Using Architectural Perspectives

Eoin Woods
Artechra
www.artechra.com
eoin@artechra.com

Nick Rozanski
Artechra
www.artechra.com
nick@artechra.com

Abstract

A crucial aspect of the software architect's role is to ensure that a system based on their architecture will exhibit the quality properties (performance, security, availability and so on) that are important to their stakeholders. A proven approach to help guide an architect through the process of designing an architecture is to use architectural views, based on formal viewpoint definitions (such as those in the well known "4+1" set). However, a practical problem we have found when using existing viewpoint sets is the lack of guidance relating to system qualities (as opposed to system structures) that they provide. To address this problem, we identified a complimentary concept, called the architectural perspective [15], to provide an architect with practical guidance as to how to ensure that their system exhibits the right set of quality properties. This paper reviews the idea of the architectural perspective and relates a specific experience of applying them to the architectural definition of an enterprise integration project for a financial markets organisation, explaining the strengths and weaknesses we found in the approach.

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 architectural process is ensuring that a system will meet its quality requirements (for security, performance, availability and so on). 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, such as security or scalability.

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.

In order to address these points, we attempted to design a simple, yet systematic, approach to guide architectural design for quality properties, based on our experience as practicing information systems architects.

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 some ways, perspectives are similar to architectural viewpoints [10], in that as a viewpoint conventionally advises on how to create and describe a particular type of architectural *structure* [9][12][14], a perspective attempts to provide similar advice relating to the cross view concerns of a particular *quality property*.

To take an example, a performance and scalability perspective could contain advice to guide the architect through a process of assessing their system's performance and scalability via various modelling techniques and suggest tactics to apply (such as partitioning and replication) if it is found wanting, as well as listing common pitfalls to be aware of in this area (such as contention and careless allocation of resources), along with common solutions to these problems.

We believe that architectural perspectives are a useful and novel approach for the following reasons.

- 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.
- The approach is the product of practitioner experience, addressing a real need that we had.
- The approach has proven to be useful in practice.

Based on our experience of applying perspectives ourselves, and with others, we feel that they have the potential to help architects share knowledge about designing for quality properties much more effectively than is the case today. By documenting best practice and proven solutions, in a practical concrete context, a set of perspectives can help architects to share proven best practice, standardise their approach to some aspects of architectural design, reduce the risk involved in designing for certain quality properties and improve communication between architects, so facilitating discussion of alternatives and options.

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

2 Description of the Approach

2.1 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 an 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 (as viewpoints developed to date conventionally do) the perspective addresses an important quality property. This said, there is a critical difference between a viewpoint and a perspective: while a viewpoint is realised directly as part of the architectural description (i.e. as a view) a perspective does not result in the creation of a single part of the architectural design, but rather guides the architect to modify a number of the existing views in order to achieve the quality properties important for their system.

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) [3] 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 help 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 [12]), 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.

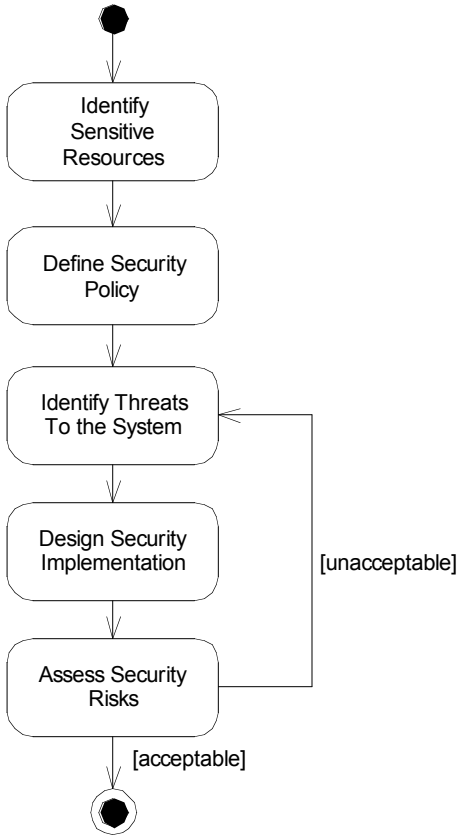

Figure 1. Security Perspective Process

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

- Application of recognised security principles (such as 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;
- 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 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 book [15], aimed at practicing software architects and those in training for the role.

2.2 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 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 mémoire* 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.

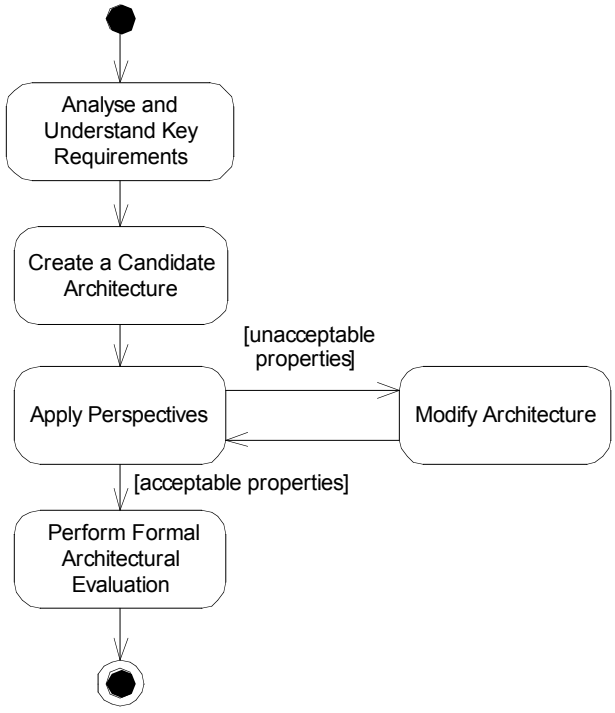

Figure 2. Using Perspectives

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 [6].

Obviously this description is a very idealised view of the process, but it provides a useful mental model and we have found it useful when explaining perspectives to other architects. In reality of course, the experienced architect is considering quality properties continually, during the design process and 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 artifact. (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 artifacts.) 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 [3]. 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.

3 An Example Application of Perspectives

3.1 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. 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 [5], to isolate the data mapping in well defined places and to allow it to be reused in future implementations. The first delivery of the system linked two source systems to one target system, supporting about 20 key business entities, with support for two or three other systems being scheduled for later iterations. This initial delivery involved the development of about 60 data transformations, a number of which were several thousand lines long, containing quite complicated mapping logic.

When the system runs, it extracts data from a number of source systems using existing data-access interfaces, converts it into a system-neutral organization wide data model and then supplies the subsets of the data required by each target system to these systems in their native formats. Initially, the 20 key business entities supported by the system resulted in several hundred megabytes of raw input data being processed in each run, the system neutral form of the data being several times this size.

The UML component diagram in Figure 3 illustrates the functional structure of the system.

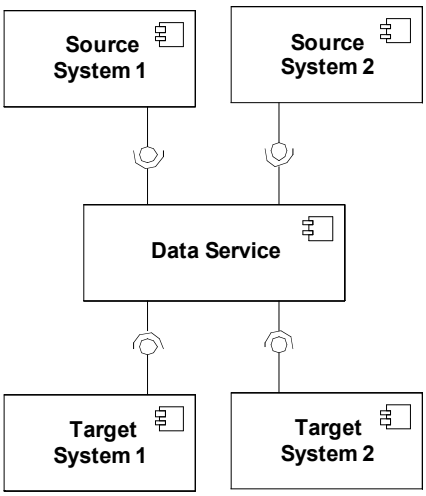

Figure 3. Data Service Functional Structure

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 megabytes 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.

3.2 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. However, we will also briefly touch on how the Evolution perspective was used.

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 architect and the mentor used the perspectives to refine it to meet the critical performance and evolution qualities required. 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 behaviour. 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 modelling, 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.

Having had previous experience of mentoring architects and performing architectural design ourselves for performance critical systems, we feel quite strongly that applying the perspective was a great improvement over the intuitive way of working that we had used before, for several reasons.

Firstly, the perspective's systematic approach and well organised information encouraged both architects to work through the system's performance characteristics, even though they suspected that all was well. In fact, as seen above, several potential performance problems were identified because of this, that would otherwise probably have been overlooked.

In addition, the reference material in the perspective allowed both architects to rapidly remind themselves of what to focus on, what to bear in mind and how to resolve problems if found, so helping them to be more effective. While all of this material is also available in specialist texts, the summary presentation in the perspective makes it much more accessible and so more likely to be used.

Lastly, the checklists and summary material in the perspective simply meant that the architects were less likely to forget something important.

As mentioned above, evolution was also an important quality requirement for this system and so the architects also applied the Evolution perspective. We do not have space to describe this activity in detail, but it is worth briefly discussing how the perspective was used.

The Evolution perspective guides the architect to consider the types and likelihood of occurrence of evolution in their system. For this system, this resulted in the architects identifying two likely future evolutionary needs, firstly the need to provide network access to the data and so host the system in an application server, and secondly the need to support new systems and business entities on a routine basis.

Based on this assessment, a key architectural design decision was made, to isolate all transformation logic in XSLT style sheets, specific to the source or destination system, as this allowed easy addition of new entities and systems and also allowed the style sheets to be reused in an application server implementation if needed. However, this decision immediately resulted in a conflict between the need to support easy evolution and the need to achieve certain scalability goals, as the XSLT processing model requires all of the input data set to be loaded into memory, so limiting scalability.

Having considered this trade-off, the architects decided to commit to the use of XSLT, as the evolution requirement was paramount in the short term, but to identify and document a number of alternative options that could be used to provide better scalability for specific transformations in the future if required.

Again, the structured approach that the perspective encouraged meant that the architects worked through their evolution needs systematically and this helped them to confidently make a trade off between two competing qualities.

In summary, using the two perspectives during the project's architectural design activity helped the architects to achieve their quality goals (performance, scalability and evolution), while being confident that they had considered the problems systematically and made a trade off that they could be satisfied with.

4 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 organizing and using knowledge, which is often a major challenge for software architects, given the breadth of the role.
- Using perspectives helps to avoid the duplication of information between views, and the inevitable resulting maintenance and traceability problems, which could otherwise occur if quality property based views were created.
- The use of 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 pitfalls to be aware of when applying 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 use of a perspective results in modifications to a number of views, but does not explicitly record the modifications (i.e. the design decisions) made. It is important that the architect maintains records of the important decisions made and their rationale, including those decisions made as a result of applying a perspective.
- 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 [2]) and the process of applying a perspective is still a skilled job that relies entirely upon the architect's abilities.

5 Lessons Learned

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

5.1 Viewpoints and Quality Properties

We have found the viewpoint-oriented approach very valuable for organising the software architecture process. However, we have found the approach, as exhibited in existing viewpoint sets, 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, or in private correspondence, on this point, with the core of the argument being that you can create any viewpoint you like (for example a "Security" viewpoint) and so this addresses our point.

In fact, our experience is that creating quality property based viewpoints and views is not a particularly effective approach, for two main reasons.

Firstly, when creating a quality-based view, you inevitably end up duplicating a lot of information from the fundamental structural views, so eliminating one of the great advantages of views (the fact that their disjoint

nature means that there is little duplication between them). Taking security as an example, a Security view will inevitably duplicate a lot of the Deployment view, in order to show how security technology is used to secure the system. Such duplication makes maintenance of the architectural description difficult and can be confusing for stakeholders reading it, as they have to relate two similar, but different, descriptions of parts of the system.

Secondly, one of the weaknesses of a view-based description is that you inevitably end up fragmenting your description into a number of related parts, which can make it hard to get an overall understanding of the system. Adding more views only makes this worse and most systems are going to need a number of quality-based views in order to discuss their important qualities.

Intuitively, we also find a separation between design advice for structures and qualities useful in organizing both information and the architectural design process, particularly for novice architects.

5.2 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 organize effectively.

5.3 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 practicing architects and the willingness of their managers to pay for its adoption.

5.4 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.

5.5 Making Architectural Tradeoffs is Difficult

One of the limitations of perspectives is that the approach 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.

6 Related Work

Within the area of software architecture, the most closely related work to architectural perspectives is probably architectural tactics [1]. Perspectives embrace and extend tactics by providing advice relating to what the architect should know, do and be aware of, as well as the specific solution advice provided by an architectural tactic.

The collection of rich sets of architectural viewpoints that are available, such as 4+1 [12], Siemens [9] and Garland and Anthony [8] are another related area of software architecture work. As noted earlier, perspectives are the result of our experience in applying viewpoints [16] and are designed to be a compatible extension to the approach. A second closely related concept is that of a viewtype [7], that provides guidance on documenting an architectural structure, based on an architectural style

Architectural perspectives are also complimentary to the work performed in the area of software architecture

evaluation, characterised by evaluation methods like ATAM [6]. Formal evaluation methods like ATAM allow an architecture to be evaluated for suitability with respect to stakeholder defined goals, whereas perspectives guide the architect through the process of achieving these desired qualities, in order to produce a suitable candidate architecture that can be evaluated.

Outside the immediate area of software architecture, architectural perspectives also have some similarities to a number of other active areas of software engineering research. In the requirements engineering sphere, the term "viewpoint" has been used to describe an approach for structuring requirements [13] in a way that allows a number of different views of a system to be described and related to each other. At the other end of the spectrum, aspect oriented programming technology [11] is another way of considering cross-cutting concerns within a system, which has some conceptual alignment with the system-wide quality property focus of perspectives.

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 for 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 interquality 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.

8 Acknowledgements

We would like to thank Professor Wolfgang Emmerich of University College London for his valuable advice and review during the preparation of a previous version of this paper.

9 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] 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.

[3] 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.

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

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

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

[7] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. N. Nord and J. Stafford, *Documenting Software Architectures, Views and Beyond*, Addison-Wesley, Upper Saddle River, NJ, USA, 2002.

[8] J. Garland and R. Anthony, Large Scale Software Architecture: A Practical Guide Using UML, John Wiley, 2002.

[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] 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.

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

[13] 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.

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

[15] N. Rozanski and E. Woods. *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives*. Addison-Wesley, Boston, MA, USA, 2005.

[16] E. Woods. Experiences Using Viewpoints for Information Systems Architecture: An Industrial Experience Report. In *Proceedings of the First European Workshop on Sofware Architecture (EWSA2004)*, volume 3047 of *Lecture Notes in Computer Science*, pages 182–193. Springer, May 2004.