Where does architecture end and technology begin?

Rami Razouk
The Aerospace Corporation

Introduction

Over the last several years, the software architecture community has reached significant consensus about what we mean by architecture; what differentiates architecture from design; the role that architecture plays in the development of products and product lines; and how we can analyze architectures to anticipate development problems. One area where some confusion remains is differentiating an architecture from the technologies and products necessary to make that architecture viable. A good example of that is CORBA - the Common Object Request Broker Architecture. CORBA is a combination of architectural style, architecture, technology and product. Adopting CORBA has profound implications on the architecture of a software system, and involves a significant amount of technology insertion. The success of a given project depends on understanding what it takes to develop CORBA-based applications as well as experience and knowledge of the capabilities of specific ORBs. Architecture evaluation approaches, while helpful in assessing the goodness of CORBA-the architecture, don't address the technology risk involved in adopting CORBA-the technology or CORBA-the product. This is not a CORBA-unique issue: it applies equally to a large number of architectural approaches that involve new and emerging technologies (e.g. DCOM, Java, XML). This paper attempts to explore this issue more fully, hopefully providing some insights into where architecture ends and technology begins.

Architectural style, Architecture, Technology & Product

For the purpose of this paper we will draw the distinction between the architectural style, the architecture, the technology and the products involved in building CORBA-based systems.

Architectural Style

Most CORBA-based designs begin with a desire to build a system using a distributed-object architectural style. We say "most" because some systems simply use CORBA as a communication mechanism and nothing more. Such systems make use of very few capabilities of CORBA. In such systems it is very clear that CORBA is nothing more that a product (little technology and no architecture). In the more common case the adoption of CORBA implies an acceptance of a certain architectural style. To succeed, the team must have some experience with that architectural style. That experience is often gained through a series of prototypes/projects that use CORBA. In that scenario it is difficult to tell if a team going through that process is struggling with understanding the distributed object architectural style, or simply struggling through the normal learning process associated with any new tool, technology or product.

Architecture

CORBA is a great deal more that an architectural style: it is also a specific architecture. It consists of a set of logically organized services that come together to support the construction of large distributed systems. These services include persistence, naming, security, and many more. As is the case with communications architectures, each service provided by CORBA has a logically thought out purpose. As CORBA has evolved, each successive iteration involved a subset of services that made sense to put together. Understanding the CORBA architecture is very important to the success of a development. Choosing to use some of the services to the exclusion of others may result in significantly diminished functionality and/or performance. In that respect, CORBA is truly an architecture.

Technology

CORBA also consists of a collection of technologies that form the foundation for the services. During the development of a CORBA-bases system it is not unusual to get involved in detailed technical questions of the following sort: Does the Internet Inter-ORB Protocol (IIOP) provide an adequate level of interoperability? What are the technical issues involved in using CORBA across firewalls? Is CORBA persistence model the right model for this application? Are language bindings available for specific languages?

These questions relate to the adequacy and maturity of certain technologies. One can have a good understanding of the architectural style and architecture of CORBA, but have an inadequate understanding of the specific technologies it incorporates. Adopting CORBA, in this way, resembles technology insertion more than architecting. One needs to take measures to mitigate technology risk through prototyping, and planning for alternate technologies (e.g. moving to DCOM or Java). When some of the underlying technologies are found to be inadequate (or over-kill), the resulting mixing and matching of technologies can pose significant additional risk to a development effort.

Product

CORBA also consists of a set of vendor supplied and supported products. Individual ORBs support the CORBA standard, but have unique features and capabilities that make inter-ORB interoperability a challenge. Some ORBs are more capable than others, some have superior performance, while others are more or less robust. Understanding the differences between the ORBs is needed to select specific products and product suites that best meet the requirements of a project. The issues of concern have to do with maturity of a product, the long-term viability of a vendor, the long-term maintenance issues and the interoperability with other products from other vendors. In this way, the adoption and use of CORBA is more of a product-selection process than technology insertion or architecture assessment

Architecture Evaluation

Each of the unique perspectives described above implies a different approach to architecture evaluation. A choice among architectural styles can be made, to a large extent, independently of the specific architecture. Similarly, choosing a particular CORBA product can be done independently of a comparative analysis of CORBA and DCOM. Below, we look at each perspective, examine typical questions that arise in doing architectural tradeoffs, and suggest some evaluations that could be done that are unique to each perspective.

Architectural Style

- **Is the chosen architectural style a good fit for the application domain?** Adopting an architectural style because it is the "in thing" is obviously the wrong (but all too common) thing to do. The distributed-object architectural style is intended to provide great benefits in application domains that don't require very tight coupling of data and processing resources. Inappropriate use of distributed-object architectures can result is extremely poor performance (if the data granularity is wrong). That poor performance is typically blamed on the architecture, the technology or the product when in fact the blame should be placed on the architectural style. Evaluation techniques that focus on matching the architectural-style to the application domain would help a great deal here.
- **Does the architectural style provide the "ilities" desired (e.g. scalability, flexibility, ditributability, and interoperability)?** Architectural style and architectures are all about "ilities." There are no absolute measures of such "ilities." Typically system architects perform relative comparisons of architectural styles given a set of "ilities" the system must possess. Different systems require different degrees of flexibility. Physical distribution requirements differ dramatically depending on the application. Similarly, interoperability often means different things in different

contexts. The kinds of analyses needed here are comparative analyses, given some set of "ility" requirements. For example, it is desirable to be able to compare a pipe-and-filter style against a distributed-object style, given a requirement of multi-platform interoperability across the Internet.

- **Is the development team experienced in building systems using this architectural style?** Understanding the distributed-object style, if one is not accustomed to it, is akin to making the transition from functional programming to object-oriented programming. Just because someone has used some limited parts of CORBA doesn't mean that they understand the basic approach to building distributed-object applications. Inexperienced development teams can struggle mightily and encounter difficulties that appear on the surface to be technology and product difficulties. The reality is that such teams architect the software in a manner that is inconsistent with the architectural style. Evaluation methods that can evaluate the consistency of the architecture with the style can be very helpful in identifying these sorts of problems. Evaluating a team's experience with architectural styles is a topic that requires attention.

Architecture

- **Does the architecture provide the "ilities" desired (e.g. flexibility, performance)?** Some "ilities" are tied to the architecture rather than the architectural style. It is therefore important to be able to evaluate specific architectures to determine if they meet such requirements. Much of the current work in architecture description and analysis typically deals with this form of evaluation.
- **How complex/simple is the architecture?** While most architectural styles appear conceptually simple, the underlying architectures can be quite complex. That complexity arises from the desire to provide a great deal of functionality. For example, CORBA provides security and persistence services. Some of those services can be quite complex while others are simple an easy to understand. This complexity/simplicity can significantly affect the likelihood of success of a project. The ability to analyze the conceptual complexity of an architecture would be very helpful in performing architectural tradeoffs.
- **Do the components/services of the architecture form a coherent whole?** Architectures as comprehensive as CORBA involve many components and services. A superficial examination of the CORBA architecture does not reveal the complex interdependencies of the components and services. One might be tempted to think that the services are completely orthogonal to one another and that one can simply pick and choose the subset of services for use in a particular system. The reality is that there are coherent subsets of services that should be used together to minimize the need for additional components. Mixing a subset of CORBA services with the persistence mechanisms of a database management system and the security services of native operating systems is a very complex task. The kind of analysis needed here is the ability to determine if a subset of services/components is internally consistent, provides integrated functionality and has a certain degree of cohesion. Currently this sort of information is gained through painful experimentation.
- **Does the development team have experience with the CORBA architecture?** The development team needs a deep understanding of the CORBA architecture. They need to understand how the components and services of CORBA inter-relate. They also need experience in using the architecture to achieve some of the desired "ilities." Assessing the experience of a team with a specific architecture is an important issue that needs to be looked at in the context of software capability evaluation (e.g. SCE and SDCE).

Technology

- **Does the technology meet the project needs?** From a technology perspective it is important to understand if the underlying technologies meet or exceed the requirements of a system. For example, does the set of technologies that comprise CORBA security services meet the security requirements of the system?

- **Is the technology immature, mature or obsolete?** In developing large-scale systems whose development takes years, technological maturity and/or obsolescence can be very significant challenge. The current technological pace in the software world is far exceeding our ability to inject the technology into large-scale developments. Attempts to anticipate technological innovation lead to many false starts. Adopting new technologies as soon as they emerge (banking on rapid development) means that development teams are constantly re-tooling and learning new technologies and never have the opportunity to leverage their understanding of the technologies. Adopting only mature technologies is a recipe for obsolescence. To make tradeoffs between technologies, we need better metrics and better methods for assessing the maturity of technologies. In the case of hardware, Moore's law has been very reliable in predicting technological pace and can be used to determine the likely obsolescence of a technology. No software equivalent to Moore's law exists, and one is badly needed.
- **How can we mitigate the technology risk, if any?** Examples of premature technology insertion abound. The rush to be on the bleeding edge of technology may be a necessary thing to live in the fast-moving e-commerce world, but it is unnecessary and dangerous in the design and construction of large-scale mission-critical systems.
- **Does the technology fit into and inter-operate with other technologies involved in a system development?** Invariably software systems are designed and built to live in particular computing environments and to interact and inter-operate with other software and hardware systems. The technologies need to inter-operate to allow the systems to inter-operate. Our ability to assess technology inter-operability is weak at best. More work in this area would be quite helpful.
- **Does the development team have experience with the technology?** The issue of development team experience is one that is consistent across all the views discussed in this paper. Technology experience specifically means that the development team understands how the technology that underlie CORBA relate to ongoing technological developments in the field, how the technologies inter-operate, and how to meet system requirements by exploiting the underlying technologies.

Product

- **Do the products provide the needed functionality and performance?** Vendor offerings are distinct from CORBA standardization efforts. Some functionality may precede or lag adoption by the Object Management Group (OMG). It is important to understand where the particular products stand in relation to each other and to the standards. Methods of evaluating products against a standard would help answer some of these questions.
- **Are the products immature, stable or out of date?** Independently of the underlying technologies, vendor products may be more or less mature. The rush-to-market to establish market-share often pushes vendors into releasing immature and buggy products. The resulting impact to a project's schedule can be significant. Product immaturity can be often mistaken for technology immaturity or architectural deficiencies. Such mistakes can cause an unwarranted abandonment of a perfectly good architecture.
- **Is the vendor stable**? Although inter-ORB interoperability is a key element of what CORBA is all about, there is still a great deal of work to be done in moving from one ORB vendor to another. Operating system dependencies can force a difficult port to a later OS release to allow the adoption of a new product. To avoid these kinds of issues, vendor stability is an important issue.
- **What are the short-term and long-term costs associated with the products?** Some of the hidden costs of using commercial-off-the-shelf products are associated with the long-term licensing and maintenance of the products. In this respect CORBA is no different than any other COTS product.

Discussion

As discussed above there are important differences to be drawn between adopting an architectural style, an architecture and a product or technology. Architectural evaluation methods are typically targeted at architectures and architectural styles. Technology and product evaluation, on the other hand, is typically done through prototyping and testing. Given the ambiguity described above, the key challenge we face is how to recognize the various views of CORBA, and to correctly attribute difficulties faced by a development team to the appropriate views. This is particularly important because each view requires unique remedies. For example, let's assume that a specific CORBA-based system has been analyzed and shown to suffer from poor performance. Is that an inherent problem with the fact that distributed objects incur overhead? Is the architect making inappropriate use of object distribution by having very small granularity objects? Is the problem an inherent limitation of the CORBA architecture? Is the problem with a particular vendor's ORB? In the first case, the remedy would be to give up the advantages of a distributed-object architectural style in favor a higher-performance, more tightly coupled architectural style. If the problem is inappropriate granularity, then the architecture team may require some training. If the problem is CORBA, then the remedy may be migrating to DCOM or JavaBeans. If the problem is the ORB vendor, then the remedy may be to select an alternate vendor.

Conclusion

As software technologies advance, we are faced with an ever-changing landscape of architectural-styles, technologies, products, buzzwords and vaporware. Yesterday we discovered object-oriented architectures. Today we're dealing with CORBA, DCOM and Java. Tomorrow we'll be dealing with XML and all its specific instantiations. Some claim to be architectures, some claim to be technologies, some claim to provide infrastructure support, and all are sold as products. The benefits of these changes and advances are evident in the rapidly changing Internet. To take advantage of these advances it is important to get beyond the buzzwords and religious arguments to better understand where the strengths and weaknesses lie. Some changes require a complete rethinking of how we build systems. Some changes require us to re-architect our systems. Some changes require us to carefully choose among technologies and technological trends. Some changes simply require us to open our pocket books and buy brand new products.