy estimation of architectural designs. An existing and established approach to architectural quality estimation is to use scenario-based analysis [12] which could extend existing work on model-based estimation. An area of future work, which would make energy estimation more accessible to many practitioners, would be to investigate how to apply scenario-based approaches to the estimation of energy efficiency as an architectural quality property. Such research could consider the sort of combination of ADL and energy measurement technology that we outline above.

9.2.2 Architectural Prioritisation

The refined model is now ready for dissemination to the practitioner community to see if it proves as useful in practice as our survey of the preliminary model suggests. To reach the practitioner community, we will publish the model in a less formal style via posts on mainstream Internet sites (such as medium.com, LinkedIn and Twitter). We will also try to publish a summary of it in practitioner-oriented publications and publicise it through conference sessions at practitioner conferences if it proves to be of interest to programme selection committees.

After practitioner-oriented publication, we are also interested in extending the Stage 3 questionnaire to architects in other geographical locations to compare and explore whether they react in the same way to the model as their colleagues in Europe and the Americas

Beyond this, it would be interesting to survey practitioners who have used the model in the future, after they have been using it for some time. This would allow us to understand whether its usefulness was borne out in practice and to find out what the practitioners are actually using it for (for example, whether it is used more as a training aid or as a personal aide memoir) and which industries and architectural job types are using it.

9.2.3 Architectural Design Guidance for Energy Efficiency

By analysing a successful industrial energy reduction project we have identified a small set of useful architectural design principles to guide architects in their consideration of energy as an architectural concern.

There is great potential in this area to identify other industrial work that is attempting to reduce the energy consumption of real software systems and from the successes and failures of those projects identify the principles and tactics that allow architects to actively manage the energy consumption of their applications as an architectural concern.

We also believe that further industrial and academic cooperation (of the sort we observed in a case study from the Netherlands [79]) could lead to the identification and validation of more principles and tactics for energy-aware architecture, as could focused academic work to propose likely principles and tactics and to validate them in both laboratory and industrial settings.

Once we have a larger proven set of principles and tactics then they would form a valuable addition to the energy architectural perspective created by Utrecht University and Centric Netherlands BV [78], which would make them available to architecture practitioners in an accessible form.

9.2.4 Runtime Application Energy Monitoring

The Apollo model and proof-of-concept implementation presented in this thesis is a research prototype that is still relatively immature, as its validation has been limited to controlled testing. There is significant scope to continue research in this area with the aim of creating a practical tool which can be applied in a general industrial setting.

There are a number of interesting avenues to explore in the area of data acquisition, including experimentation with hardware event counters and event-based direct data collection from operating system resource counters (rather than the sampling-based approach used in today's implementation). Extending the cost-based energy model beyond CPU usage to include network, disk and memory resource usage measurements would also be an interesting area to explore to see whether the increased accuracy is valuable enough to justify the additional complexity.

In a related area, the current implementation requires dedicated microservices for the monitored workload, to simplify the collection of statistics. This is a reasonable simplification because modern microservice infrastructure makes it straightforward to add additional container instances dedicated to a specific workload. However, an interesting future research direction would be to utilise thread-level resource consumption statistics rather than process level ones and investigate whether this would allow us to relax this constraint.

We could also fruitfully explore the extension of the current model to include a more diverse set of architectural element and connector types, such as message-based service invocation and the energy consumption of service processes such as databases.

The current model and implementation do not take the energy overhead of the data centre environment into account in the energy consumption estimates. This would be a relatively straightforward aspect to add to the model provided that a reliable source of PUE data for the environment(s) that the software is running in was available. PUE varies over time, but many DCIM products provide estimates of the PUE of an environment and so could be used as a source for this data. This would allow an allowance for the data centre's infrastructure to be added to the energy estimates, so highlighting the implications of deployment options to the architect.

The current system is batch based and as we described in Chapter 8 we tested it by running tests and collecting data sets from them, which were then processed by Apollo. However there is nothing in the model, or in fact the current implementation, that would prevent Apollo being used to analyse data in "mini batches" as soon as it is available. A potentially fruitful avenue of research would be to create an event-driven data collection system that generated a data set for Apollo whenever a scenario (i.e. an inbound request) completes, which can be observed from the Zipkin database. The data set could then immediately be processed by Apollo (which takes a couple of seconds), providing a near real-time view of the application's energy characteristics.

Another possible avenue for research is the outputs of the tool. The current software reports the energy and resource utilisation measurement values as text messages written to logs or the console. While perfectly functional, this means that the user needs to process this data themselves to analyse it (using text processing tools and spreadsheets, as we did during the validation process reported in Chapter 8). An interesting research topic would be to apply modern analytical and visualisation techniques and tools to the output of Apollo in order to provide the architect with insight into the results and perhaps automated guidance on how to improve the energy characteristics of the application.

Finally, the software needs to be made available as open source software via Github to allow other research groups and interested practitioners to access it. It has been developed "in the open" on Github but does not have the supporting materials to allow others to understand and use it at present.

9.3 Concluding Remarks

This research was motivated by the urgent need to reduce the ever-increasing amount of energy required to support the world's burgeoning digital transformation. This is needed for both environmental and cost reasons, to allow a sustainable transition to the next phase of the information age, particularly as developing countries become digital economies.

The journey has taken us from architectural description languages to the prioritisation of architecture work, through design guidance for energy-efficient applications, to the

creation of a novel model and tool to provide architects with insight into the runtime energy characteristics of their systems.

During the journey we have understood the state of the art in architectural description languages and tried to apply them, investigated how expert architects balance concerns to focus their attention to be most effective, identified architectural design principles for energy efficient systems and designed, built and validated a novel and practical tool for architects to use to measure the energy efficiency of their applications. The overall conclusion from the work is that it is now entirely possible to start treating energy efficiency as a first class architectural concern in software architecture, although a significant amount of work is needed to mature the field to the point where it can become part of mainstream practice.

As we progressed through the work we have answered our research questions, some positively and some negatively, but beyond those relatively narrow topics, we have been exposed to the huge amount of intellectual effort being expended across a fascinating range of topics related to the architectural design, analysis and energy efficiency of complex systems. Sadly much of this thinking, while creative and innovative, fails to have a significant impact due to a lack of validation, industrial alignment and accessible communication to practitioners.

Surely now, with the environmental imperative of controlling the energy usage of our digital economy, we can summon the motivation to realign our research and industrial communities in a united effort to address this problem? The world needs us to.

Appendix A

Architectural Description Languages

In this appendix, we list the characteristics of all of the architectural description languages that met our inclusion criteria for the literature review described in Section 2.1.

Due to the amount of information needed to characterise the ADLs, it is presented in three tables, Table A.1 that contains the basic characteristics of the languages, Table A.2 that describes the architectural concepts that appear in each language, and Table A.3, which lists the architectural description mechanisms provided by both.

TABLE A.1: General Characteristics of the ADLs

ADL	Description	Institutions	Year	Domain	Application	Reference
AADL	An architectural description language with industrial roots, having come from work in the avionics industry based on key concepts from MetaH and ACME. It is a rich ADL targeted at embedded systems, supporting a range of architectural views and having good tool support available. It has been standardised by the Society of Automotive Engineers (SAE).	Over 20 industrial and academic organisations	2006	Embedded Systems	Industrial Projects	[54]
ABC/ADL	Developed with the aim of improving the link to implementation from the architectural description and as part of this, providing good support for composition. It focuses on the functional structure of a system and prototype tools have been developed for it.	Peking University, Beijing, China	2002	General	Experiments	[115]
AC2-ADL	It is an aspect-oriented ADL that allows aspects on both components and connectors to be integrated into the architectural description.	Wuhan University Wuhan, China; University of California Irvine, USA	2008	General (AO)	Examples	[82]
ACDL	An ADL that represents the centralized-mode architectural connection in which all components are linked by a single connector. The connectors described in ACDL are structurally flexible in the sense that protocols implemented in them have no restriction on the numbers of attached same-type components.	University of Technology, Sydney; Tsinghua University, Beijing, China	2010	General	Examples	[164]
ACME	Developed primarily as a mechanism for the interchange of architectural information and as such is extremely flexible, intended as a base for more specific ADLs rather than being used directly.	Carnegie-Mellon University, USA	1997	General	Case Studies	[60]

TABLE A.1: General Characteristics of the ADLs

ADL	Description	Institutions	Year	Domain	Application	Reference
ADLARS	Developed with the aim of creating an ADL with first class support for embedded systems product lines. It supports multiple views of the system and variation points in the architecture.	Queens University Belfast, UK	2005	Embedded Systems, Software Product Lines	Experiments	[16]
ADML	The primary focus of ADML is dynamic behaviour rather than structure and it is based on a dynamic description logic called DDL(SHON(D)).	Four research groups in Beijing, China	2012	Concurrent and Distributed Systems	Examples	[176]
Aesop	Aesop was developed to allow the description of architectural styles, rather than just architectures. Aesop is a tool and the language that it implements, which allows style-specific architectural tools to be created.	Carnegie-Mellon University, USA	1994	General (Styles)	Experiments	[59]
ALI	Developed with the aim of producing an industrially relevant language, which would be usable for product lines as well as individual systems. It supports first class components, connectors and configuration but also includes explicit features for variation and reuse.	Queens University Belfast, UK	2008	General	Examples	[18]
AO-ADL	An aspect-oriented ADL is a descendent of DAOP-ADL. It allows aspects to be used as first class architectural constructs, in this case to assist in isolating parts of the system that address cross-cutting concerns (such as security).	University of Malaga, Spain	2011	General	Research Projects	[140]
Archface	A component and connector based language. It aims to bridge the gap between architectural description and code and does this by compiling the ADL into partially complete AspectJ code for the developer to complete.	Kyushu Institute of Technology and University of Tokyo, Japan	2010	General	Examples	[172]

TABLE A.1: General Characteristics of the ADLs

ADL	Description	Institutions	Year	Domain	Application	Reference
Aspectual-ACME	An extension to the ACME language discussed above. It adds aspect support to the language by extending ACME's connector element type.	Universities of Lancaster (UK), Bahia, Rio Grande do Norte and PCU of Rio de Janeiro (Brazil)	2006	Aspect-Oriented	Experiments	[58]
Backbone	An ADL developed with the concept of "resemblance" that provides a modelling construct that allows an inheritance-like concept to be applied to components at all levels, providing uniform reuse and evolution support.	Imperial College, UK	2006	General	Examples	[112]
Breeze/ADL	Breeze provides a means of describing a component and connector structure by means of an XML encoding of a graph formalism.	Shanghai Jiao Tong University, Shanghai, China	2013	General	Experiments	[101]]
byADL	byADL's name is a contraction of "Build Your ADL" because it was created based on the premise that it isn't possible to create an all-purpose ADL and so it is better to create an extensible base upon which domain specific ADLs can be created. It has formal underpinnings and semantics and uses model driven development (MDD) technology to automatically generate software to manage and transform ADL descriptions.	Universita dell'Aquila, Italy	2010	General	Case Studies	[154]
C2SADEL	It was created to simplify the definition of architectures following the Chiron-2 ("C2") style. The language is a more sophisticated and complete ADL for the style including software tool support for it.	University of California at Irvine, USA	1999	Concurrent distributed systems	Experiments	[113]

TABLE A.1: General Characteristics of the ADLs

ADL	Description	Institutions	Year	Domain	Application	Reference
C3	An architecture-centric approach that gives a new structure for connectors in which attachments are encapsulated within the definition of connectors. It defines and manipulates configurations as first classes entities. Also, a description of architectures from two different views, a model architecture view (logical architecture) created by the architect and an application architecture view (physical architecture instances of the logical architecture) generated automatically which serves as support to maintain the consistency and the evolution of the application architectures.	University of Nantes, France and University Center of Souk Ahras, Algeria	2009	General	Examples	[8]
CBabel	An ADL developed to support the description applications that are implemented using the concurrent CR-RIO framework. It was developed at Brazilian universities and is a formal ADL, focusing on correctness, with semantics defined in rewriting logic.	Universidades Federal Fluminense and Estado do Rio de Janeiro, Brazil	2005	Concurrent distributed systems	Experiments	[149]
CLARA	An ADL dedicated to real-time system design, which is part of REACT project. It describes the functional architecture of reactive systems and also some support for the description of the behaviour of the components and for the expression of real-time requirements (timing constraints) and properties (time budgets).	University of Nantes, France	2004	Real-time systems	Research Projects	[52]

TABLE A.1: General Characteristics of the ADLs

ADL	Description	Institutions	Year	Domain	Application	Reference
DAOP-ADL	The predecessor to AO-ADL and was created to allow applications written on the DAOP platform to be easily described and the ADL is directly interpreted by the platform, so retaining the architecture of the system at runtime. Like AO-ADL, aspects are first class architectural entities at design and runtime.	University of Malaga, Spain	2003	Distributed CBS	Case Studies	[140]
Darwin	Darwin was created to allow the creation and analysis of design specifications for distributed systems and its features include hierarchical decomposition and static and dynamic structures. It has formal semantics, specified in the π-Calculus. It has influenced many ADLs since, although it has not seen significant industrial usage.	Imperial College, UK	1996	Distributed systems	Research Projects	[108]
DiaSpec-ADL	DiaSpec-ADL addresses the needs of the pervasive systems domain. It supports a fairly specific architectural style based on sensors, controllers and actuators, provides simulation of the architectural model and generates framework applications for developers to complete to create the application.	INRIA	2009	Control Systems	Research Projects	[27]
DPD-ADL	Aimed at the description of systems in the areas of data collection, analysis and reporting (although in fact it is still a generic component and connector language).	Tsinghua University, China	2010	General	Examples	[[190]
DSOPL	ADL with the concept of SOA allows describing three types of information: architecture's structural elements, variability elements and system's configuration. Furthermore, it introduces context elements on which service reconfiguration is based.	University of Montpellier, France	2015	SOA	Examples	[3]

TABLE A.1: General Characteristics of the ADLs

ADL	Description	Institutions	Year	Domain	Application	Reference
EAST-ADL	EAST-ADL was originally developed for the avionics industry but has broadened its scope across embedded systems and has been developed by a range of academic and industrial research centres and standardised by the SAE in the USA. It is a component and connector based language, specialised for the needs of the embedded systems industry with strong support for product line concepts like variability. It has been used in a number of significant industrially based research projects.	Continental Automotive, ETAS, Mentor Graphics, Volvo, University of Hull, TU Berlin, Mecel, CEA, KTH, Carmeq	2010	Embedded Automotive	Research Projects	[37]
FuseJ	Created to support a component model, which unified components and aspects. The language describes architectural structures following this model, which are then executed in a novel container runtime also created as part of the project.	Vrije Universiteit Brussel, Brussels, Belgium	2005	General	Examples	[166]
Grasp	Created with the intent of adding traceable design rationale to architectural descriptions. The language is a component and connector based language and was supported by quite sophisticated tooling based on Microsoft's "Oslo" project.	University of St Andrews, UK	2011	General	Examples	[41]
I3	I3 was an early attempt at adding functional semantics to a component and connector language, by using coloured petri net (CPN) semantics in the language.	University of Illinois at Chicago, USA	1999	General	Examples	[28]
KADL	A formal architecture description language based on the Korrigan formal specification language. It was created to allow the definition of component-based systems with clear semantics, to abstract away from the details of individual component platforms like JEE and .NET. 7	Universite d'Evry and INRIA, France	2006	General (for CBS)	Case Studies	[141]

TABLE A.1: General Characteristics of the ADLs

ADL	Description	Institutions	Year	Domain	Application	Reference
Koala	Koala is an extension of Darwin, in order to support the development of embedded systems for consumer electronics. The primary motivation for the approach was to allow widespread reuse of components across many product configurations and so it contains product line features as well as the component based structures to describe an individual system	Phillips, Netherlands	2000	Embedded consumer electronic systems	Industrial Projects (100+ developers)	[174]
LEDA	LEDA was created to try to address perceived shortcomings of earlier ADLs, such as refinement, validation and analysis, particularly for dynamic systems. It attempts to address these shortcomings by using process algebras for the semantics of the description.	Universidad de Malaga, Spain	1999	General	Examples	[26]
MetaH	MetaH was developed to support the development of systems in the guidance, navigation and control (GN&C) domain. It was paired with Control-H, a domain specific language for GN&C, and provides the generic embedded software constructs to allow the system structure to be defined. It has been used quite widely on industrial projects and proof-of-concepts, particular in the US military domain.	Honeywell, USA	1996	Embedded Systems	Industrial Projects	[23]
MoDeL	MoDeL aims to provide a detailed "blueprint" for a distributed system, linking directly to the code structure and so it is closer to a module interconnection language than most of the other ADLs described here.	RWTH Aachen, Germany	2010	General	Case Studies	[89]

TABLE A.1: General Characteristics of the ADLs

ADL	Description	Institutions	Year	Domain	Application	Reference
MontiArcHV	Developed with the aim of integrating variability management into a hierarchical component model, in an attempt to reduce the complexity of managing software product lines.	RWTH Aachen, Germany	2011	Interactive distributed and Cyber-Physical systems	Examples	[66]
PrimitiveC-ADL	PrimitiveC provides a means to describe systems defined using the PCOM "context aware" component model, which was prior work of the same researchers. It is a component and connector based language, using XML as its notation, with a number of novel features needed for PCOM, including contextual conditions that can affect the runtime architectural configuration.	Trinity College of Dublin, Ireland	2010	General (context oriented systems)	Examples	[107]
PRISMA AOADL	An aspect oriented ADL, to allow the description applications being built on their PRISMA research platform and it allows aspects to be used as first class architectural constructs. PRISMA is based on a formal underlying language (OASIS), extending it with concepts like systems, components, connectors, aspects and architectural configuration.	Polytechnic University of Valencia, Spain	2006	General	Research Projects	[137]
Rapide	An ADL specifically for event based systems, which allows the architecture of an event based system to be rapidly defined and prototyped. Rapide defines systems as components with well-defined interfaces that exchange events and as such does not provide first class connectors. It has been influential in the event driven systems domain.	Stamford University, TRW Research, USA	1995	Event driven systems	Case Studies	[106]

TABLE A.1: General Characteristics of the ADLs

ADL	Description	Institutions	Year	Domain	Application	Reference
SADL	The focus is on the functional view of a system and its provably correct refinement to implementation, by allowing the formal definition of architectural structures and the relationships between them. In particular SADL aims to provide support for the definition of architectural hierarchies.	SRI Computer Science Laboratory, USA	1995	General	Case Studies	[120]
Service-ADL	Service-ADL provides modelling elements for interaction patterns defining services, as well as for mapping sets of services to target component configurations. The language describes a comprehensive software development process that considers services as first class modelling elements. By decoupling the modelling of services from their implementation on target component configurations this process enables exploration of multiple architectures implementing the same set of services. The view of services as cross-cutting architectural aspects is substantiated by providing a mapping from services to aspects in AspectJ.	University of California, San Diego, USA	2004	SOA	Experiments	[94]

TABLE A.1: General Characteristics of the ADLs

ADL	Description	Institutions	Year	Domain	Application	Reference
SKwyRL-ADL	The motivation for this language's development was the development of secure multi-agent systems for distributed information systems and the lack of an ADL that met their unique needs. The language is based on a computation model called "believe-desire-intention" that views the world as a set of independently executing autonomous agents which are each trying to achieve their goals. Agents execute by reacting to events which are generated as a result of a change of state, goals or messages from the environment (such as other agents).	University of Louvain, Belgium	2003	Multi-agent systems	Experiments	[53]
SOADL-EH	SOADL is an ADL for service-based systems and specifies the interfaces, behaviour, semantics and quality properties for services and allows modelling and analysis of a service-based architecture. As the 'EH' in its name suggests, it explicitly provides error handling constructs in the ADL.	Wuhan University, China; China University of Geoscience, China	2012	SOA	Experiments	[175]
TADL	TADL was developed to extend a component and connector style language to include security concepts as first class constructs. As such as well as the usual architectural primitives, it includes concepts like safety contracts and security mechanisms.	Concordia University, Canada	2008	General	Experiments	[118]

TABLE A.1: General Characteristics of the ADLs

ADL	Description	Institutions	Year	Domain	Application	Reference
UniCon	This language implements an early component and connector model of software architecture and is one of the early attempts at introducing connectors as first class language elements. The aim of the language was to create a useful, pragmatic and extensible test-bed that would allow the architectural abstractions used by practitioners (such as pipes, filters, objects, clients and servers) to be captured and reasoned about in a systematic manner.	Carnegie Mellon University, USA	1995	General	Experiments	[158]
vADL	vADL is an ADL for product lines. It is a component and connector based language that uses the pi-Calculus to define its semantics and provides explicit variability support within architectural elements.	School of Computer Science, Northwestern Polytechnic University, China	2005	Product Line Architecture (PLA)	Examples	[189]
Weaves	A visual ADL developed to investigate how visual programming could be applied to large scale problems for an architectural style that constructs a software system from a directed graph of separate computational elements communicating by transferring typed objects over lightweight message queues. Weaves is particularly suited to domains that involve stream processing, such as satellite telemetry and satellite ground station prototyping.	The Aerospace Corporation and The University of Hawaii, USA	1991	Distributed stream processing	Case Studies	[64]

TABLE A.1: General Characteristics of the ADLs

ADL	Description	Institutions	Year	Domain	Application	Reference
Wright	A formally defined ADL to allow automated analysis of the architectural description. Wright is a component and connector language that was developed to have well defined semantics and a set of reasoning techniques to allow architectural analysis. It allows both architectures for individual systems and architectural styles to be described.	Carnegie Mellon University, USA	1998	General	Case Studies	[6]
xADL	An XML based component and connector language defined as a set of XML schemas. Designed to use standard XML infrastructure and be easily extensible using standard XML-Schema extension mechanisms. xADL was developed over an extended period, from about 2001 onwards. The current version of xADL is 3.0 and the language tool set it still being actively enhanced at the time of writing.	University California Irvine and The SEI, USA	2005	General	Research Projects	[39]
XYZ/ADL	A formal ADL which was developed as part of a wider programme of research in the area of service oriented architecture (SOA). XYZ/ADL is a component and connector based language with formal semantics to allow it to be used as the basis of analysis techniques. Most of the literature on the language is in Chinese.	Chinese Academy of Sciences, China	2011	General	Examples	[188]
ZETA	An ADL that focuses on component interactions, with the intent of it being used to define the composition of complex components in order to create systems from them. The key concepts ZETA provides are components (and interfaces) and the concepts and semantics of messaging to link components.	University of Savoie at Annecy, France	2002	Distributed Software-intensive system	Case Studies	[7]

TABLE A.1: General Characteristics of the ADLs

ADL	Description	Institutions	Year	Domain	Application	Reference
π-ADL	A formal architecture description language for describing distributed and mobile systems. The language principles are that it should be a formal language, that it will focus on the runtime aspects of a system, that it should be executable and that it should be user-friendly (meaning that a number of syntaxes should be available for different uses). As its name suggests, its semantics of the language are based on an extension of π-calculus	University of Savoie at Annecy, France	2004	Distributed systems, particularly with a mobile aspect	Case Studies	[134]
π-SPACE	A component and connector based language based on the π-calculus that has a focus on architectural evolution. As well as allowing the definition of components and connectors, it allows the architectural description to describe how they can be added, removed or changed during operation.	University of Savoie at Annecy, France	2000	General	Case Studies	[30]

TABLE A.2: ADL Support for Architectural Concepts

ADL	Viewpoints	Architectural Concepts	Behavioural Semantics	1st Class Conns	1st Class Config
AADL	Functional, Deployment	Component (process, thread, thread group, data, subprogram, processor, memory, device, bus); Port (data port, event data port, synchronous call, direct data access); Interconnection type (message passing, event passing, synchronised data access, RPC); Property System; Information Flow; Operational Mode; Package Annex; Library (extension or specialisation)	Yes (but user defined form)	No	Yes
ABC/ADL	Functional, Deployment	Component; Connector; Aspect; Architecture; Configuration	Yes (but user defined form)	Yes	Yes
AC2-ADL	Functional	Component; Aspect Component; Connector; Aspect Connector; Architecture; Configuration	No	Yes	Yes
ACDL	Functional	Architecture type; Component type; Connector; Architecture (configuration)	Yes (π-Calculus)	Yes	Yes
ACME	Functional	Component; Representation-Map (to capture alternative implementation options for components); Connector; System; Template and Style	No	Yes	Yes
ADLARS	Functional, Deployment	Component; Task; System; Interaction Theme; Feature	Yes (but user defined form)	No	Yes
ADML	Functional	System; Component; Connector	Yes (built in)	Yes	No
Aesop	Functional	Component; Connector; Configuration; Architectural Style	No	Yes	Yes
ALI	Functional	Interface; Component; Connector; System; Pattern Template; Variant Selection (based on feature catalogue)	No	Yes	Yes
AO-ADL	Functional	Component; Connector; Aspect; Architectural Configuration	No (although other notations can be used)	Yes	Yes
Archface	Functional, Development	Component; Connector; Aspect (pointcut & advice)	Yes (code corresponding to the ADL structure)	Yes	No
Aspectual-ACME	Functional	Extends ACME concepts with:; Aspectual Connector	No	Yes	Yes
Backbone	Functional	Component (ports, parts [leaf component], connectors)	No	No	No
Breeze/ADL	Functional	Component; Connector; Interface; Configuration; Style constraints and state transformation	No	Yes	Yes
byADL	Functional	Components and Interfaces; Connectors and Interfaces; Architectural Configurations	No	Yes	Yes

TABLE A.2: ADL Support for Architectural Concepts

ADL	Viewpoints	Architectural Concepts	Behavioural Semantics	1st Class Conns	1st Class Config
C2SADEL	Functional	Components and Interfaces; Connectors (but only C2 message connections); Architectural configuration; Message Filters (allowing dynamic architectures)	Yes (built in)	Yes	Yes
C3	Functional	Component; Connector (properties, constraints, services, hierarchical level, glue and attachment); Configuration	No	Yes	Yes
CBabel	Functional	Components (âĂIJmodulesâĂİ); Connectors; Contracts; Architectural Configuration	Yes (built in)	Yes	Yes
CLARA	Functional	Component; Links; Configuration	Yes (Petri-net specifications)	Yes	Yes
DAOP-ADL	Functional	Components; Properties; Aspects; Architectural Composition Rules	No	No (implicit in the platform)	Yes
Darwin	Functional	Components; Services; Architectural Configuration (via composite components)	No	No	Yes
DiaSpec-ADL	Functional	Components (Devices, Controllers & Contexts); Interaction Specifications	No	No	Yes
DPD-ADL	Functional	Components; Connectors	No	Yes	Yes
DSOPL	Functional	Structural description (services [interfaces]); Variability description (variation point, reference element);; Context description (context type); Configuration description (initialization, dynamic	No	No	Yes
EAST-ADL	Functional, Dep, Development	Too many to list, very wide spectrum and detailed language, covering System, Feature, Function, Hardware and Environment modelling.	Yes (via links to descriptions like Simulink or MATLAB)	Yes	Yes
FuseJ	Functional, Deployment	Component (regular component or aspect); Interface (and Gate); Connector	No	Yes	No
Grasp	Functional	Requirement; Quality Property; Components ; Connectors; Layers; Systems; Architectures; Templates; Rationale	No	Yes	Yes
I3	Functional	Component; Interface Net; Interconnection Net; Interoperation Net	Yes (Petri-Nets)	No	No
KADL	Functional	Components and Ports; Architectural Configuration (âĂIJglue rulesâĂİ)	Yes (Korrigan)	No	Yes
Koala	Functional, Development	Components and Interfaces; Architectural Configuration	No (link to code)	No	Yes

TABLE A.2: ADL Support for Architectural Concepts

ADL	Viewpoints	Architectural Concepts	Behavioural Semantics	1st Class Conns	1st Class Config
LEDA	Functional	Components; Roles; Composition; Attachment; Adaptors (âĂIJglueâĂİ allowing construction of composites from components which are not strictly compatible).	Yes (π-Calculus)	No	No
MetaH	Functional, Concurrency, Deployment	Components (âĂIJprocessesâĂİ); Hardware Components; Configuration (âĂIJconnectionsâĂİ and âĂIJmodesâĂİ); Application	No (separate ControlH language)	No	Yes
MoDeL	Functional, Concurrency	Module; Subsystem; Process; Mutex	Yes (Z specifications)	No	No
MontiArcHV	Functional, Deployment	Component; Port; Connector; Variation Point; Variant; Variant Config	No	No	Yes
PrimitiveC-ADL	Functional	Components; Connectors; Design Pattern; Decision Policy; Architecture Configuration	Yes (no details provided)	Yes	Yes
PRISMA AOADL	Functional	Component; Connector; Aspect; Architectural Pattern; Architectural Configuration	Yes (π-Calculus)	Yes	Yes
Rapide	Functional, Concurrency	Components (as interfaces) and Functions, Actions and Services; Architectures (with connection rules and constraints); Events and Posets	Yes (built in)	No	Yes
SADL	Functional	Components and Interfaces (containing ports); Connectors; (Architectural) configurations; Mappings (between architectures); Architectural Styles; Refinement Patterns	No	Yes	Yes
Service-ADL	Functional, Development	Roles (states); Services (interaction); Component (plays, in service); Configuration	Yes	No	Yes
SKwyRL-ADL	Functional	Agents (as components); Ports (sensors or effectors) ; Knowledge Base; Goals and Plans; Services; Architectural Configuration; Architecture	No	No	Yes
SOADL-EH	Functional, Deployment	Components; Connectors; Service; Policy; Configuration	Yes (π-calculus, code)	Yes	Yes
TADL	Functional, Deployment	Interface Type; Connector Role Type; Connector Type; Architecture Type; Component Type	No	Yes	Yes
UniCon	Functional	Components with Interfaces; Connectors with Protocols; Properties; Architectural Configuration (via composite components)	No (link to code)	Yes	Yes
vADL	Functional	Component; Connector; Interface; Behaviour; Architecture	Yes (π-calculus)	No	No

TABLE A.2: ADL Support for Architectural Concepts

ADL	Viewpoints	Architectural Concepts	Behavioural Semantics	1st Class Conns	1st Class Config
Weaves	Functional	Components (âĂIJtool fragmentsâĂİ) and Ports; Connectors (message queues); Annotations	No (but link to code)	Yes	No
Wright	Functional	Components and Interfaces; Connectors and Roles; Architectural Configurations (âĂIJattachmentsâĂİ); Styles	Yes (CSP extension)	Yes	Yes
xADL	Functional, Deployment	Components and Interfaces; Connectors and Interfaces; Architectural Configuration (âĂIJlinksâĂİ); Runtime Structure (component instances, link instances)	No	Yes	Yes
XYZ/ADL	Functional	Components and Interfaces; Connectors and Roles; Architectural configurations (via composite components)	Yes (XYZ/E temporal logic language)	Yes	Yes
ZETA	Functional	Interaction Interface (Activity Interface, Artefact Interface, Message Types, Port Interface, Protocol Interface); Interaction; Message; Port; Attachment; Container Types	Yes (built in)	Yes	No
π-ADL	Functional	Components and Ports; Agents (mobile components); Connectors and Ports; Architecture (configuration)	Yes (π-calculus)	Yes	Yes
π-SPACE	Functional	Port Type; Behaviour Type; Component Type; Connector Type; Composition/ Decomposition/ Recomposition	Yes (π-calculus)	Yes	Yes

TABLE A.3: ADL Language Mechanisms and Support

ADL	Structuring	Capturing Qualities	Syntax	Analysis	Tools
AAADL	Packages; Composition	Properties	Textual; Graphical	Yes (Performance; Real-Time Scheduling; Reliability); others via extensions	Commercial
ABC/ADL	Composition	Properties	Textual; Graphical	Via 3rd party tools	Prototype
AC2-ADL	Composition	None	Textual	None	None
ACDL	Composition	None	Textual	Yes (Consistency)	None
ACME	Composition	Properties	Textual; Graphical	Via ADL extensions or 3rd party tools	Research
ADLARS	Composition	Specific Attributes (for Timing)	Textual (JavaCC); Graphical	Yes (Concurrency; Consistency)	Prototype
ADML	Composition	Properties	Textual	None	None
Aesop	Composition	Style Properties	Textual; Graphical	Via 3rd party tools	Research
ALI	Composition	None	Textual; Graphical	None	None
AO-ADL	Composition	None	Textual; Graphical	None	Research
Archface	None	None	Textual	Via 3rd party tools	Prototype
Aspectual-ACME	Composition	Properties	Textual; Graphical	Via 3rd party tools	None
Backbone	Composition	None	Textual; Graphical	None	Prototype
Breeze/ADL	Packages; Composition	None	Textual (XML-based)	None	None
byADL	Packages; Composition	Properties	Various	Via ADL extensions	Research
C2SADEL	None	None	Textual; Graphical	Via ADL extensions	Research

TABLE A.3: ADL Language Mechanisms and Support

ADL	Structuring	Capturing Qualities	Syntax	Analysis	Tools
C3	Composition	Properties	Textual; Graphical (UML)	None	None
CBabel	None	Language QoS Contracts; Properties	Textual	Yes (Concurrency; Consistency)	None
CLARA	Composition	Specific Attributes (for Timing)	Textual; Graphical	Yes (Concurrency; Consistency)	None
DAOP-ADL	None	None	Textual (XML); Graphical	None	Research
Darwin	Composition	Properties	Textual; Graphical	Via 3rd party tools	Research
DiaSpec-ADL	None	None	Textual	Yes (Consistency)	Research
DPD-ADL	Composition	None	Textual; Graphical (UML)	None	None
DSOPL	Packages; Composition	None	Textual (XML); Graphical (UML)	None	None
EAST-ADL	Packages; Composition (inherited from UML)	Language Elements (for Timing, Safety, Dependability); Properties	Graphical notation, based on and extending UML (i.e. graphical notation and semi-structured natural language)	Via 3rd party tools	Commercial
FuseJ	Packages; Composition	None	Textual	None	Prototype
Grasp	Composition	Specific Attributes; Properties	Textual	None	Prototype
I3	None	None	Loosely defined graphical notation; Petri net representations	Yes (Concurrency)	None
KADL	Composition	None	Textual; Graphical	Yes (Consistency)	Prototype

TABLE A.3: ADL Language Mechanisms and Support

ADL	Structuring	Capturing Qualities	Syntax	Analysis	Tools
Koala	Packages; Composition	None	Textual; Graphical	None	Commercial
LEDA	Composition	None	Textual	Yes (Consistency)	None
MetaH	Composition	Specific Attributes (for Timing, Safety, Dependability)	Textual; Graphical	Yes (Real-Time Scheduling)	Commercial
MoDeL	Subsystems; Composition	None	Textual; Graphical	None	Research
MontiArcHV	Packages; Composition	None	Textual; Graphical.	None	None
PrimitiveC-ADL	Composition	None	Textual (XML based)	None	None
PRISMA AOADL	Composition	None	Textual; UML Profile	None	Research
Rapide	Composition	None	Textual; Simplified graphical	Yes (Correctness; Timing)	Research
SADL	Composition	None	Textual	None	Research
Service-ADL	Packages; Composition	None	Textual; Graphical	None	None
SKwyRL-ADL	Composition	Language Elements (for Security - security constraints, mechanisms, etc.)	Textual	None	None
SOADL-EH	Packages; Composition	Properties	Textual (XML), Graphical	Yes (Concurrency; Consistency)	None
TADL	Packages; Composition	Language Elements (for Safety and Security, safety constraints, security structures); Properties	Textual	Yes (Safety; Timing; Security)	None
UniCon	Composition	Properties	Textual; Graphical	Via 3rd party tools	Prototype
vADL	Composition	Properties	Textual	None	None

TABLE A.3: ADL Language Mechanisms and Support

ADL	Structuring	Capturing Qualities	Syntax	Analysis	Tools
Weaves	Composition	Properties	Graphical	Yes (Performance)	Research
Wright	Composition	None	Textual	Yes (Correctness)	Prototype
xADL	Composition	Properties (via extension)	XML (and tool specific graphical)	None	Research
XYZ/ADL	Composition	None	Textual (with mathematics)	Yes (Consistency)	None
ZETA	Composition	None	Textual; Graphical	None	None
π-ADL	Composition	Properties	Textual; Graphical (UML extension)	Yes (Correctness; Consistency)	Research
π-SPACE	Composition	None	Textual; Graphical (UML)	None	None

Bibliography

[1] Hayri Acar, Gülfem I. Alptekin, Jean-Patrick Gelas, and Parisa Ghodous. Beyond CPU: Considering memory power consumption of software. In *SMARTGREENS 2016 - Proceedings of the 5th International Conference on Smart Cities and Green ICT Systems*, pages 1–8, 2016.

[2] Mark Acton, Paolo Bertoldi, John Booth, Liam Newcombe, Andre Rouyer, and Robert Tozer. 2018 Best Practice Guidelines for the EU Code of Conduct on Data Centre Energy Efficiency. Technical report, European Union Joint Research Centre, 2018.

[3] Seza Adjoyan and Abdelhak-Djamel Seriai. An architecture description language for dynamic service-oriented product lines. In *SEKE: Software Engineering and Knowledge Engineering*, 2015.

[4] David Allen. *Getting Things Done: The Art of Stress-free Productivity*. Piatkus, 2015.

[5] Robert Allen, Steve Vestal, Dennis Cornhill, and Bruce Lewis. Using an architecture description language for quantitative analysis of real-time systems. In *Proceedings of the third international workshop on Software and performance - WOSP 02*. ACM Press, 2002.

[6] Robert J. Allen. *A Formal Approach to Software Architecture*. PhD thesis, Carnegie Mellon University, 1997.

[7] Ilham Alloui and Flavio Oquendo. Supporting decentralised software-intensive processes using zeta component-based architecture description language. *Enterprise Information Systems III*, 2001.

[8] Abdelkrim Amirat and Mourad Oussalah. First-class connectors to support systematic construction of hierarchical software architecture. *The Journal of Object Technology*, 8(7):107–130, 2009.

[9] Nadine Amsel and Bill Tomlinson. Green Tracker : A Tool for Estimating the Energy Consumption of Software. In *CHI '10 Extended Abstracts on Human Factors in Computing Systems*, pages 3337–3342, 2010.

[10] Mike Amundsen, Matt McLarty, Ronnie Mitra, and Irakli Nadareishvili. *Microservice Architecture: aligning principles, practices, and culture*. O'Reilly Media, 2016.

[11] AppDynamics Inc. AppDynamics App iQ Platform Documentation. https://docs.appdynamics.com, 2018.

[12] Muhammad Ali Babar and Ian Gorton. Comparison of scenario-based software architecture evaluation methods. In *Software Engineering Conference, 2004. 11th Asia-Pacific*, pages 600–607. IEEE, 2004.

[13] Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Paulo Merson, Robert Nord, and Judith Stafford. *Documenting Software Architectures: Views and Beyond*. Addison-Wesley Professional, 2011.

[14] Rami Bahsoon. A framework for dynamic self-optimization of power and dependability requirements in green cloud architectures. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2010.

[15] Rabih Bashroush. A Comprehensive Reasoning Framework for Hardware Refresh in Data Centres. *IEEE Transactions on Sustainable Computing*, page 1, 2018.

[16] Rabih Bashroush, T. John Brown, Ivor Spence, and Peter Kilpatrick. ADLARS: An Architecture Description Language for Software Product Lines. In *29th Annual IEEE/NASA Software Engineering Workshop*, pages 163–173. Ieee, 2005.

[17] Rabih Bashroush, Ivor Spence, Peter Kilpatrick, and John Brown. Towards More Flexible Architecture Description Languages for Industrial Applications, 2006.

[18] Rabih Bashroush, Ivor Spence, Peter Kilpatrick, T. John Brown, Wasif Gilani, and Mathias Fritzsche. ALI: An Extensible Architecture Description Language for Industrial Applications. In *15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2008)*, pages 297–304. IEEE, mar 2008.

[19] Rabih Bashroush and Eoin Woods. Architectural Principles for Energy-Aware Internet-Scale Applications. *IEEE Software*, 34(3):14–17, 2017.

[20] Rabih. Bashroush, Eoin. Woods, and Adel. Noureddine. Data center energy demand: What got us here won't get us there. *IEEE Software*, 33(2):18–21, 2016.

[21] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison Wesley, 3 edition, 2012.

[22] Roy F. Baumeister and Mark R. Leary. Writing narrative literature reviews. *Review of General Psychology*, 1(3):311–320, 1997.

[23] Pam Binns, Matt Englehart, Mike Jackson, and Steve Vestal. Domain Specific Software Architectures for Guidance, Navigation and Control. *International Journal of Software Engineering and Knowledge Engineering*, 06(02):201–227, jun 1996.

[24] Aurélien Bourdon, Adel Noureddine, Romain Rouvoy, and Lionel Seinturier. PowerAPI: A Software Library to Monitor the Energy Consumed at the Process-Level, 2013.

[25] Simon Brown. Software Architecture for Developers, eBook. https://softwarearchitecturefordevelopers.com, 2018.

[26] Calos Canal, Ernesto Pimentel, and José M Troya. Specification and refinement of dynamic software architectures. In *Software Architecture*, pages 107–125. Springer, 1999.

[27] Damien Cassou, Benjamin Bertran, Nicolas Loriant, and Charles Consel. A generative programming approach to developing pervasive computing systems. *ACM Sigplan Notices*, 45(2):137–146, 2009.

[28] Carl K Chang and Seongwoon Kim. I/sup 3: a Petri-net based specification method for architectural components. In *Computer Software and Applications*

Conference, 1999. COMPSAC'99. Proceedings. The Twenty-Third Annual International, pages 396–402. IEEE, 1999.

[29] Kathy Charmaz. *Constructing grounded theory: a practical guide through qualitative analysis*. Sage, 2006.

[30] Christelle Chaudet and Flavio Oquendo. pi-SPACE: a formal architecture description language based on process algebra for evolving software systems. In *Automated Software Engineering, 2000. Proceedings ASE 2000. The Fifteenth IEEE International Conference on*, pages 245–248. IEEE, 2000.

[31] Feifei Chen, John Grundy, Jean-Guy Schneider, Yun Yang, and Qiang He. Automated analysis of performance and energy consumption for cloud applications. In *Proceedings of the 5th ACM/SPEC international conference on Performance engineering*, pages 39–50. ACM, 2014.

[32] Cisco. United Kingdom: Prioritising Security at Management and Board Levels. Technical report, Cisco Systems Inc, 2016.

[33] Cloud Foundry Foundation. Cloud Foundry Home Page. https://docs.cloudfoundry.org, 2018.

[34] Cloud Native Computing Foundation. Open Tracing - What is a span? http://opentracing.io/documentation/#what-is-a-trace, 2018.

[35] Nelly Condori-Fernandez and Patricia Lago. Can we know upfront how to prioritize quality requirements? In *5th International Workshop on Empirical Requirements Engineering, EmpiRE 2015 - Proceedings*, 2015.

[36] Victor Cordero, Ignacio García Rodríguez De Guzmán, and Mario Piattini. A first approach on legacy system energy consumption measurement. In *Proceedings - 2015 IEEE 10th International Conference on Global Software Engineering Workshops, ICGSEW 2015*, pages 35–43, 2015.

[37] Philippe Cuenot, Patrick Frey, Rolf Johansson, Henrik Lönn, Yiannis Papadopoulos, Mark Oliver Reiser, Anders Sandberg, David Servat, Ramin Tavakoli Kolagari, Martin Törngren, and Matthias Weber. The EAST-ADL architecture description language for automotive embedded software. In *Lecture*

Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2010.

[38] Current Analysis Inc. Delivering Energy Efficiency for Australian Data Centres. Technical report, Current Analysis Inc, 2016.

[39] Eric M Dashofy, André van der Hoek, and Richard N Taylor. A comprehensive approach for the development of modular software architecture description languages. *ACM Transactions on Software Engineering and Methodology*, 14(2):199–245, apr 2005.

[40] Miyuru Dayarathna, Yonggang Wen, and Rui Fan. Data Center Energy Consumption Modeling: A Survey. *IEEE Communications Surveys & Tutorials*, 18(1):732–794, 2016.

[41] Lakshitha de Silva and Dharini Balasubramaniam. A Model for Specifying Rationale Using an Architecture Description Language. In Ivica Crnkovic, Volker Gruhn, and Matthias Book, editors, *5th European Conferene on Software Architecture (ECSA 2011), LNCS 6903*, volume 6903 of *Lecture Notes in Computer Science*, pages 319–327. Springer Berlin / Heidelberg, 2011.

[42] Pierre Delforge. America's Data Centers Are Wasting Huge Amounts of Energy. *Natural Resources Defense Council (NRDC)*, 2014.

[43] Davide Di Ruscio, Ivano Malavolta, Henry Muccini, Patrizio Pelliccione, and Alfonso Pierantonio. ByADL: An MDE framework for building extensible architecture description languages. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2010.

[44] Distributed Management Task Force. Redfish Scalable Platforms Management API Specification. Technical report, Distributed Management Task Force, 2018.

[45] Docker Inc. Docker Home Page. https://www.docker.com, 2018.

[46] Dynatrace Inc. Dynatrace Documentation. https://www.dynatrace.com/support/doc, 2018.

[47] EBay. Digital Service Efficiency, 2013.

[48] ENEA. Description of Energy Metrics for Data Centres D7.1. Technical report, DC4Cities Project, 2014.

[49] David Evans. The Internet of Things Infographic. http://blogs.cisco.com/diversity/the-internet-of-things-infographic, 2008.

[50] Eric Evans. *Domain Driven Design*. Addison Wesley, 2006.

[51] Davide Falessi, Giovanni Cantone, Rick Kazman, and Philippe Kruchten. Decision-making techniques for software architecture design. *ACM Computing Surveys*, 2011.

[52] Sébastien Faucou, Anne-Marie Déplanche, and Yvon Trinquet. An ADL centric approach for the formal design of real-time systems. In *Architecture Description Languages*, pages 67–82. Springer, 2005.

[53] Stéphane Faulkner and Manuel Kolp. Towards an agent architectural description language for information systems. In Olivier Camp and Mario Piattini, editors, *5th International Conference on Enterprise Information Systems (ICEIS 03)*, pages 59–66, Angers, France, 2003. Kluwer.

[54] Peter H Feiler, David P Gluch, and John J Hudak. The Architecture Analysis & Design Language (AADL): An Introduction. Technical Report CMU/SEI-2006-TN-011, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, 2006.

[55] Peter H Feiler, Bruce Lewis, and Steve Vestal. Improving Predictability in Embedded Real-Time Systems. Technical Report December, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, dec 2000.

[56] Florian Forster. collectd Documentation. https://collectd.org/documentation.shtml, 2018.

[57] John F. Gantz and Pam Miller. The Salesforce Economy: Enabling 1.9 Million New Jobs and $389 Billion in New Revenue Over the Next Five Years. Technical report, International Data Corporation (IDC), 2016.

[58] Alessandro Garcia, Christina Chavez, Thais Batista, Cláudio Sant'Anna, Uirá Kulesza, Awais Rashid, and Carlos Lucena. On the modular representation of architectural aspects. In *European Workshop on Software Architecture*, pages 82–97. Springer, 2006.

[59] David Garlan, Robert Allen, and John Ockerbloom. Exploiting style in architectural design environments. *ACM SIGSOFT Software Engineering Notes*, Volume 19(Issue 5):175 – 188, 1994.

[60] David Garlan, R Monroe, and D Wile. ACME : An Architecture Description Interchange Language. In J. Howard Johnson, editor, *CASCON '97 Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research*, pages 169–183, Toronto, Ontario, 1997. IBM Press.

[61] Bill Gillham. *Developing a questionnaire*. A and C Black, 2008.

[62] Sebastien Godard. Unix sar(1) command manual page. https://linux.die.net/man/1/sar, 2018.

[63] Google Inc. cAdvisor Home Page. https://github.com/google/cadvisor, 2018.

[64] Michael. M. Gorlick and Rami. R. Razouk. Using weaves for software construction and analysis. In *13th International Conference on Software Engineering*, pages 23–34, Austin, TX, 1991. IEEE Comput. Soc. Press.

[65] Håkan Grahn and Jan Bosch. Some initial performance characteristics of three architectural styles. In *Proceedings of the 1st international workshop on Software and performance*, pages 197–198. ACM, 1998.

[66] Arne Haber, Holger Rendel, Bernhard Rumpe, Ina Schaefer, and Frank Van Der Linden. Hierarchical variability modeling for software architectures. In *Software Product Line Conference (SPLC), 2011 15th International*, pages 150–159. IEEE, 2011.

[67] Adrian Hilton. CRE life lessons: What is a dark launch, and what does it do for me? *Google Cloud Platform Blog*, 2017.

[68] Hitachi. Hitachi Ultrastar C10K1800 Hard Drive Data Sheet. https://www.hgst.com/sites/default/files/resources/USC10K1800_ds.pdf, 2015.

[69] iDA Singapore. Green Data Centre Innovation Programme. https://www.imda.gov.sg/infocomm-and-media-news/buzz-central/2015/5/greening-the-data-centre, 2015.

[70] InfluxData Inc. InfluxDB Documentation, 2018.

[71] InfluxData Inc. Telegraf Documentation. https://docs.influxdata.com/telegraf, 2018.

[72] Uptime Institute. Tier Classification System. http://uptimeinstitute.com/tiers, 2015.

[73] Intel Corporation, Hewlett Packard Company, NEC Corporation, and Dell Inc. Intelligent Platform Management Interface Specification, v2.0. Technical report, Intel Corporation, 2013.

[74] International Organization for Standardization. Information technology – Data centres – Key performance indicators – Part 2: Power usage effectiveness (PUE). Technical report, ISO, Geneva, CH, 2016.

[75] Syed Islam, Adel Noureddine, and Rabih Bashroush. Measuring energy footprint of software features. In *IEEE International Conference on Program Comprehension*, 2016.

[76] Istio Authors. Mirroring. https://istio.io/docs/tasks/traffic-management/mirroring, 2018.

[77] Jaeger Committers. Jaeger Home Page. https://www.jaegertracing.io, 2018.

[78] Erik Jagroep, Jan Martijn van der Werf, Sjaak Brinkkemper, Leen Blom, and Rob van Vliet. Extending software architecture views with an energy consumption perspective. *Computing*, 99(6):553–573, 2017.

[79] Erik A. Jagroep, Jan Martijn van der Werf, Sjaak Brinkkemper, Giuseppe Procaccianti, Patricia Lago, Leen Blom, and Rob van Vliet. Software energy profiling: Comparing releases of a software product. In *Proceedings of the 38th International Conference on Software Engineering Companion - ICSE '16*, pages 523–532, 2016.

[80] Erik A Jagroep, Jan Martijn E M van der Werf, Ruvar Spauwen, Leen Blom, Rob van Vliet, and Sjaak Brinkkemper. An energy consumption perspective on software architecture. In *European Conference on Software Architecture*, pages 239–247. Springer, 2015.

[81] Dmitry Jemerov and Svetlana Isakova. *Kotlin in Action*. Manning Publications Company, 2017.

[82] Wen Jing, Ying Shi, Zhang LinLin, and Ni YouCong. AC2-ADL: Architectural description of aspect-oriented systems. In *Advanced Software Engineering and Its Applications, 2008. ASEA 2008*, pages 147–152. IEEE, 2008.

[83] Andrew Josey, Donald Cragun, Nicholas Stoughton, Mark Brown, and Cathy Hughes. The Open Group base specifications issue 6 IEEE Standard 1003.1. Technical report, The IEEE and The Open Group, 2004.

[84] Melanie Kambadur and Martha A. Kim. An experimental survey of energy management across the stack. In *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications - OOPSLA '14*, pages 329–344, 2014.

[85] Aman Kansal and Feng Zhao. Fine-grained energy profiling for power-aware application design. *ACM SIGMETRICS Performance Evaluation Review*, 36(2):26–31, 2008.

[86] Rohit Khare, Michael Guntersdorfer, Peyman Oreizy, Nenad Medvidovic, and Richard N. Taylor. xADL: enabling architecture-centric tool integration with XML. In *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*, page 9. IEEE Comput. Soc, 2001.

[87] Gene Kim, Jez Humble, Patrick Debois, and John Willis. The DevOps Handbook : How to Create World-Class Agility, Reliability, and Security in Technology Organizations, 2016.

[88] Barbara Kitchenham and Stuart Charters. Guidelines for performing Systematic Literature Reviews in Software Engineering. *Engineering*, 2007.

[89] Peter Klein. *Architecture Modeling of Distributed and Concurrent Software Systems*. Phd, heinisch-WestfaÌĹlischen Technischen Hochschule Aachen, 2000.

[90] Richard Koch. *The 80/20 principle: the secret to achieving more with less.* Bantam Doubleday Dell Publishing Group, 1998.

[91] Jonathan G. Koomey, Stephen Berard, Marla Sanchez, and Henry Wong. Implications of historical trends in the electrical efficiency of computing. *IEEE Annals of the History of Computing*, 2011.

[92] Philippe Kruchten. Architectural blueprints - the '4+ 1' view model of software architecture. *IEEE Software*, 12(6):42–50, 1995.

[93] Philippe Kruchten. What do software architects really do? *Journal of Systems and Software*, 81(12):2413–2416, 2008.

[94] Ingolf Heiko Kruger and Reena Mathew. Systematic development and exploration of service-oriented software architectures. In *Software Architecture, 2004. WICSA 2004. Proceedings. Fourth Working IEEE/IFIP Conference on*, pages 177–187. IEEE, 2004.

[95] Kubernetes Authors. Kubernetes Documentation. https://kubernetes.io/docs, 2018.

[96] Klaus Dieter Lange. Identifying shades of green: The SPECpower benchmarks. *Computer*, 42(3):92—-97, 2009.

[97] Marc M. Lankhorst, Henderik Alex Proper, and Henk Jonkers. The architecture of the ArchiMate language. In *Lecture Notes in Business Information Processing*, volume 29 LNBIP, pages 367–380. Springer Berlin Heidelberg, 2009.

[98] Stephen Lapan, MaryLynn Quartaroli, and Frances Riemer. *Qualitative research: An introduction to methods and designs.* John Wiley and Sons, 2012.

[99] Grace A. Lewis and Patricia Lago. A Catalog of Architectural Tactics for Cyber-Foraging. In *Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures - QoSA '15*, 2015.

[100] Grace A Lewis, Patricia Lago, and Paris Avgeriou. A decision model for cyber-foraging systems. In *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, pages 51–60. IEEE, 2016.

[101] Chen Li, Linpeng Huang, Luxi Chen, and Chengyuan Yu. Breeze/ADL: Graph grammar support for an xml-based software architecture description language. In *Computer Software and Applications Conference (COMPSAC), 2013 IEEE 37th Annual*, pages 800–805. IEEE, 2013.

[102] Jin Liu, Jiaming Jiang, Xiaohui Cui, Wei Yang, and Xiao Liu. Power consumption prediction of web services for energy-efficient service selection. *Personal and Ubiquitous Computing*, 19(7):1063–1073, 2015.

[103] Chris Loken, Daniel Gruner, Leslie Groer, Richard Peltier, Neil Bunn, Michael Craig, Teresa Henriques, Jillian Dempsey, Ching Hsing Yu, Joseph Chen, L. Jonathan Dursi, Jason Chong, Scott Northrup, Jaime Pinto, Neil Knecht, and Ramses Van Zon. SciNet: Lessons learned from building a power-efficient top-20 system and data centre. *Journal of Physics: Conference Series*, 256(1):1–35, 2010.

[104] Henrik Lönn, Tripti Saxena, Mikael Nolin, and Martin Törngren. FAR EAST: modeling an automotive software architecture using the EAST ADL. In *"Software Engineering for Automotive Systems" Workshop W14S - 26th International Conference on Software Engineering*, pages 43–50. IEE, 2004.

[105] Orlando Loques and Alexandre Sztajnberg. Customizing Component-Based Architectures by Contract. In *Lecture Notes in Computer Science*, pages 18–34. Springer Berlin Heidelberg, 2004.

[106] David C. Luckham and James Vera. An event-based architecture definition language. *IEEE Transactions on Software Engineering*, 21(9):717–734, 1995.

[107] Basel Magableh and Stephen Barrett. Primitive component architecture description language. In *Informatics and Systems (INFOS), 2010 The 7th International Conference on*, pages 1–7. IEEE, 2010.

[108] Jeff Magee and Jeff Kramer. Dynamic structure in software architectures. *ACM SIGSOFT Software Engineering Notes*, 21(6):3–14, nov 1996.

[109] Ivano Malavolta, Patricia Lago, Henry Muccini, Patrizio Pelliccione, and Antony Tang. What industry needs from architectural languages: A survey. *IEEE Transactions on Software Engineering*, 39(6):869–891, 2013.

[110] Georg E. Matt and Thomas D. Cook. Threats to the validity of research synthesis. In H Cooper and L.V. Hedges, editors, *The Handbook of Research Synthesis*, pages 503–520. Russell Sage Foundation, New York, New York, USA, 1994.

[111] Genc Mazlami, Jurgen Cito, and Philipp Leitner. Extraction of Microservices from Monolithic Software Architectures. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 524–531, jun 2017.

[112] Andrew McVeigh, Jeff Kramer, and Jeff Magee. Using resemblance to support component reuse and evolution. In *Proceedings of the 2006 conference on Specification and verification of component-based systems*, pages 49–56. ACM, 2006.

[113] Nenad Medvidovic, David S Rosenblum, and Richard N Taylor. A language and environment for architecture-based software development and evolution. In *Proceedings of the 21st international conference on Software engineering*, pages 44–53. ACM, 1999.

[114] Nenad Medvidovic and RN Taylor. A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Transactions on Software Engineering*, 26(1):70–93, 2000.

[115] Hong Mei, Feng Chen, Qianxiang Wang, and Yaodong Feng. ABC/ADL: An ADL supporting component composition. In *International Conference on Formal Engineering Methods*, pages 38–47. Springer, 2002.

[116] Christoph Möbius, Waltenegus Dargie, and Alexander Schill. Power consumption estimation models for processors, virtual machines, and servers. *IEEE Transactions on Parallel and Distributed Systems*, 25(6):1600–1614, 2014.

[117] Parastoo Mohagheghi and Mario Ek Aparicio. An industry experience report on managing product quality requirements in a large organization. *Information and Software Technology*, 2017.

[118] Mubarak Mohammad and Vasu Alagar. TADL-an architecture description language for trustworthy component-based systems. In *European Conference on Software Architecture*, pages 290–297. Springer, 2008.

[119] Daniel Moody. The physics of notations: Toward a scientific basis for constructing visual notations in software engineering. *IEEE Transactions on Software Engineering*, 35(6):756–779, nov 2009.

[120] Mark Moriconi and Robert A. Riemenschneider. Introduction to SADL 1 .0: A Language for Specifying Software Architecture Hierarchies. Technical report, Computer Science Laboratory, SRI International, 1997.

[121] Blake Morrison. Windows Performance Monitor Overview. https://blogs.technet.microsoft.com/askperf/2014/07/17/windows-performance-monitor-overview,
2014.

[122] I. Murwantara, Behzad Bordbar, and Leandro L. Minku. Measuring energy consumption for web service product configuration. In *ACM International Conference Proceeding Series*, pages 224–228, 2014.

[123] Dean Nelson and Raymond Paquet. How eBay's I&O Organization Is Supporting Business Initiatives. In *Gartner Data Center Conference*, 2013.

[124] New Relic Inc. New Relic Documentation. https://docs.newrelic.com, 2018.

[125] Sam Newman. *Building Microservices: designing fine-grained systems*. O'Reilly, 2015.

[126] Nlyte Software. Nlyte Energy Optimizer Product Overview. https://resources.nlyte.com/neo/nlyte-energy-optimizer, 2018.

[127] Adel Noureddine, Aurelien Bourdon, Romain Rouvoy, and Lionel Seinturier. Runtime monitoring of software energy hotspots. In *Automated Software Engineering (ASE), 2012 Proceedings of the 27th IEEE/ACM International Conference on*, pages 160–169, 2012.

[128] Adel Noureddine, Syed Islam, and Rabih Bashroush. Jolinar: analysing the energy footprint of software applications. In *Proceedings of the 25th International*

Symposium on Software Testing and Analysis - ISSTA 2016, pages 445–448, 2016.

[129] Adel Noureddine, Romain Rouvoy, and Lionel Seinturier. A review of energy measurement approaches. *ACM SIGOPS Operating Systems Review*, 47(3):42–49, 2013.

[130] Adel Noureddine, Romain Rouvoy, and Lionel Seinturier. Unit testing of energy consumption of software libraries. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing - SAC '14*, pages 1200–1205, 2014.

[131] Adel Noureddine, Romain Rouvoy, and Lionel Seinturier. Monitoring energy hotspots in software: Energy profiling of software code. *Automated Software Engineering*, 22(3):291–332, 2015.

[132] OpenZipkin Contributors. Zipkin Home Page. https://zipkin.io, 2018.

[133] Flavio Oquendo. Formally Describing Dynamic Software Architectures with π-ADL. *World Scientific and Engineering Transactions on Systems*, 3(8):673–679, jun 2004.

[134] Flavio Oquendo. π-ADL: an Architecture Description Language based on the higher-order typed π-calculus for specifying dynamic and mobile software architectures. *ACM SIGSOFT Software Engineering Notes*, 29(3):1–14, may 2004.

[135] Ipek Ozkaya, Len Bass, Raghvinder S Sangwan, and Robert L Nord. Making practical use of quality attribute information. *IEEE Software*, 25(25-33), 2008.

[136] Mark P. Mills. The cloud begins with coal: Big data, Big Networks, Big infrastructure, and Big power - An Overview of the Electricity use by the Global Digital Ecosystem. *by Digital Power Group*, 2013.

[137] Jennifer Pérez, Isidro Ramos, Javier Jaén, Patricio Letelier, and Elena Navarro. Prisma: Towards quality, aspect oriented and dynamic software architectures. In *Quality Software, 2003. Proceedings. Third International Conference on*, pages 59–66. IEEE, 2003.

[138] James Phung, Young Choon Lee, and Albert Y. Zomaya. Application-Agnostic Power Monitoring in Virtualized Environments. In *Proceedings - 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2017*, pages 335–344, 2017.

[139] Mónica Pinto, Lidia Fuentes, and José M. Troya. Towards an aspect-oriented framework in the design of collaborative virtual environments. In *Proceedings of the IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*. IEEE Comput. Soc, 2001.

[140] Monica Pinto, Lidia Fuentes, and Jose-Maria Troya. DAOP-ADL : An Architecture Description Language for Dynamic Component and Aspect-Based Development. In Frank Pfenning and Yannis Smaragdakis, editors, *2nd international conference on Generative programming and component engineering (GPCE '03)*, pages 118–137, Erfurt, Germany, 2003. Springer-Verlag.

[141] Pascal Poizat and Jean-Claude Royer. A Formal Architectural Description Language based on Symbolic Transition Systems and Modal Logic. *Journal of Universal Computer Science*, 12(12):1741–1782, 2006.

[142] Eltjo R. Poort and Hans Van Vliet. RCDA: Architecting as a risk- and cost management discipline. In *Journal of Systems and Software*, 2012.

[143] Giuseppe Procaccianti, Stefano Bevini, and Patricia Lago. Energy Efficiency in Cloud Software Architectures. *27th International Conference on Informatics for Environmental Protection*, 2013.

[144] Giuseppe Procaccianti, Héctor Fernández, and Patricia Lago. Empirical evaluation of two best practices for energy-efficient software development. *Journal of Systems and Software*, 117:185–198, 2016.

[145] Giuseppe Procaccianti, Patricia Lago, and Stefano Bevini. A systematic literature review on energy efficiency in cloud software architectures. *Sustainable Computing: Informatics and Systems*, 7:2–10, 2015.

[146] Giuseppe Procaccianti, Patricia Lago, and Grace A. Lewis. A catalogue of green architectural tactics for the cloud. In *Proceedings - 2014 IEEE 8th International*

Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems, MESOCA 2014, 2014.

[147] Giuseppe Procaccianti, Patricia Lago, and Grace A Lewis. Green Architectural Tactics for the Cloud. In *Software Architecture (WICSA), 2014 IEEE/IFIP Conference on*, pages 41–44. IEEE, 2014.

[148] Prometheus Authors. Prometheus Home Page. https://prometheus.io, 2018.

[149] Alexandre Rademaker, Christiano Braga, and Alexandre Sztajnberg. A Rewriting Semantics for a Software Architecture Description Language. *Electronic Notes in Theoretical Computer Science*, 130:345–377, may 2005.

[150] Red Hat Inc. Rkt Home Page. https://coreos.com/rkt, 2018.

[151] Chris Richardson. Patterns of Microservice Architecture. http://microservices.io/patterns/microservices.html, 2018.

[152] Dan Rogers and Ulrich Homann. Application Patterns for Green IT. *Microsoft Architecture Journal*, 18(1):16–21, 2008.

[153] Nick Rozanski and Eoin Woods. *Software Systems Architecture: working with stakeholders using viewpoints and perspectives*. Addison Wesley, 2 edition, 2011.

[154] Davide Di Ruscio and Ivano Malavolta. Developing next generation ADLs through MDE techniques. In Jeff Kramer and Judith Bishop, editors, *32nd ACM/IEEE International Conference on Software Engineering*, pages 85–94, Cape Town, South Africa, 2010. ACM.

[155] SAE International. Architecture Analysis & Design Language (AADL). Technical report, Society of Automotive Engineers, 2009.

[156] Chiyoung Seo, George Edwards, Sam Malek, and Nenad Medvidovic. A framework for estimating the impact of a distributed software system's architectural style on its energy consumption. In *7th IEEE/IFIP Working Conference on Software Architecture, WICSA 2008*, 2008.

[157] Steven Shapiro. Myths and Realities About Designing High Availability Data Centers, 2015.

[158] Mary Shaw, Robert DeLine, and Gregory Zelesnik. Abstractions and implementations for architectural connections. *IEEE International Workshop on Configurable Distributed Systems -Proceedings*, pages 2–10, 1996.

[159] Mary Shaw and David Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

[160] Benjamin H Sigelman, Luiz André Barroso, Mike Burrows, Pat Stephenson, Manoj Plakal, Donald Beaver, Saul Jaspan, and Chandan Shanbhag. Dapper, a Large-Scale Distributed Systems Tracing Infrastructure. Technical report, Google, Inc., 2010.

[161] Vivek Kumar Singh, Kaushik Dutta, and Debra VanderMeer. Estimating the energy consumption of executing software processes. In *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*, pages 94–101. IEEE, 2013.

[162] Karl Smolander, Kalle Lyydnen, Veli-Pekka Tahvanalnen, and Pentti Marttiin. MetaEdit - A Flexible Graphical Environment for Methodology Modelling. In *International Conference on Advanced Information Systems Engineering*, pages 168–193. Springer Berlin Heidelberg, 1991.

[163] Jonathan Stoikovitch. RESTful API Modeling Language, 2018.

[164] Guoxin Su, Mingsheng Ying, and Chengqi Zhang. An ADL-approach to specifying and analyzing centralized-mode architectural connection. In *European Conference on Software Architecture*, pages 8–23. Springer, 2010.

[165] Sunbird Software. Power IQ DCIM Monitoring Software. https://www.sunbirddcim.com/sites/default/files/DS007_Sunbird_DataSheet_PowerIQ6_2.pdf, 2018.

[166] Davy Suvée, Bruno De Fraine, and Wim Vanderperren. FuseJ: An architectural description language for unifying aspects and components. In *Software-engineering Properties of Languages and Aspect Technologies Workshop@ AOSD2005*. Citeseer, 2005.

[167] Richard Berntsson Svensson, Tony Gorschek, Björn Regnell, Richard Torkar, Ali Shahrokni, Robert Feldt, and Aybuke Aurum. Prioritization of quality requirements: State of practice in eleven companies. In *Proceedings of the 2011 IEEE 19th International Requirements Engineering Conference, RE 2011*, 2011.

[168] Swagger Contributors. OpenAPI Specification. https://swagger.io/specification, 2018.

[169] Technical Committee ISO/IEC JTC 1/SC 7 Software and systems engineering. ISO/IEC 42010 Systems and Software Engineering-Recommended Practice for Architectural Description of Software-Intensive Systems, 2011.

[170] TechUK. Data Centres and Power: Fact or Fiction? Technical report, Information Technology Telecommunications And Electronics Association, 2013.

[171] The Green Grid. Recommendations for Measuring and Reporting Overall Data Centre Efficiency Version 2 - Measuring PUE for Data Centres. Technical report, The Green Grid, 2011.

[172] Naoyasu Ubayashi, Jun Nomura, and Tetsuo Tamai. Archface: a contract place where architectural design and code meet together. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 75–84. ACM, 2010.

[173] Uwe van Heesch and Paris Avgeriou. Mature architecting - A survey about the reasoning process of professional architects. In *Proceedings - 9th Working IEEE/IFIP Conference on Software Architecture, WICSA 2011*, 2011.

[174] Robert. van Ommering, Frank. van der Linden, Jeff. Kramer, and Jeff. Magee. The Koala component model for consumer electronics software. *Computer*, 33(3):78–85, mar 2000.

[175] Quan Yu Wang, Shi Ying, Xiang Yang Jia, Guo Bin Lv, and Yun Shuai. SOADL-EH: Service-Oriented Architecture Description Language Supporting Exception Handling. In *Advanced Materials Research*, volume 433, pages 3500–3509. Trans Tech Publ, 2012.

[176] Zhuxiao Wang, Hui Peng, Jing Guo, Ying Zhang, Kehe Wu, Huan Xu, and Xiaofeng Wang. An architecture description language based on dynamic description logics. In *International Conference on Intelligent Information Processing*, pages 157–166. Springer, 2012.

[177] Roel Wieringa and AyÅ§e Morali. Technical Action Research as a Validation Method in Information Systems Design Science. In *Design Science Research in Information Systems. Advances in Theory and Practice 7th International Conference, DESRIST 2012, Las Vegas, USA*, pages 220–238, 2012.

[178] Wikipedia. Situational Awareness. https://en.wikipedia.org/wiki/Situation_awareness, 2018.

[179] Wikipedia Authors. Wikipedia - Microservices. https://en.wikipedia.org/wiki/Microservices, 2018.

[180] Eoin Woods and Rabih Bashroush. Using an architecture description language to model a large-scale information system - An industrial experience report. In *Proceedings of the 2012 Joint Working Conference on Software Architecture and 6th European Conference on Software Architecture, WICSA/ECSA 2012*, pages 239–243, 2012.

[181] Eoin Woods and Rabih Bashroush. Modelling large-scale information systems using ADLs - An industrial experience report. *Journal of Systems and Software*, 99:97–108, 2015.

[182] Eoin Woods and Rabih Bashroush. A Model for Prioritization of Software Architecture Effort. In *European Conference on Software Architecture*, pages 183–190. Springer, 2017.

[183] Eoin Woods and Rabih Bashroush. A model for prioritization of software architecture effort. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 10475 LNCS, pages 183–190, 2017.

[184] Eoin Woods and Rich Hilliard. Architecture Description Languages in Practice Session Report. In Robert Nord, Nenad Medvidovic, René Krikhaar, Judith

Stafford, and Jan Bosch, editors, *5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*, pages 243–246, Pittsburgh, PA, USA, 2005. IEEE.

[185] Eoin Woods and Nick Rozanski. Using Architectural Perspectives. In Robert Nord, Nenad Medvidovic, René Krikhaar, Judith Stafford, and Jan Bosch, editors, *5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*, pages 25–35, Pittsburgh, PA, USA, 2005. IEEE.

[186] Mark Ylvisaker, Mary Hibbard, and Timothy Feeney. Tutorial: Concrete vs. Abstract Thinking. Technical report, The Brain Injury Association of New York State, 2006.

[187] Tomofumi Yuki and Sanjay Rajopadhye. Folklore confirmed: Compiling for speed = Compiling for energy. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pages 169 – 184, 2014.

[188] Guang-quan Zhang, Mei Rong, and Hui Wei. Description and Analysis for Web Service Composition Based on XYZ/ADL. In *2009 WRI World Congress on Software Engineering (WCSE09)*, pages 185–188, Xiamen, 2009. IEEE.

[189] Tao Zhang and Haipeng Wang. vADL: A Variability-Supported Architecture Description Language for Specifying Product Line Architectures. In *2nd International Software Product Lines Young Researchers Workshop (SPLYR 2005)*, 2005.

[190] Li Zheng, Zhanwei Wu, Chao Zhang, and Fang Yang. Developing an Architecture Description Language for Data Processing Product Line. In *2010 Seventh International Conference on Information Technology*, pages 944–949. IEEE, 2010.