Using Architectural Perspectives

Eoin Woods
Zuhlke Engineering Limited
49 Great Cumberland Place
London W1H 7TH
eoin.woods@zuhlke.com

Wolfgang Emmerich
University College London
Gower Street
London WC1E 6BT
w.emmerich@cs.ucl.ac.uk

Nick Rozanski
Marks and Spencer Limited
Stockley Park
Uxbridge UB11 1AW
nick@rozanski.com

Abstract

Many projects that aim to develop embedded or distributed systems need to address quality requirements, such as performance, reliability and security. These quality requirements are often not directly attributable to individual components of the system but rather need to be achieved by the overall software architecture. Designing the right software architecture for a system is an inherently difficult problem. We have developed the notion of architectural perspectives that assist software architects in addressing such quality requirements by guiding them with checklists, suggestions for activities and indications of pitfalls towards choosing the right software architecture. In [19], we provide a large number of architectural perspectives that assist, amongst others, in addressing performance, scalability and security requirements. This paper gives an account of the experience we gained when applying architectural perspectives for defining the enterprise application integration architecture in a financial services firm.

1. Introduction

Designing a software architecture is a complex process, involving the creation of solutions to complex, multi-faceted problems, that often do not have a single optimal solution, but only a number of acceptable ones. One particularly difficult aspect of the architectual process is ensuring that a system will meet its quality requirements (for security, performance, availability and so on).

Most architects would agree that the quality properties their systems exhibit are crucial to meeting the needs of the system's stakeholders. Systems that are too slow or that are unavailable at critical moments simply are not used; systems that prove to be insecure are quickly abandoned or never make it to production; systems that cannot be changed easily quickly become obsolete. While getting a system's functionality correct is obviously important, this point is moot for many systems that are considered to be failures because they are lacking in one or more critical non-functional qualities.

Most software architects use an intuitive approach to achieving quality properties, relying on a combination of instinct, background knowledge and experience to guide them through the design process. This intuition led process often works well, as the number of useful, effective large-scale computer systems attests to; indeed, we used to work in this intuitive way, which served us reasonably well for a long time. However we have found that it has its limitations. In particular:

- It is difficult to share knowledge between architects to allow successful approaches to be reused and painful lessons avoided;
- Few architects can honestly claim deep expertise across all of the possible quality property areas that they have to work with and so there is always a danger of focusing on an arbitrary set of properties because these are the ones that the architect knows about; and
- The lack of a systematic approach increases the risk that something important will be overlooked until it is too late to address it.

It was a realisation of these limitations that led us to try to create a simple, yet systematic approach to guide architectural design for quality properties. The approach is based on our experience working as practising software architects, across a wide variety of types of information systems, for a combined total of about 45 years of professional software development practice.

We call our approach *Architectural Perspectives* and it provides a framework for structuring knowledge about how to design systems to achieve particular quality properties. In many ways, Perspectives are similar

to Architectural Viewpoints [10], but whereas a particular Viewpoint conventionally advises on how to create and describe a particular type of architectural *structure* [10, 14, 9, 18], a Perspective relates to the cross view concerns of a particular architectural *quality property*.

We believe that Architectural Perspectives are a novel approach because

- they are a knowledge sharing framework structured around quality properties, as opposed to types of architectural structure;
- the approach does not mandate any particular architectural structure or style;
- perspectives work well when combined with Viewpoints and so neatly extend an already proven approach [20];
- the approach is the product of practitioner experience and so is addressing a real need that practitioners have; and
- The approach has proven to be useful in practice.

The remainder of this paper presents the Perspectives approach and illustrates its use by means of a real project example. Section 2 compares the approach to related work; Section 3 explains the approach and how to use it; Section 4 describes an application of the approach to an information systems development project; Section 5 outlines the strengths and weaknesses of the approach; Section 6 explains the lessons learned while developing the approach; and Section 7 summarises the paper and presents our conclusions.

2. Related Work

The notion of a viewpoint has been used to a considerable extent, sometimes with slightly different meanings. Finkelstein et al, who have coined the term viewpoints, suggested them as a conceptual framework to define templates whose instances describe software systems from different perspectives [8, 17]. Our use of perspectives is similar in that we define a template for addressing different quality properties. However, instances of our templates then define how particular perspectives need to be incorporated into a suitable architectural description. The reference model included in the ISO Open Distributed Processing standard [12] suggests the use of five different viewpoints to capture distributed systems. Kruchten [14] suggests the use of 4+1 views to define software architectures. Again both of these are rather static approaches that do not emphasise the process of arriving at an architecture nor do they focus on reusing architectural knowledge.

There has been a large body of work on architecture description languages (ADLs), such as Darwin [16], Wright [2], Rapide [15] and CHAM [11] to name but a few. These languages support the definition of components, their interfaces and sometimes the association of behaviour to component interconnection. ADLs, however do not provide any guidance on how to define an architecture in order to address a particular quality property. Our work is more closely related architectural styles [1] in that these too provide reusable descriptions of architectures. In practice, however, we found architectural styles to be of limited use; often different perspectives need to be combined to arrive at a suitable architecture and this process of combining perspectives is not addressed by architectural styles.

Our work also exhibits conceptual similarities to aspect-oriented programming [13]. Architectural perspectives are cross-cutting concerns that need to be woven into an architecture. The majority of the work on aspects though concerns itself with formal language primitives that support the automatic weaving of concerns into code. Our work on architectural perspectives instead assumes that architectural perspectives are described relatively informally and are woven manually by the software architect into the target software architecture.

Architectural perspectives are complementary to the growing body of work in the area of software architecture evaluation, characterised by evaluation methods like ATAM [7]. While formal evaluation methods like ATAM allow an architecture to be evaluated for suitability with respect to stakeholder defined goals, architectural perspectives guide the architect through the process of achieving these desired quality properties, to allow a suitable candidate for evaluation to be produced.

Architectural perspectives are also closely related to architectural tactics [4], embracing and extending this work, by providing advice relating to what the architect should know, do and be aware of as well as specific solution tactics.

3. Description of the Approach

3.1 Defining Architectural Perspectives

We developed the concept of the Architectural Perspective (or just "Perspective") in order to provide an extensible framework, within which we could capture knowledge about designing systems that need to exhibit specific quality properties. From the outset, we aimed to develop an approach that could be used with

existing viewpoint-based and architectural evaluation approaches.

Our definition of Architectural Perspective is *a collection of activities, checklists, tactics and guidelines to guide the process of ensuring that a system exhibits a particular set of closely related quality properties that require consideration across a number of the system's architectural views.* In other words, a Perspective is a collection of guidance on achieving a particular quality property in a system.

This wording and structure of the definition is similar to that used for the definition of an architectural viewpoint in IEEE standard 1471 [10]. This similarity is intentional, as it is meant to suggest, that Perspectives are analogous to Viewpoints (in the 1471 sense of the term) but rather than addressing an aspect of the system's structure the Perspective addresses an important quality property.

This close analogy between Viewpoints and Perspectives is intentional as it makes it easy to relate the two concepts and use them together, with Perspectives acting as an extension to an existing viewpoint-based approach.

A Perspective has a standard suggested structure, to make the use of sets of Perspectives easier and to ensure that they all address a quality property in the same general way. A Perspective contains the following information:

- the *Concerns* that the perspective is addressing;
- the *Applicability* of the perspective to the different possible architectural views of a system (and the types of system to which the advice within it relates, if this is not obvious);
- a set of possible *Activities* that are suggested as part of the process of achieving the quality property (ideally related to each other via a process to follow);
- a set of proven *Architectural Tactics* (i.e. design strategies) [4] that the architect can consider as part of their design;
- a list of common *Problems and Pitfalls* that the architect should be aware of and common solutions to them; and finally
- a *Checklist* that the architect can use to ensure that nothing has been forgotten.

As an example, consider what might be in a Security Perspective, to guide an architect in achieving a secure system.

The *concerns* for the Security Perspective would include:

- Policy (the actions that difference principals can perform on sensitive resources);
- Threats (the security threats that the system faces);
- Governance (the mechanisms for implementing the policy securely, including authentication, authorisation, confidentiality, integrity and accountability);
- Availability (ensuring that attackers cannot prevent access to a system); and
- Detection and Recovery from Breach (allowing recovery when security fails).

The Security Perspective is particularly *applicable* to the Physical and Development architectural views (in "4+1" terminology), adding security related hardware and software to the Physical view and setting system wide security related standards within the Development view. Changes could also be required to the Logical view to support a secure implementation (e.g. partitioning the system differently to allow access to be controlled to sensitive parts of it).

The *activities* defined in the Security Perspective would include:

- Identification of sensitive resources;
- Definition of a security policy;
- Creation of a threat model;
- Design of a security implementation; and
- Assessment of security risks.

The activities in the Security Perspective would be inter-related by use of a process description (such as a UML activity diagram) like the one in Figure 1.

The primary *architectural tactics* that the Security Perspective would explain and suggest would include:

- Application of recognised security principles (least privilege, separation of responsibilities, simplicity, auditing, secure default behaviour, not relying on obscurity and so on);
- Principal identification mechanisms;
- Access control mechanisms;
- Information protection mechanisms;

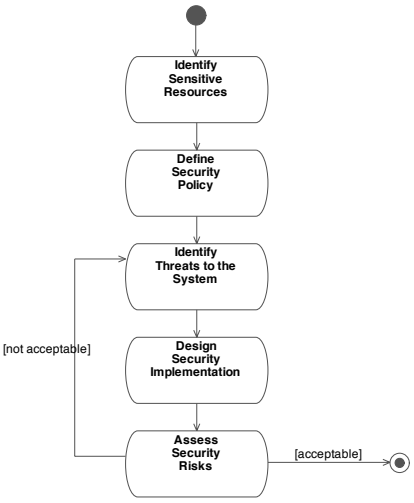

Figure 1. Security Perspective Process

- How to ensure accountability via auditing and non-repudiation mechanisms;
- How to protect availability with hardware and software system protection mechanisms;
- Integration approaches for existing technology;
- Provision of security administration; and
- Use of 3rd party security technology.

The common *problems and pitfalls* that the Security Perspective would list (and provide common solutions for) would include:

- Complex security policies;
- Use of unproven security technology;
- Not designing for secure failure conditions;
- Not providing effective administration facilities;
- Driving the process by technology choice rather than security threats;
- Ignoring the need for secure time sources;
- Leaving security as an afterthought;
- Embedding security policy in the application; and
- Use of ad-hoc technology to enforce security.

The Security Perspective would also include a *checklist* containing points such as:

- Is there a clear security policy that defines which principals are allowed to perform which operations on which resources?
- Is the security policy as simple as possible?
- Have security requirements been reviewed with external experts?
- Has each threat identified in the threat model been addressed to the extent necessary?

Space prevents us from presenting the entire Perspective, and it should be stressed that the above presentation is only an outline, as our real Security Perspective is 20 pages long, but hopefully this gives a flavour of the content that a perspective contains.

There are a large number of potential Perspectives that could be written and the set that will be of use to an architect depends very much on the type of system that they are working on: an architect working on vehicle control systems is unlikely to use the same set of Perspectives as an architect working on a credit card billing system. Indeed, it is important that Perspectives are written for a specific target audience so that inappropriate advice is not included in them. That said, as with viewpoints, we think it is likely that useful Perspectives can be written for certain broad system types.

For large scale information systems in particular (which is the type of system that all of the authors work with) we have found a good core set of Perspectives to be:

- *Security* to ensure the ability of owners of resources in the system to reliably control, monitor and audit who can perform what actions on these resources as well as the ability of the system to detect and recover from failures in security mechanisms;
- *Performance and Scalability* to ensure the system's ability to predictably execute within its mandated performance profile and to handle increasing processing volumes;
- *Availability and Resilience* to ensure the system's ability to be fully or partly operational as and when required, and to effectively handle failures which could affect system availability; and
- *Evolution* ensuring system flexibility in the face of the inevitable change that all systems experience after deployment, balanced against the costs of providing such flexibility.

Other Perspectives that we have found applicable to many information systems, but that are less widely applicable than the core set suggested above, include:

- *Internationalisation* to ensure the systems independence from any particular language, country or cultural group;
- *Accessibility* to ensure the ability of the system to be used by people with disabilities;
- *Usability* to ensure that people who interact with the system can easily work effectively;
- *Regulation* to ensure the ability of the system to comply with local and international laws, quasi-legal regulations, company policies, and other rules and standards;
- *Location* to ensure the ability of the system to overcome problems brought about by the absolute geographical location of its elements and the distances between them; and
- *Development Resource* to ensure that the system can be designed, built, deployed and operated within known constraints around people, budget, time and materials.

Based on our experience as architects of large information systems, we have developed full definitions of the first four Perspectives listed above, as well as outline definitions of the remainder. The definitions are presented in the form of a forthcoming book [19], aimed at practising software architects and those in training for the role.

3.2 Using Perspectives

We have found that a set of Perspectives can play three distinct roles for a software architect.

Firstly, Perspectives act as a *store of knowledge*, allowing knowledge related to achieving a particular quality property to be gathered and represented in a standardised way, so making it easy for the architect to use them to extend their knowledge. It is important to note that the Perspective is a much more flexible and much less constrained source of knowledge than a design pattern or an attribute based architectural style. A perspective documents things the architect should know and do as well as simply a set of technical solutions (although it can include these too, in the Architectural Tactics section of the Perspective).

Secondly, Perspectives act as a *guide* to a novice architect or an architect having to deal with a quality property that they are not an expert in (a situation

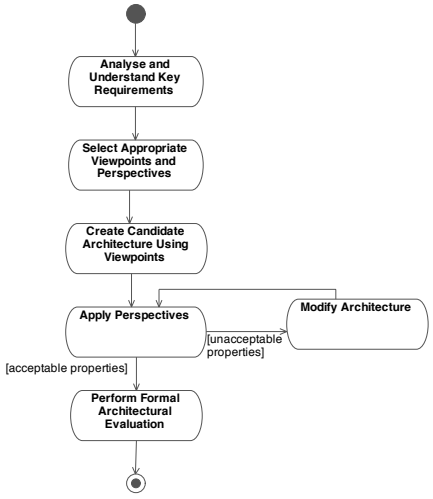

Figure 2. Using Perspectives

that many architects meet routinely, even if they do not always feel that they can admit it!) The information in the Perspective allows the architect to quickly learn what is important about achieving the particular quality property under consideration, provides them with a set of proven activities and tactics to use and points out the likely problems that will be encountered.

Finally, Perspectives act as an *aide memoir* for the experienced architect working in an area that they are familiar with. However, even in such cases, it is very valuable to have standardised reference material that can be quickly and conveniently accessed. When used in this way, Perspectives help the architect to work in a systematic manner (in as much as they need to) and help to avoid important details being overlooked.

The process of using a set of Perspectives within a viewpoint-based architectural design process is illustrated by the UML activity diagram in Figure 2.

As can be seen in the diagram, the architect starts by understanding the key system requirements, which allows him to select the appropriate set of viewpoints and perspectives to use to guide the architectural design process. Next, he produces a potential architectural design to meet the system's key requirements, at this stage focusing primarily on the system's functional structure. Then, for each important quality property, the architect uses the information in the corresponding Perspective to drive the process of ensuring that the system will exhibit that quality property satisfactorily. In most cases, this will mean changes to the architecture, which are reflected by updating the views describing the architecture. Then, when the architect believes that they have a satisfactory architecture, it

can go forward for formal architectural evaluation, using a method like ATAM [7].

Obviously this description is a very idealised view of the process, but it provides a useful mental model for the architect and we have found it useful when explaining Perspectives to other architects. In reality of course, the experienced architect is considering quality properties continually, from the beginning of the design process and so is using Viewpoints and Perspectives simultaneously, rather than in two distinct stages.

We term the process of using a Perspective "*applying the perspective*" to stress that the process of using a Perspective is primarily about making cross-view changes to the architecture, rather than about creating a new architectural design artefact. (This said, applying many perspectives can actually produce outputs such as threat models, performance models and so on, but these are really supporting information rather than first-class architectural design artefacts.) We also use this term to reinforce the point that using a Perspective is not just a review process but is an active part of architectural design, performed by the architect in order to produce an acceptable architecture.

It is worth noting that the process presented here is quite similar to the architectural design process that Jan Bosch defines in his book [5]. While Bosch explains that the Architect must modify the architecture in order to achieve the system's desired quality properties, there is no specific guidance on how to go about this. The contribution that Perspectives make to the process is providing structure and specific advice on how to achieve the quality properties that the system is lacking.

4. An Example Application of Perspectives

4.1 The Project

Like many organisations, a UK-based financial institution had ended up with a large number of business applications, many of which needed the same reference information in order to perform their processing. The types of information that needed to be shared between systems included details of counterparties, countries, financial exchanges, terms and conditions for financial instruments, closing prices for financial products, holdings of financial products and so on. This information is characterised by changing relatively slowly (at least for the uses that the systems this project was concerned with put it to) with a daily or hourly update being sufficient. However, when the information is duplicated and maintained across a number of systems then inconsistency nearly always occurs, maintaining the data becomes very difficult and errors occur in business processing. Where inter-system data integration had been implemented, it had been done in a tactical "point to point" manner, which had resulted in an inflexible structure with many inter-system dependencies.

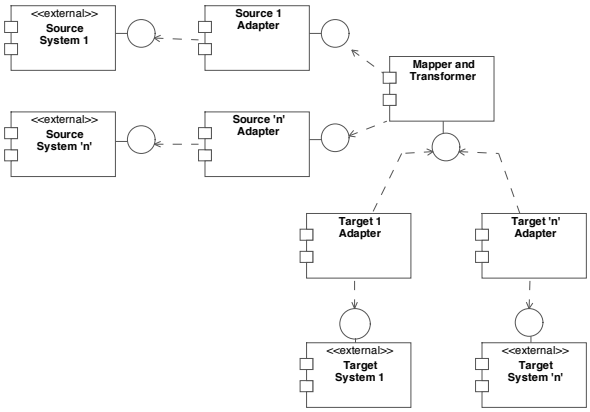

Figure 3. Data Service Functional Structure

The solution identified for these problems was to create an organisation-wide "Data Service" that could provide reference information to any of the organisation's systems on a regular schedule, in the format that the target system required. An important benefit of the Data Service is that it acts to totally decouple the systems supplying the data (the sources) from the systems consuming it (the targets), without changing either.

The initial implementation of the system was batch based, distributing data to the target systems according to a regular schedule. The system was implemented using Java and XML-based technologies, with all of the data manipulation required being implemented using XSLT [6], in order to isolate the data mapping in well understood places and to allow it to be reused in other possible future implementations.

When the system runs, it extracts data from a number of source systems using existing data-access interfaces, converts it into a system-neutral organisation-wide data model and then supplies the subsets of the data required by each target system to these systems in their native formats. The functional structure of the system is illustrated by the UML component diagram in Figure 3.

The two key quality properties that this system had to exhibit were:

- *Performance*, in terms of throughput, because the system needed to handle a reasonably large

amount of data (several hundred Mb of raw data) in a very limited amount of time; and

- *Evolution*, in terms of adding sources, targets and data types easily, because without the ability to add new data sources, target systems and data types the system would rapidly become obsolete as the business evolved.

4.2 The Use of Perspectives

A number of Perspectives were used on the project, but due to space limitations, we will describe how one particular Perspective – *Performance and Scalability* – was applied and the effect that this had.

Although the project was relatively simple, it was a critical system for the organisation, it was the first attempt to apply the implementation technology in that organisation and it had to meet quite stringent quality properties in order to be considered a success. For these reasons, we found it involved enough to make a useful case study for this paper.

Another interesting feature of this project was that there were actually two architects, the architect responsible for the project, who worked for the financial institution ("the Architect") and one of the authors who acted as a consultant to the organisation, advising on the work to be performed, working full time in the project team and mentoring the organisation's architect during the project ("the Mentor"). This feature of the project means that it also illustrates the different uses that different architects can put Perspectives to.

Due to the mentoring aspect of this project, Perspectives were applied in a relatively simplistic way, with the Architect being encouraged to understand the system's functional requirements thoroughly and design a sound functional structure before focusing on achieving particular quality properties. Once a candidate functional structure was identified, the Perspectives were used by the Architect and the Mentor to refine it to meet the critical performance and evolution qualities. Perspectives were a useful aid to process structuring, as they encouraged the Architect to work in a systematic manner and provided the Mentor with a metaphor to use when explaining the process to the Architect.

The Performance and Scalability Perspective defines the relevant concerns as being response time, throughput, scalability, predictability, hardware resource requirements and peak load behavior. Using these concerns at the start of the process helped the Architect to understand what was included (and excluded) from the performance exercise and allowed context and clear objectives for the exercise to be defined.

The activities the Perspective suggests are capturing performance requirements, creating and analysing performance models and performance testing. The documentation of these activities in the Perspective provided the Architect with background information on performance engineering (which he was not aware of before) and acted as useful reference material to start learning about them. Having said that, most of the knowledge transfer was driven by the Mentor, referring to the Perspective as needed.

A concrete result of following the process suggested in the Perspective was the creation of a performance model for the system. While the Perspective did not contain enough detail for the Architect to do this totally independently, the information in the Perspective did help him to understand what he was doing and why and so provided context for the additional information and guidance provided interactively by the Mentor.

Once a performance model and some representative performance testing had been completed, it was established that the system was likely to run acceptably fast and to complete its processing within the processing schedule required of it. However, the exercise did reveal two important points. Firstly, 70% of the systems runtime was consumed by the processing of two business entities (out of a total of about 20) and secondly, the execution time of the system was uncomfortably close to its acceptable limit, considering that data volumes were likely to increase in the future.

The insights gained by the performance modelling and testing were valuable for two reasons. Firstly, they allowed the Architect to set realistic expectations for the throughput that could be achieved and secondly, they revealed the need for contingency planning at the architectural design level in case of slower throughput than expected or an unexpected increase in data volumes.

The Architectural Tactics section of the Perspective was used to consider possible design changes to increase throughput, so that allowance could be made in the architecture for their possible future implementation. The tactics in the Perspective included optimizing common processing, decomposition and parallelization of long operations, reducing contention via replication, prioritizing processing, consolidation of related workload, distribution of processing in time, minimizing the use of shared resources and considering the use of asynchronous processing. Particularly valuable tactics in this particular situation were parallelization, prioritizing processing and distributing processing in time. Possible approaches for each were sketched to ensure that the proposed architecture was compatible with them.

Finally, the problems and pitfalls contained in the Perspective were used to check that nothing important had been overlooked. The problems and pitfalls documented in the Perspective include having imprecise performance and scalability goals, an over-reliance on modeling, using simple measures for complex cases, inappropriate partitioning, invalid environment and platform assumptions, too much indirection, concurrency related contention, careless allocation of resources and ignoring network and in-process invocation differences. While no serious problems were found, having reviewed the list, we did decide that we had relied too heavily on modelling (over testing) and that some of our testing was assuming that results from simple cases scaled linearly for more complex cases. Both of these possible problems caused us to revisit some of our performance work and in fact, we did find that we had made some invalid assumptions about XML processing performance as document size increases.

Specific results of applying the Performance and Scalability Perspective in this project were:

- a systematic approach being adopted to achieving performance goals;
- the system's Architect gaining a rapid understanding of the process to use to ensure acceptable performance (including concerns, techniques, tactics and pitfalls);
- the creation of a performance model and supporting performance tests;
- an early understanding of the likely performance that could be gained and the risks that this implied;
- the identification of possible future solutions to likely performance problems; and
- several potential problems being noted and rectified during the process.

In summary, using this Perspective reduced performance related risks on the project significantly and encouraged a systematic approach to achieving the required goals.

5. Strengths and Weaknesses of the Approach

The main strengths of Architectural Perspectives that have been found in this and other projects are described below.

- Perspectives provide a framework for organising and using knowledge, which is often a major challenge for software architects, given the breadth of the role.
- Using Perspectives helps an architect to work in a systematic way to ensure that certain key quality properties are exhibited by their system, so helping to organise the work and ensure that nothing is forgotten.
- Perspectives encourage architects to share and reuse knowledge about achieving quality properties.
- We have found Perspectives to be useful to both novice and experienced architects alike, due to the different ways that they can be used. Indeed, we wrote the set of Perspectives outlined above and routinely use them ourselves in our own work and well as when mentoring other architects.
- The approach works well with an architectural design process that is using Viewpoints and a quality-property-centric evaluation approach such as ATAM. This means that the approach fits well with the current state of the art in software architecture practice.
- The approach is very simple, can be explained in a few minutes and we have found that people understand it very quickly.
- The approach does not dictate a particular style or structure for the architecture and so can be used with many types of system.
- Perspectives are the result of practitioner experience and solve a real problem that we had.

Like any approach, there are also weaknesses with Perspectives, the more important of which are described below.

- Each Perspective addresses a single quality property, which means that for any complex system the architect has to apply a number of them and there is no guarantee that the advice in each will be compatible. Indeed, you would expect the advice in a number of them to conflict (between performance and flexibility concerns in different Perspectives for example) and the architect needs to resolve these conflicts when they arise.
- The approach does not help the architect to make the right decisions for their particular stakeholders and this is still a difficult, risky, but key part of the architect's role.

- The approach does not help the architect to select the right set of Perspectives to apply, as this is totally dependent on the needs of their particular system and so this is still a matter of the architect's skill and judgment.
- The Perspectives just contain written advice (rather than any sort of automated assistance such as that provided by research tools like ArchE [3]) and the process of applying a Perspective is still a skilled job that relies entirely upon the architect's abilities.

6. Lessons Learned

The primary lessons that we have learned as a result of our work with Perspectives are summarised below.

6.1. Viewpoints and Quality Properties

We have found the viewpoint-oriented approach very valuable for organising the software architecture process. However, we have found it limited when considering how a system should be designed to meet particular quality properties. The fundamental problem we have found is that viewpoints are typically oriented around a particular type of architectural structure (concurrency, information, modules) whereas achieving a quality property nearly always requires the consideration of cross-cutting concerns that cross a number of structural dimensions. A number of people have disagreed with us verbally on this point, with the core of the argument typically being that you can create any viewpoint you like (for example a "Security" viewpoint) and so our argument is moot. Our experience suggests that creating quality property based viewpoints is not an effective approach, and in particular, when we have tried to create quality property specific views of a system, we have found that a significant problem is the amount of redundancy introduced into the architectural model – redundancy that often results in the architectural description being abandoned as it is too hard to maintain. Intuitively, we also find a separation between design advice for structures and qualities useful in organising both information and the architectural design process, particularly for novice architects.

6.2. The Value of Structure

When using Perspectives, for ourselves and with customers, we have continually been struck by how useful people find the simple and immediately understandable structure that both Viewpoints and Perspectives implicitly impose on the architectural design process. Having the process structured around a set of Viewpoints and Perspectives seems to help people to understand the process and organise their work within it. This structuring is particularly valuable in the architectural design process as it is characterised by a large number of important factors that all need to be considered simultaneously, making it difficult to organise effectively.

6.3. The Importance of Simplicity of Approach

An important strength that we see in both Viewpoints and Perspectives is the simplicity of the approaches. The basics of both approaches can be explained with the help of a whiteboard in 10 or 15 minutes and we have found it rare for people not to understand the approaches within this time. We feel that this indicates that the approaches are fairly intuitive and they seem to reflect an idealisation of the way that people work (or at least think they work). The importance of this simplicity is hard to underestimate, as it helps both adoption of the technique by practising architects and the willingness of their managers to pay for its adoption.

6.4. Sharing Architectural Knowledge is Valuable

When software architects meet, there is usually a discussion of the architectural challenges that the architects are dealing with at the time and it is usually the case that the challenges are similar. Over time, most architects develop a set of standard solutions to problems that they encounter and build up background knowledge that tells them what to focus on and what to avoid in common design situations. However, this sort of personal knowledge base takes a long time and a lot of specific experience to develop. In many cases, sharing architectural knowledge can help to circumvent this learning process and both Viewpoints and Perspectives can fulfil the role of the knowledge source to help make this a reality.

6.5. Making Architectural Tradeoffs is Difficult

One of the limitations of the Perspectives approach is that it only deals with a single quality property at a time; this is intentional and is meant to keep the approach simple and usable. However, it does mean that the architect still has to make the tradeoffs between the demands of different quality properties. Given how system specific the set of tradeoffs required normally is and how dependent it is on the needs of a particular

collection of stakeholders, we are not particularly optimistic that this problem will be solved by a generally applicable approach in the near future. We view it simply as one of the taxing but fascinating parts of the architect's role.

7. Summary and Conclusions

We have introduced an approach to capturing, managing, using and sharing architectural knowledge for achieving quality properties, that we term Architectural Perspectives. Like architectural viewpoints, the approach provides a standardised framework to capturing architectural knowledge, but rather than being organised around types of architectural structure, it is organised around the desired quality properties of the system being designed. We have found that the approach works well in practice and is compatible with existing architectural approaches including architectural evaluation and architectural viewpoints.

Based on our experiences with the approach, we would suggest that it can be a useful tool to encourage the sharing of architectural knowledge between both experienced and novice architects, although it does not fundamentally alter the complex process of inter-quality trade off that is at the core of the architectural decision making process. However, as more perspectives are developed, to address quality properties for different types of systems, we feel that the approach will become widely applicable to the problems that software architects face in their work.

References

[1] G. Abowd, R. Allen, and D. Garlan. Formalizing Style to Understand Descriptions of Software Architecture. *ACM Transactions on Software Engineering and Methodology*, 4(4):319–364, Oct. 1995.

[2] R. Allen and D. Garlan. A Formal Basis for Architectural Connection. *ACM Transactions on Software Engineering and Methodology*, 6(3):213–249, June 1997.

[3] F. Bachmann, L. Bass, and M. Klein. Preliminary Design of ArchE: A Software Architecture Design Assistant. Technical Report CMU/SEI-2003-TR-004, Software Engineering Institute, Carnegie Mellon University, March 2003.

[4] F. Bachmann, L. Bass, and M. Klein. Deriving Architectural Tactics: A Step Toward Methodical Architectural Design. Technical Report CMU/SEI-2003-TR-021, Software Engineering Institute, Carnegie Mellon University, March 2004.

[5] J. Bosch. *Design and Use of Industrial Software Architectures*. Addison-Wesley, Boston, MA, USA, 2000.

[6] J. Clark. XSL Transformations (XSLT) Version 1.0. W3c recommendation, http://www.w3.org/TR/xslt/, 1999.

[7] P. Clements, R. Kazman, and M. Kline. *Evaluating Software Architectures*. Addison-Wesley, Upper Saddle River, NJ, USA, 2001.

[8] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, and M. Goedicke. Viewpoints: a framework for integrating multiple perspectives in system development. *Int. Journal of Software Engineering and Knowledge Engineering*, 2(1):21–58, 1992.

[9] C. Hofmeister, R. Nord, and D. Soni. *Applied Software Architecture*. Addison-Wesley, Upper Saddle River, NJ, USA, 1999.

[10] IEEE Standards Board. *Standard 1471, Recommended Practice for Architectural Description of Software-Intensive Systems*. IEEE Computer Society Press, 2000.

[11] P. Inverardi and A. L. Wolf. Formal Specification and Analysis of Software Architectures using the Chemical Abstract Machine Model. *IEEE Transactions of Software Engineering*, 21(6):373–386, 1995.

[12] ISO 10746-1. Open Distributed Processing – Reference model. Technical report, International Standardization Organization, 1998.

[13] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopez, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In M. Aksit and S. Matsuoka, editors, *Proc. of the European Conference on Object-Oriented Programming (ECOOP 1997)*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242. Springer, 1997.

[14] P. Kruchten. The 4+1 View Model of Architecture. *IEEE Software*, 12(6):42–50, November 1995.

[15] D. Luckham, J. Kenney, L. Augustin, J. Vera, D. Bryan, and W. Mann. Specification and Analysis of System Architecture using Rapide. *IEEE Transactions on Software Engineering*, 21(4):336–355, 1995.

[16] J. Magee, N. Dulay, S. Eisenbach, and J. Kramer. Specifying distributed software architectures. In *Proceedings of the 5th European Software Engineering Conference*, pages 137–153, Barcelona, Spain, September 1995. Springer.

[17] B. Nuseibeh, J. Kramer, and A. Finkelstein. A Framework for Expressing the Relationships Between Multiple Views in Requirements Specification. *IEEE Transactions on Software Engineering*, 20(10):760–773, 1994.

[18] J. Putman. *Architecting with RM-ODP*. Prentice-Hall PTR, Upper Saddle River, NJ, USA, 2000.

[19] N. Rozanski and E. Woods. *Software Systems Architecture: Viewpoint Oriented Software Development*. Addison-Wesley, Boston, MA, USA, 2004. to appear.

[20] E. Woods. Experiences Using Viewpoints for Information Systems Architecture: An Industrial Experience Report. volume 3047 of *Lecture Notes in Computer Science*, pages 182–193. Springer, May 2004.