Valuing Scalability in Distributed Architectures

Rami Bahsoon
School of Engineering and Applied Science, Computer Sc,
Aston University Birmingham,
Aston Triangle, B4 7ET, Birmingham, UK
r.bahsoon@aston.ac.uk

Wolfgang Emmerich
London Software Systems, Dept. of Computer Science,
University College London,
Gower Street, WC1E 6BT, London, UK
w.emmerich @cs.ucl.ac.uk

ABSTRACT
Drawing on a case study that adequately represents a medium-size component-based distributed architecture, the novel contribution of this paper is an economics-driven software engineering approach to the valuation of scalability in distributed architectures. Using real options analysis, we report on how ranges in which changes in scalability requirements can inform the selection of distributed components technology and subsequently the selection of application server products. As the exact method for analyzing scalability is subject to debate, we identify views for analyzing scalability. We then focus the analysis on throughput as a way for measuring scalability. We describe a real options model for valuing the ability of a given architecture to scale. We apply the model on two versions of a given architecture, each induced with a distinct middleware: one with CORBA and the other with J2EE. The results show that the application of real options theory to the said problem is superior to that of traditional techniques, such as Net Present Value (NPV), for estimating value in software. This is because the latter systematically underestimate the value of the architectural flexibility under uncertainty, where uncertainty is attributed to the unpredicted change in load.

1. INTRODUCTION
Software architecture is the earliest design artifact, which realizes the requirements of the software system. It is the manifestation of the earliest design decisions, which comprise the architectural structure (i.e., components and interfaces), the architectural topology (i.e., the architectural style), the architectural infrastructure (e.g., the middleware), the relationship among them, and their relationship to the other software artifacts (e.g., low-level design) [Bah05a]. In many software systems, the architecture is the level that has the greatest inertia when external circumstances change and consequently incurs the highest maintenance costs when evolution becomes unavoidable [Coo01]. Current industrial evidence is revealing situations where system evolution is unavoidable and much of the promise is leaved to the architecture in scaling the system and its services. For example, the number of mergers between companies is increasing and this trend is bound to continue. The different divisions of a newly merged company have to deliver unified services to their customers and this usually demands scaling the system, while leaving the core architecture intact. The time frame is often so short that building a new system is not an option and therefore existing system components have to be integrated into the architecture to appear as an integrated computing facility. Secondly, the trend of providing new services or evolving existing services to target new customers, devises and platforms, and distribution settings (e.g., mobility setting) is increasing. For example, moving from a fixed distributed setting to mobility carries critical changes, mainly to non-functionalities, such as changes in availability, security, and scalability requirements. Often the core "fixed" architecture falls short in scaling up; henceforth, changes to the architecture becomes necessary. Thirdly, it is often the case that components are procured off-the-shelf, rather than built from scratch, in response to changes in requirements and then need to be integrated into the core architecture. The architecture may fail to scale up, as these components often have incompatible requirements on the hardware and operating system platforms they run on. Fourthly, a lean economy has forced those with a limited IT budget to more fully "utilize" the architecture so it becomes more flexible in responding to rapidly evolving markets and scale to support business growth. As many companies have come to the conclusion that it is essential to more fully leverage the computing assets they already have, the importance of utilization has increased; it has become, for example, necessary to utilize what the architecture may support in handling more business transactions at a unit-time.

Failing to accommodate the scalability requirements may "break" the software architecture necessitating changes to the architectural structure (e.g., changes to components and interfaces), architectural topology (e.g., changes to the architectural style), or even changes to the underlying architectural infrastructure (e.g., middleware). It may be expensive and difficult to change the architecture as requirements evolve [Fin00]. Conversely, failing to accommodate the change leads ultimately to the degradation of the usefulness of the system. Hence, an architecture which is *flexible* and scale to address such changes in requirements with limited resources and shorter time-to-market is a significant asset for surviving the business, cutting down maintenance costs, utilizing resources, and creating value.

Reflecting on the discipline, [Sul99] note that the important book on software architecture begins, "As the size and complexity of software systems increase, the design and specification of overall system structure become more significant issues than the choice of algorithms and data structures…". [Sul99] add, "This statement is true, without a doubt. The problem in the field is that no serious attempt is made to characterize the link between structural decisions and value added". Hence, the challenge that is facing the software engineering community is that there is a general lack of adequate models and methods, which connect technical engineering concepts to value creation under given circumstances. Despite the clear connection of scalability to value, there is a general lack of value-driven models and methods, which connect this property to value under given circumstances. Our contribution aims to address this need.

As a motivating example, consider a distributed software architecture that is to be used for providing the back-end services of an organization. This architecture will be built on middleware, such as Java 2 Enterprise Edition (J2EE) [Sun02] and the Com-

mon Object Request Broker Architecture (CORBA) [Obj00]. Depending on which middleware is chosen, different architectures may be induced [DiN99]. These architectures will have differences in how well the system is going to cope with changes. For example, a CORBA-based solution might meet the functional requirements of a system in the same way as a distributed component-based solution that is based on a J2EE application server. A notable difference between these two architectures will be that increasing scalability demands might be easily accommodated in the J2EE architecture because J2EE primitives for replication of Enterprise Java Beans can be used, while the CORBA-based architecture may not easily scale. The choice is not straightforward as the J2EE-based infrastructures usually incur significant upfront license costs. Thus, when selecting an architecture, the question arises whether an organization wants to invest into an J2EE application server and its implementation within an organization, or whether it would be better off implementing a CORBA solution. Answering this question without taking into account the *flexibility* that the J2EE solution provides and how *valuable* this flexibility will be in the future relative to the likely change in load might lead to making the wrong choice. Furthermore, the ranges in which scalability requirements change may need to inform the selection of distributed components technology, and subsequently the selection of application server products.

Drawing on a case study that adequately represents a medium-size component-based distributed architecture, the novel contribution of this paper is an economics-driven software engineering approach to the valuation of scalability in distributed architectures. Using real options analysis, we report on how ranges in which changes in scalability requirements can inform the selection of distributed components technology and subsequently the selection of application server products. As the exact method for analyzing scalability is subject to debate, we identify views for analyzing scalability. We then focus the analysis on throughput as a way for measuring scalability. We describe a real-options based model for valuing the ability of a given architecture to scale. We apply the model on two versions of a given architecture, each induced with a distinct middleware: one with CORBA and the other with J2EE. Traditional economics approaches to software design appeal to the concept of static Net Present Value (NPV) and Discount Cash Flows (DCF) as a mechanism for estimating value [Boe00]. We show that options theory is said to be superior to PV and DCF in valuing scalability, as the latter fall short in valuing the flexibility of an architecture under uncertainty, where uncertainty is attributed to the unpredicted change in load. The case demonstrates how change impact analysis on a system of a given architecture can be complemented with value-based reasoning. The rationale is that the combination could provide the architect/analyst with a useful tool for understanding the extent to which the software system is flexible to accommodate the change; provide insights on the likely success (failure) of software evolution; and consequently on the potential stability of the architecture to change. This combination could also account for the economics ramification of the change on the structure and the behaviour of the system. For example, throughput, a scalability measure, is correlated with value. That is, the more business transactions can be performed on a system of a given architecture, the more value is said to be created for the enterprise. Hence, "hurting" the performance of the software, upon accommodating the change in scalability requirements, implies "hurting" value.

The paper is further structured as follows. Section 2 describes the case study, devises an options based model to value scalability, reports and discuss the model's application. Section 3 discusses closely related work. Section 4 concludes.

2. CASE STUDY

2.1 Setting

We use the Duke's Bank application [Sun02], an online banking application, which adequately represents a medium-size component-based distributed system. The architecture of the Duke's Bank application has a three-tier style, given in Figure 1. The architecture has two clients: an application client used by administrators to manage customers and accounts, and a Web client used by customers to access account statements and perform transactions. The server-side components perform the business methods: these include managing: customers, accounts, and transactions. The clients access the customer, account, and transaction information maintained in a database.

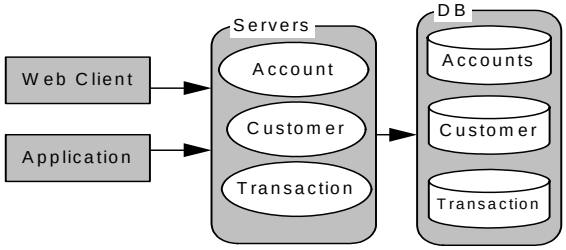

Figure 1. The Architecture of the Duke's Bank

We instantiate from the core architecture two versions, each induced by a different middleware: one with CORBA and the other with J2EE. Assume that the Duke's Bank system needs to scale to accommodate the growing number of clients in one-year time. Scalability denotes the ability to accommodate a growing future load, be it expected or not. We observe how a likely future change in scalability, a representative critical change in non-functional requirement, could impact the architectural structure of each version. The challenge of building a scalable system is to support changes in the allocation of components to hosts without breaking the architecture of the software system; changing the design and code of a component [Emm02]; and/or rippling the change to impact other non-functionalities such as performance, reliability, and availability. We use replication, an architectural mechanism, to achieve scalability. The reason is due to the fact that both CORBA and J2EE do provide the primitives or guidelines for scaling a software system using replication, which make the comparison between the two versions feasible. In particular, the Object Management Group's CORBA specification defines a fault tolerance and a load balancing support, which provides the core capability for implementing scalability through replication. Similarly, J2EE provides clustering primitives for scaling the software system through replication. Interested reader may refer to [Bah05b] for more details.

2.2. Valuation Views to Scalability

Scalability is frequently thought of in terms of numbers of users that can be supported on either a single node or collectively on all nodes in a system; it denotes the ability to accommodate a growing future load. The exact method of analyzing scalability is subject to some debate: First, the change in load demands is critical as it could impact the architecture at its various levels: structure, topology, and infrastructure. For example, the challenge of building a scalable system is to support changes in the allocation of

components to hosts without breaking the architecture of the software system, or changing the design and code of a component [Emm00b]. Second, the change in load could impact other non-functional requirements such as performance, reliability, and availability, when the change is poorly accommodated by the architecture. As a result, this debate is appealing to the use of the multi-perspective valuation points of view: structural and behavioral valuation points of view to the analysis of the change in scalability requirements.

On the *structural point of view*, in [bah05b] we have observed how the architecture of the given system, when induced by a particular middleware, is ready to cope or need to be maintained for supporting the change in scalability. We have analyzed the impact of the change by looking at the structural changes and the source lines of code (SLOC) that need to be modified/added for implementing the change, configuring, and deploying the software system. We have also quantified the value of the structure in scaling to accommodate the change, by looking at the cost of change on the structure of each version and by valuing the savings in maintenance, deployment, and configuration costs to realize the change. As reported in [Bah05], an observable advantage of scaling the software architecture when induced by J2EE is that no development effort is required to realize the scalability requirements through replication, as when compared to the CORBA version. J2EE does provide clustering primitives for scaling the software system, which result in making the architecture of the software system more *flexible* in accommodating the change in scalability, as when compared to the CORBA version. Considering the CORBA-induced architecture of the Duke's Bank, supporting scalability through replication has not leave the middleware infrastructure and the application layer intact. Though the use of both CORBA specification and design patterns[OMG00] has simplified the task of realizing the requirements for achieving fault tolerance and load balancing, implementation and integration overhead have not been abandoned. In particular, the fault tolerance and load balancing services need to be implemented and be integrated into the used middleware. The server and the client application need to be updated. Interested reader may refer to [Bah05b] for results and details on this view.

In this paper, we complement the analyses by looking at the behavioral point of view to analyze scalability. On the *behavioral point of view*, we use *throughput* or the capacity of the system to measure scalability. Throughput is a performance criterion, which expresses the amount of work performed by the system under test during a unit of time. For this *view*, we elicit the likely ranges in future load. We discuss the impact of *likely* change in future load on the behavior of the system. We then describe how ArchOptions can be used to reason about the change. Throughout the paper, we focus on this view.

2.3. The Throughput View to Valuing Scalability

A possible way to treat scalability is to assume that scalability can be measured by *throughput* or capacity of the system. Throughput is a generic performance criterion, which expresses the amount of work performed by the system under test during a unit of time. This criterion is based on the observation that for a fixed system with a given throughput (e.g., a single host), there is an inverse relationship between the response time and the number of clients. In other words, the more clients submitting requests, the longer are the delays. A well-known throughput metric is the Total Operations Per Second (TOPS) completed during the measurement interval, referred to as TOPS [http://www.spec.org/]. TOPS is composed of the total number of business transactions completed in the customer domain, added to the total number of work orders completed in the manufacturing domain, normalized per second[http://www.spec.org/].

To understand how Duke's architecture may behave once induced with J2EE or CORBA, we have screened relevant performance benchmarks (e.g., [Den04]; http://www.spec.org/jAppServer2005/). We appeal to the use of published benchmarks, because the system of the given architecture need not be implemented during the evaluation. Thus, performance measures may not be available. Benchmarks are revealing on the performance dimension because, for example, if multiple benchmarks are conducted with a suitable mix of relevant factors, it may be possible to obtain a set of basic scalability results that can be used for estimating the throughput of possible configurations of the architecture. Depending on the benchmarking algorithm, the relevant scalability factors can be, for example, the number of objects, the number of clients, or the number of nodes in the system etc. supported in response to growing load. A major problem in comparing benchmark results, however, is that different hardware platforms and configurations (e.g., memory, disk drives etc) often produce different results making the comparisons difficult. Further, vendors often try many different ways to optimize performance, including adding cache memory and putting cache buffers on disk arrays. Therefore, we only use benchmarks, which are close to the case at hand. We then normalize the screened benchmarks for easing the comparison. It could be also argued that in iterative development (e.g., in the Unified Process) partial implementations might be available at the end of each phase. In this context, it is possible to create benchmarks from the partial implementations and to use them in recalibrating the screened ones. The intention is to have more meaningful figures which we could use for understanding the impact of likely change in future load on the behavior (throughput) of the system and the corresponding economics ramifications.

Figure 2 shows the likely throughput trend that the J2EE-induced architecture may exhibit relative to the CORBA-induced one, upon varying the TOPS and the number of hosts. For the J2EE-induced architecture, we provide throughput estimations for two possible implementations: one with JBoss and the other with WLS. For the CORBA-induced architecture, we provide estimates upon the use of JacORB to induce the architecture. Table 1 depicts the upper limit of TOPS supported per host for each of WLS, JBOSS, JacORB induced architectures for 1 to 4 hosts.

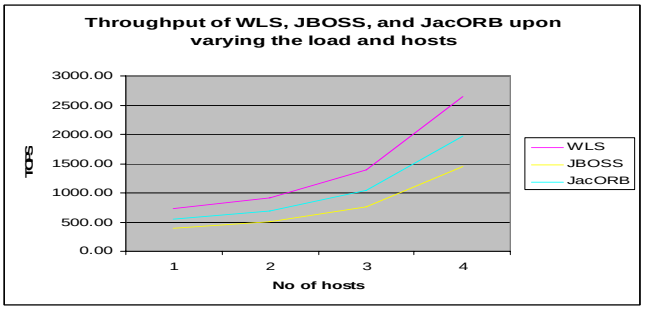

Figure 2. Plotting the TOPS per host for each of WLS, JBOSS, JacORB for 1 to 4 hosts

Table 1. Upper limit of TOPS per host for each of WLS, JBOSS, JacORB

Hosts	WLS	JBOSS	JacORB
1	732.00	400.26	546.80
2	918.36	502.16	686.01
3	1395.44	763.03	1042.39
4	2640.96	1444.08	1972.79

Figure 3 shows the likely cost-trend upon inducing the Duke's bank architecture with J2EE (using either WLS or JBOSS) and with CORBA (using JacORB). The likely cost is plotted against the number of hosts (1 to 4). The cost refers to the lifecycle cost of the System Under Test (SUT). The cost includes Application Servers/Containers, Database Servers, network connections, etc. Assuming, for example, a five-year lifecycle, cost would include all hardware (purchase price), software including license charges, and hardware maintenance. For the CORBA version, it assumed that the investment incurs an upfront cost to the development of the replication mechanism to support fault-tolerance and load-balancing services for high load scenarios [Bah05b]. For the J2EE version of WLS, a license cost is incurred per host.

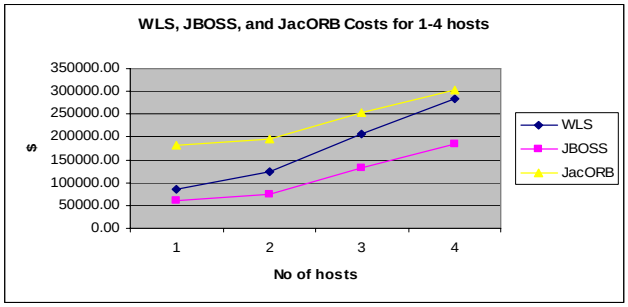

Figure 3. The likely cost-trend upon inducing the Duke's bank architecture with J2EE-(WLS or JBOSS) and with CORBA (JacORB).

Though the structural analysis appears to be in favor of the J2EE-induced architecture [bah05b], the throughput analysis may reveal a different trend. From the throughput valuation point of view, Figure 2 shows that when the Duke's architecture will be induced with JBOSS, a J2EE implementation, the system is likely to be slower than that of the JacORB one. This is because Jboss uses reflection [http://www.jboss.org]. This also implies that there are some chances for the JBOSS-induced architecture to require more hardware for addressing this deficiency. When inducing the Duke's architecture with WLS, another J2EE implementation, the system is very likely to be faster than that of the JacORB implementation. WLS, however, comes with significant licenses costs; this cost grows with the number of hosts, as the load increases. Coining the TOPS with their associated costs, Figure 2, Figure 3 and Table 1, hint that there might be a case for JacORB in certain throughput range. Moreover, note that once the services for realizing scalability (i.e., the fault-tolerance and load balancing service) are implemented, the cost is incurred once and amortized across the hosts. Hence, as the load grows, the analysis becomes complex.

2.4. Valuing Scalability with Options Theory

2.4.1 ArchOptions: background

Let us assume that we are given the choice of two middleware M_0 and M_1 to induce the architecture of a particular system. Let us assume that S_0, S_1 are the architectures obtained from inducing M_0 and M_1 respectively. Say, M_1 is an economical choice, if it adds value to S_1 relative to S_0. We attribute the added value to the enhanced flexibility of S_1 over S_0 in scaling up the architecture. But the added value is uncertain, as the demand and the nature of the future change and load are uncertain. A question of interest is: how valuable is the flexibility of either alternative, relative to likely change in scalability, will be in the long-run? Which solution is more valuable? Using options theory is suited to answer theses questions.

Real options analysis recognizes that the value of the capital investment lies not only in the amount of direct revenues that the investment is expected to generate, but also in the future opportunities that flexibility creates [Erd99; Erd00; Erd02; Sul99; Sul02]. These include growth, abandonment or exit, delay, and learning options. An option is an asset that provides its owner the right without a symmetric obligation to make an investment decision under given terms for a period of time into the future ending with an expiration date [Tri95]. If conditions favourable to investing arise, the owner can exercise the option by investing the exercise price defined by the option. A call option gives the right to acquire an asset of uncertain future value for the strike price [Tri95].

The problem of selecting a particular middleware to induce a given architecture is an *option* problem. From the evolution perspective, the flexibility of the middleware induced-architecture in coping with changes in non-functional requirements has a value. More specifically, flexibility adds to the architecture values in the form of *real options* that give the right but not a symmetric obligation- to evolve the software system and enhance the opportunities for strategic growth. The added value is strategic in essence, uncertain as the demand on the future changes are uncertain, and may not be immediate. The added value may take the form of (i) accumulated savings through coping with the change without "breaking" the architecture, mostly these are changes in non-functional requirements; (ii) extending the range of services while leaving the architecture intact; and (iii) the ability to respond to competitive forces and changing market conditions that may pose higher Quality of Service (QoS) requirements, such as the demands for higher availability, scalability, reliability and so forth. From an early development perspective, given several middleware candidates, the architect has the right without the symmetric obligation to embark on a selection for inducing an architecture. A "wise" selection could be regarded as an investment to buy flexibility, which could be valued as future *growth options* [Tri96] on the architecture of the software system. These options differ from one middleware to another.

ArchOptions[Bah05a; Bah05b; Bah04a; Bah03b], a real options based model that we developed, values the *growth* options of an architecture relative to some future changes, as a way for understanding the architectural flexibility and its stability implications. A growth option is a real option to expand with strategic importance [Tri95] and is common in infrastructure-based investments, as it is the case with software architectures. Since the future changes are generally unanticipated, the value of the growth options lies in the enhanced flexibility of the architecture to cope with uncertainty. ArchOptions

builds on a simple and intuitive analogy with Black and Scholes [1973], as described in Table 2.

Table 2. Financial/real options/ArchOptions analogy

Option on stock	Real option on a project	ArchOptions
Stock Price	Value of the expected cash flows	value of the "architectural potential" relative to the change $x_i V_p$
Exercise Price	Investment cost	Estimate of the likely cost to accommodate the change C_{eip}
Time-to-expiration	Time until opportunity disappears	Time indicating the decision to implement the change (*tp*)
Volatility	Uncertainty of the project value	"Fluctuation" in the return of value of V over a specified period of time (σp)

Accommodating the change, thus, is analogous to buying an "architectural potential" (i.e., an option on an asset) with uncertain future value paying an exercise price. The exercise price corresponds to the cost of accommodating the change on the system of the given architecture. The value of the call option, whether in-the-money or out-of-the-money, is a measure of the architecture flexibility in accommodating change. This value is an indicative measure of the "architectural potential" in unlocking future growth opportunities (e.g., case of reuse, new market products), enhancing the upside potentials of the architecture, generating value (e.g., savings in maintenance), or incurring losses (e.g., case of a disruptive changes), as a consequence of accommodating the change. The value of the call is a powerful heuristic, which can provide a basis for analyzing many architecture-centric evolution problems, which place considerable emphasis on the flexibility of the architecture as a way for easing software evolution.

Choosing a particular middleware to induce the architecture of the software system can be seen as an investment to purchase flexibility in the induced software architecture. The range in which the load change influence the choice. In this context, deciding on a particular middleware to induce the software system architecture can be seen as an investment to purchase future *growth options* that enhance the upside potentials of the structure when the load change. That is, S_1 is said to be more accommodating to the change than S_0, if S_1 holds more growth options than S_0. For a valuation point of view *p*, we focus the analysis on the calls of the ArchOptions model for valuing the growth options, as given in (1) accounting for both the expected value and exercise cost to accommodate future requirements i_i, for $i \leq n$. Valuing the expectation E of expression *(1)* uses the assumptions of Black and Scholes[Bla73] and detailed in previous work[Bah05b; Bah04a;.Bah03b]:

$$\sum_{i=1\ldots n} E\,[\max\,(x_i V_p - C_{eip}, 0)] \qquad (1)$$

The payoff of the constructed call option gives an indication of how valuable the flexibility of an architecture is, when enduring some likely changes in requirements. The selection has to be guided by the expected payoff in $(\sum_{i=1\ldots n} E\,[\max\,(x_i V_p - C_{eip}, 0])_{S1}$ relative to that of $(\sum_{i=1\ldots n} E\,[\max\,(x_i V_p - C_{eip}, 0])_{S0}$. That is, if $(-I_e + \sum_{i=1\ldots n} E\,[\max\,(x_i Vp - C_{eip}, 0)]_{S1} > \sum_{i=1\ldots n} E\,[\max\,(x_i V_p - C_{eip}, 0)]_{S0})$ for some likely changes, then it is worth investing in M_1, as the investment in M_1 is likely to generate more growth options for S_1 than for S_0 and relative to the p valuation point of view.

If $(E\,[\max\,(x_k V_p - C_{epk}, 0)])_{S1}=0)$, then M_1 is not likely to payoff, relative to M_0, as the flexibility of the architecture to the change is not likely to add a value for S_1 on *p*, if the change need to be exercised. Two interpretations might be possible: (i) the architecture is overly flexible in the sense that its response to the change(s) has not "pulled" the options relative to *p*. This implies that the embedded flexibility (or the resources invested in implementing flexibility- if any) are wasted and unutilized to reveal the options relative to the changes and relative to *p* (ii) the other case is when the architecture is inflexible relative to the change. This is when the cost of accommodating the change on S_1 is much more than the cumulative expected value of the architecture responsiveness to the change.

2.4.2 Application of ArchOptions

The case of valuing the scalability of the architecture is appealing to ArchOptions for the following major reasons: First, there is cone of uncertainty associated with the growing load and consequently in the value added as result of our choice. Moreover, the TOPS are of straightforward contribution to value. That is, the more operations are completed per second, the more value is added to the enterprise. However, TOPS incur a price upon executing the operations. The price again is dependent on several factors such as the number of hosts, the hardware, the license cost, and any additional costs that are necessary for making the middleware adaptable to the growing load. In the context of the Duke's Bank, the TOPS range is often uncertain as it is dependent on the customers' behavior at a time. The *uncertainty* in the likely range (i.e., TOPS), the associated costs for executing the TOPS, and the "fluctuation" in the value added as a result make the case very appealing to the use of ArchOptions. For the throughput valuation point of view, the analysis using ArchOptions aims at complementing the behavioral analysis to understand the trend in the added value upon embarking on either J2EE (JBOSS or WLS) or CORBA(JacORB) to induce the architecture of a given system. Second, in the context of ArchOptions, our use of benchmarks resembles the use of a twin asset. Real options valuation based on Black and Scholes pricing technique determines the value of an asset in question in span of the market value using a correlated *twin asset* [Tri95]. The twin asset is an asset that has the same risks as the asset in question will have when the investment has been completed [Tri95]. The intuition is that to understand the behavior of the asset in question, we can use a twin asset, also referred to as a replicated portfolio. The assumption is that under similar conditions the twin asset and the asset in question are interchangeable for all practical purposes and should be worth the same. That is, if we know how much the twin asset is worth in the present, we can then determine how much the option on the asset in question is worth in the present. We argue that using benchmarks satisfies the concept of the twin asset as we are relying on historical information showing possible variations in performance in connection to change in load and relative to the candidate implementations. These benchmarks often hint that the throughput is dependent on and can be estimated from the middle-tier "processing power" of the architecture. Such variation, we believe, is a wealth as it reveals pros and cons of the Duke's Bank execution under possible operating environments and/or in relation to other participating applications. This is advantageous because scalabil-

ity is also a factor of the number of independently developed applications that might share an execution platform. The advantage of this approach is that the published benchmarks could reveal risks of the operating environment on the choice.

For the throughput valuation point of view, *Pthro*, an additional operation is said to "buy" an architectural potential paying an exercise price. In terms of throughput, the architectural potential is a performance measure. That is, the more TOPS are said to be completed at a host (or for a configuration), the more value is said to be added to the enterprise. The more valuable is said the architectural potential relative to the TOPS. The exercise price is price/TOPS. If we assume that $x_i V_{PthroS1}$ is the value added in S_1 over S_0 due to the support of more TOPS, it is reasonable to consider that if $(\sum_{i=1\ldots n} E\ [\max\ (x_i V_{Pthro} - C_{eiPthro}, 0)]_{S1} > \sum_{i=1\ldots n} E\ [\max\ (x_i V_{Pthro} - C_{eiPthro}, 0)]_{S0})$, then investing in M_1 is said to payoff relative to *Pthro*. We construct call options for a likely change in load-range. The objective is to analyze the architectural potential in supporting a likely growth of TOPS. Below, we show how we estimate the parameters relative to *Pthro*.

Estimating ($C_{eiPthro}$). A change in a load-range is said to buy an architectural potential paying an exercise price. As we mentioned before, TOPS denotes the Total Operations completed per Second. For the simplicity of explanation, let us assume that the system of the induced architecture needs to scale up to support an additional operation per unit-time. An additional operation is said to buy an architectural potential paying an exercise price. In terms of throughput, the architectural potential is a performance measure. Hence, what an extra operation pays, if materializes, is a bandwidth for performing that operation. Inducing the Duke's bank with either J2EE or CORBA provide different bandwidth capabilities for performing the operation at different price. If the implementation of either happens to hold embedded growth options in supporting the extra operation, then the operation is said to pay an exercise price to buy options on the architecture. To estimate the exercise price, we use a well-known normalization factor, which is the *price/performance* [http://www.spec.org/jAppServer2005/] (i.e., the lifecycle cost of the System Under Test (SUT) as configured for the benchmark divided by the throughput). As an example, assuming five-year lifecycle, the cost would include all hardware (purchase price), software including license charges, and hardware/software maintenance. If the total price is $5,734,417 and the reported throughput is 105.12 TOPS, then the price/performance is $54,551.16/TOPS (54,551.151 rounded up).

Estimating ($x_i V_{Pthro}$). For simplicity, we estimate $x_i V_{Pthro}$ relevant to the business domain. For every completed online operation, Duke's would not need to have to serve a customer in person at a branch. That is, the Duke's savings are in the manual-effort for serving the clients at a branch. For example, let us assume a scenario where a clerk needs one minute for completing a business operation: if we assume an overhead cost of $100,000/year for each clerk, then an online operation saves about a dollar per operation in a minute: $100000/ (220day * 8hours * 60minutes). Computing savings per second is then straightforward. We use scenarios of 8 and 20 clerks for computing $x_i V_{Pthro}$.

Estimating volatility (σ_{Pthro}). Volatility represents uncertainty attributed to the likely growing of load. For some computation, we abide to the real options principles in computing volatility: we use the standard deviation of $x_i V_{Pthro}$s due supporting extra operations for a range of load at a particular host (as the range is said to be revealing to the fluctuation in the value). For other computations, we use modeling estimates for volatility, representing uncertainty, with the objective of demonstrating how volatility is said to influence the options results.

Exercise time (t_{Pthro}) and free risk interest rate(r_{Pthro}). As a simulation assumption, we set the exercise time to one year, assuming that the Duke's Bank needs to accommodate the change in one-year time. We set the free risk interest rate to zero (i.e., assuming that the value of money today is the same as that in one year's time).

2.4.3 Results, analysis, and discussion

Flexibility creates real options. Let us consider the flexibility that S_1 provides over S_0, relative to *Pthro*: Consider a scenario, where the likely load is 1042 TOPS. Table 3 shows that 1042 TOPS can be supported by three hosts, if the Duke's architecture is induced with either M_1 (WLS) or M_0 (JacORB). Table 3 shows that for three hosts, supporting 1042 TOPS costs $1488.88 for S_1 when induced with WLS but $243.05 for S_0 when induced with JacORB. The cost is denoted by $C_{eiPthro}$. Supporting 1042 TOPS online is assumed to eliminate manual-overhead and create $x_i Vs$, and computed using eight clerks scenario. Using high volatility modeling assumptions for σ_{Pthro}= 100% for simplicity, Table 3 shows that S_1 adds more value than S_o for three hosts. This is because the cost of implementing both load balancing and fault-tolerance is far from breaking even on S_0 for three hosts. Let us now suppose that Duke's can only afford to invest in three hosts and the investment is to be made. Let us now assume that the load is likely to grow from 1042 TOPS to the range of 1250-1395 TOPS, as a result of accommodating more customers.

According to Table 4, as the load increases over 1042 TOPS, M_1 continues to be of a better value for flexibility as when compared to M_0 for the following reasons: First, S_0 will be inflexible to support an extra operation beyond 1042 TOPS for three hosts (Table 1). That is, the growing load requires an additional host; henceforth, incurring hardware costs. Second, the cost of implementing both load balancing and fault-tolerance is far from breaking even on S_0 for three hosts. As a result, S_0 ceases to create real options on three hosts if the load exceeds the expected 1024 TOPS. Conversely, for the range of 1250-1395 TOPS, S_1 tends to carry growth options on three hosts. This is because at threshold, S_1 can support around 1395 TOPS (Table 1). That is, S_1 when induced with WLS, tends to create value for an additional 371 TOPS on three hosts. Formalizing this thinking,

The architectural potential of S_1 (WLS) = value in supporting 1042 TOPS now + growth options in supporting an additional 371 TOPS;

The architectural potential of S_0 (JacORB) = value in supporting 1042 TOPS now + zero growth options beyond 1042 TOPS.

Table 3. Supporting 1042 TOPS with three hosts and their options value, if the Duke's architecture is induced with either M_1 (WLS) or M_0 (JacORB), σ_{Pthro}= 100%

1042 TOPS	No Hosts	$C_{eiPthro}$	$X_{iVPthro}$	$Options_{Pthro}$
S_1(WLS)	3	148.88	131.61	45.44
S_1(JBOSS)	4	126.96	131.61	51.86
S_0(JacORB)	3	243.05	131.61	27.59

Table 4. Supporting 1395 TOPS with three hosts and their options value, if the Duke's architecture is induced with either M_1 (WLS) or M_0 (JacORB) σ_{Pthro}= 100%

1250-1395 TOPS	No Hosts	$C_{eiPthro}$	$X_{iVPthro}$	$Options_{Pthro}$	Growth Options
S_1(WLS)	3	148.88	176.61	77.05	31.61
S_1(JBOSS)	4	126.96	176.1	85.79	33.93 for 4 hosts
S_0(JacORB)	3	243.05	131.61	27.59	0

Hence, for three hosts and with the likely growing load in the range of 1250-1390 TOPS, S_1 exhibits that it has flexibility under uncertainty. This flexibility takes the form of growth options held on S_1. The value of these options is in supporting an additional 371 TOPS. The more uncertain we are about the likely growth in load (i.e., beyond 1024 TOPS and in the range of 1250-1390 TOPS), the more valuable is the flexibility in S_1 relative to S_0. Real options is suited to address typical software evolution problems, where uncertainty attributed to the change in requirements is the norm. Using real options theory is better suited than techniques that are based on Present Value (PV) and Discount Cash Flow (DCF) as these techniques tend to systematically underestimate the value of flexibility under uncertainty[Trig95; Erd02]. As we have mentioned in several occasions, in our case the likely change in load is the major source of uncertainty that the Duke's Bank faces. To address such uncertainty and provide better insights on value creation, we have appealed to the use of real options theory. Let us provide an evidence to support our use: Let us assume that the load is assumed to be in the range of 30- 50 TOPS. Based on the benchmarks, 30-50 TOPS could be easily addressed by one host using either M_0 (JacORB) or M_1 (Jboss or WLS). Figure 4 sketches the likely associated costs when inducing the architecture with either alternative. For such a low throughput requirements, inducing the architecture with M_0 may appear to be more attractive as when compared to inducing the architecture with M_1 (using either JBOSS or WLS). This is because M_1 incurs license costs for WLS. Moreover, looking at S_1 when induced with JBOSS, S_1 is likely to be in magnitude slower than S_0 as when induced with JacORB. This means that S_1 (JBOSS) will support fewer TOPS and consequently will create less value added per second as when compared to S_0. For this low load, the fault-tolerance and load-balancing services need not be implemented on S_0. If options analysis is not used, M_0 will be a no-brain choice for inducing the Duke's Bank architecture. Though inducing the architecture S_1 with M_1 (using WLS) appears less attractive than M_0 (JacORB), S_1 may still carry embedded growth options which will only materialize if the load grows.

If we use a PV or DCF approach, the resulted valuation will compute the present value as realized and ignore these growth options. In other words, inducing the architecture with WLS if undertaken, PV or DCF would hint that S_1 would destroy value rather than create it. Formulating this argument, a PV approach, for example, will leave us with *Value* S_1 = PV. However, *Value*S_1 is actually *Value* S_1 = PV + *Opt*. That is, M_1 carry embedded growth options, Opt. The *Opt*, if left unexercised, are ignored by the non-options analysis. Hence, *Value* for S_1 is then said to be underestimated. As a result, S_0 may look more attractive (Table 5).

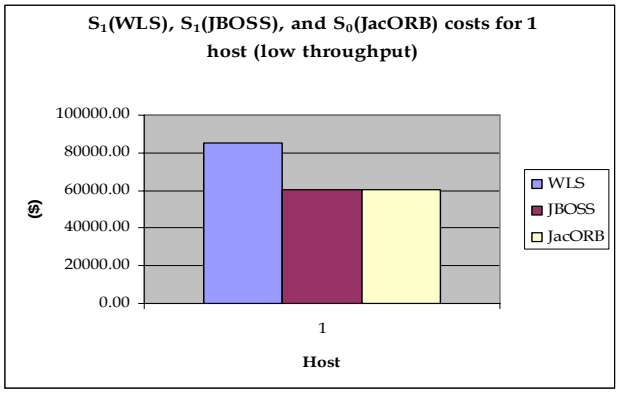

Figure 4. The likely associated costs compared upon inducing Duke's architecture with WLS, JBOSS, and JacORB for very low throughput requirements on one host

The PV and DCF calculation of Table 5 shows that S_1 is the least attractive for this range of load. The computation is based on the benefits of supporting 100 TOPS less their costs. However, the computation ignores the growth options on S_1 in supporting additional 632 TOPS using the first host. Similarly, the PV and DCF systematically undervalue the growth potential of S_1 (JBOSS) and S_0 (JacORB) in respectively supporting 300.26 TOPS and 446.26 TOPS. In other words, PV and DCF ignore the flexibility value of S_1 and S_0 in responding to the growing load at host 1. Note that it is a fact that NPV or DCF does not work well for projects with future decisions that depend on how uncertainty resolves. Though they can be used to evaluate the operational benefits in a stable environment with well-understood and measurable costs and benefits, they have little to offer when capturing additional value due to flexibility under uncertainty, such as strategic opportunities and the ability to respond to changing conditions. Using PV or DCF, S_1, when induced with WLS, reports negative values upon inducing the architecture with WLS for this range of load. However, the situation indicates that these results underestimate the value of S_1, as S_1 can better respond to uncertainty, where the load is likely to grow over 100 TOPS. In Table 6, we have turned to the intuition and used ArchOptions to capture the growth options on S_1 and S_0. The volatility parameter is an expression of the range of "benefits" at a host. For example, consider S_1 (WLS): the benefits could "wander" from zero (i.e., idle state with no operations executing at a second) to the benefits derived from full utilization of capacity (i.e., in the support of 732 TOPS). That is, the volatility of 66% for S_1 (WLS) indicates that the benefits of executing the TOPS is in the range of \$0(idle) to \$92.42(full utilization) per second on host 1. Similarly, for S_0 (JacORB): the 45% volatility for S_0 (JacORB) indicates that the benefits of executing the TOPS are in the range of \$0(idle) to \$69.04 (full utilization) per second on host 1. As far as the options on S_1(WLS) are concerned, S_1 has "pulled" the options on one host for this range of load. This is because we have accounted for the possible fluctuation in the derived values from supporting the TOPS. Considering such "fluctuation" provides us with better insights on the architectural potential of S_1 in support of this likely change in load. Table 6 suggests S_1 has reported a value added of \$0.017 on 1 host.

Table 5. Illustration NPV and DCF per second ($) very low throughput scenario (100 TOPS)

100 TOPS	Max TOPS	$C_{eiPThro}$	$X_{iVPThro}$	PV	DCF	Value Ignored (TOPS)
S_1(WLS)	732.00	853.11	12.63	-840.48	-933.87	-632
S_1(JBOSS)	400.26	603.11	12.63	-590.48	-656.09	-300.26
S_0(JacORB)	546.80	603.11	12.63	-590.48	-656.09	-446.80

Table 6. Illustration options per second ($) very low throughput scenario (100 TOPS)

100 TOPS	$C_{eiPThro}$	$X_{iVP-Thro}$	σ_{Pthro}	Options	Actual Value (TOPS)
S_1(WLS)	853.11	92.424	66%	0.01700	100 + 632
S_1(JBOSS)	603.11	50.53	35%	0+	100 + 300.26
S_0(JacORB)	603.11	69.04	49%	0.00001	100 + 446.80

The impact of volatility on value. A critical difference between PV/DCF and real options is the effect of uncertainty (or risk) on value. Figures 5a-c shows that PV and DCF systematically underestimate the potential value of S_1 and S_0 in supporting a range in load on one to four hosts. The reason why DCF reports steeper values is due to the discount rate (10% is used for illustration purposes only). We have turned to the intuition and have used a more powerful technique offered by the theory of option pricing to capture the value of flexibility under the dynamic and the uncertain range of load. However, how this uncertainty is expressed? How does this relate to Duke's case? Let us have a close look at the impact of the volatility parameter, which is an expression of the value of flexibility under uncertainty.

In the context of ArchOptions, the volatility parameter estimates the "cone of uncertainty" in the future value of the asset, rooted as its current value and extending over time as a function of volatility. As volatility increases, total uncertainty around the benefits also increases. The more TOPS a host is likely to support, the more likely that the actual benefits to "wander" up and down and deviate from the expected present value if the load grows. Hence, the more volatile the environment is said to be.

Let us now assume that Duke's Bank needs to support more customers. Assume that the load is likely to grow and be in the range of 600- 686 TOPS (Table 7): S_1, when induced with WLS, realizes the change in load by one host. S_0, when induced with JacORB, will need two hosts and will incur the cost of developing the fault-tolerance and load-balancing services on the structure. Yet, S_1 when induced with JBOSS will require three hosts and will incur additional hardware costs for completing the 686 TOPS. Figure 6 shows a scenario for a likely load of 600-686 TOPS for S_1 when induced with WLS and for S_0 when induced with JacORB. S_1 could be regarded as an investment with a wide range of possible outcomes. However, S_0 is an investment with a relatively narrower range. For S_1, the investment is said to be more volatile. This is because S_1 can support more TOPS/host resulting in a possible range of values. Relating this to PV, this means that there is a chance of producing positive PV in the future. Hence, a real option under this set of outcomes would have value. As for the S_0, the valuation under this scenario is more stable. This is because S_0 can support at most 686 TOPS for the existing configuration. This means that S_0 has no chance of producing a project with a positive NPV beyond 686 TOPS. That is an option using the latter set of outcomes would have no value.

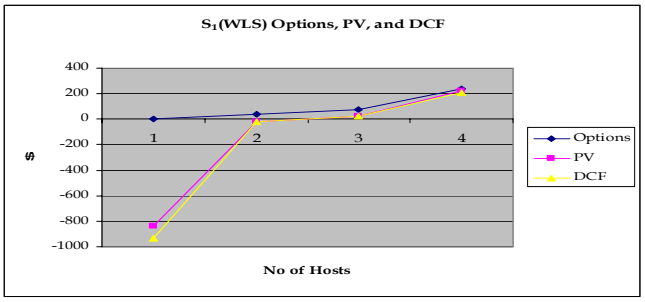

Figure 5a. The options, PV, and DCF on S_1 when induced with WLS relative to the throughput valuation point of view

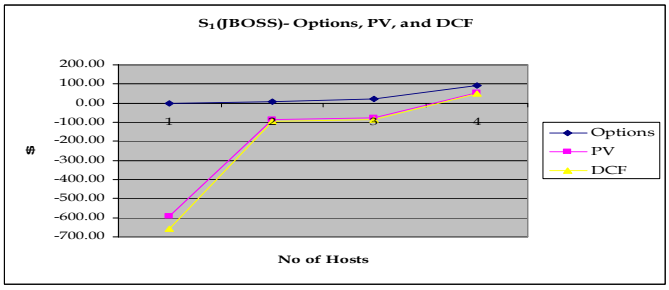

Figure 5b. The options, PV, and DCF on S_1 when induced with JBOSS relative to the throughput valuation point of view

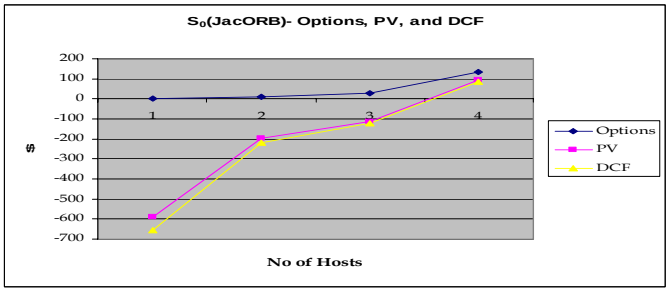

Figure 5c. The options, PV, and DCF on S_0 when induced with JacORB relative to the throughput valuation point of view

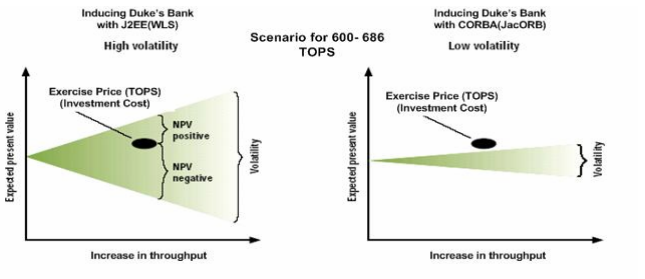

Figure 6. Impact of volatility on value

Table 7. PV and DCF ($) per second for supporting 686 TOPS on S_0 and S_1 and the values they ignore

686TOPS	No Hosts	Max TOPS	$C_{elPThro}$	$X_{IyPThro}$	PV	DCF	Value Ignored (TOPS)
S_1(WLS)	1	732	124.36	216.54	92.18	83.80	-46 TOPS
S_1(JBOSS)	3	763	193.51	216.54	23.03	20.93	-77 TOPS
S_0(JacORB)	2	686	285.32	216.54	-68.78	-76.42	0 TOPS

Table 8. Adjusted PV and the options in ($) per second under full utilization scenario of hosts for load greater than 686 TOPS on S_0 and S_1 and the values added per second

Full Utilization	No Hosts	C_{dPThro}	$X_{IyPThro}$	σ_{Pthro}	PV prior Adjustment	Adjusted PV	Options Predicted($)	Added Value	Actual Value (TOPS)
S_1(WLS)	1	124.36	231.06	10.52%	92.18	106.7	106.7	14.52	686 TOPS Plus 46 TOPS
S_1(JBOSS)	3	193.51	240.85	6.9%	23.03	47.34	47.34	24.34	686 TOPS Plus 77 TOPS
S_0(JacORB)	2	285.32	216.54	0%	-68.78	-68.78	0	0	0 TOPS

Let us now assume that we have induced the Duke's architecture with M_1 (WLS) for one version and M_0 (JacORB) for the other. Hence, investment is made. As time passes, let us assume that an increase in load materializes. As change in load materializes, uncertainty is assumed to be resolved. Thus, the present value, as a result of supporting more TOPS (analogous to the future value of a stock), can be then calculated more accurately. If we examine the PV of this scenario, we can see that PV reports $92.18/second for WLS for 686 TOPS. That is, this is equal to the benefits minus the costs of completing the 686 TOPS. However, this value is said to be underestimated, as it ignores the additional 46 TOPS that S_1 can support using one host (i.e., 732 minus 46 TOPS). S_1, when induced with JBoss, reports a PV of $23.03, ignoring the additional value of supporting 77 TOPS for this configuration. S_0, when induced with JacORB, reports a negative PV. The negative value is attributed to cost incurred upon the development of the fault tolerance and the load balancing services on S_0. Let us now turn to options: Table 8 suggests that for 686 TOPS, S_1, when induced with WLS, creates more options than S_0 using one host. In particular, S_1 (WLS) reports a value of $106.7. S_1 (JBOSS) reports a value of $47.3. S_0 (JacORB) reports a value of $0. Why is this difference? Technically speaking, this is because of the volatility parameter that captures variation in the value potentials of the said structures. For S_1 (WLS), the difference for S_1 (WLS) is attributed the range of possible returns that the additional 46 could ascribe to S_1(WLS). This means that for S_1 (WLS), the additional future values, if the range in load changes, is in the bound of $0(i.e., at most 686 TOPS) to $46*216.54/686(i.e., assuming equal returns upon supporting the additional 46TOPS). This will leave us with a volatility of %10.52, using the standard deviation of the returns over this bound. Similar argument applies for S_1 (JBOSS), leaving us with a volatility equal to 6.9% in support of the additional 77 TOPS. S_0 (JacORB) reports $0 options. This is because S_0 (JacORB) cannot support additional TOPS on this structure. In the language of options, S_0 (JacORB) is not volatile and ceases to create options beyond 686 TOPS; henceforth, the reported zero values.

Let us now turn to PV again and assume an additional load has materialized (i.e., uncertainty has been resolved). Let us adjust the PV based on the new information at hand: if we compute the PV of the additional 46 TOPS for S_1 (WLS), this will leave us with an added value of $14.52 over the previously computed PV, as reported in Table 8. If we compute the PV of the additional 77 TOPS for S_1 (JBOSS), this will leave us with an added value of $24.34 over the previously computed PV for S_1 (JBOSS)- see Table 8. Adjusting the PV, we sum these values with the previously reported PVs of Table 8. This will leave us with $106.7 value for S_1 (WLS) and $47.3 value for S_1 (JBOSS). Henceforth, this is a match with the ArchOptions results for S_1 (WLS) and S_1 (JBOSS).

This observation leaves us with following conclusions: First, though it is still possible to adjust PV or DCF techniques for capturing the options, ArchOptions provides us with a ready and closed-form solution, rooted in options theory, for capturing the value of flexibility under uncertainty on a given architecture. This solution is said to be superior to PV and DCF, as the latter they systematically underestimate the value of the flexibility of an architecture under uncertainty. Secondly, the analysis of matching the adjusted PV values with that of ArchOptions confirms the correctness and the effectiveness of the model. Nevertheless, the effectiveness of ArchOptions is essentially rooted in our use of Black and Scholes options theory. The analysis, however, has established confidence on both its correctness and effectiveness. Third, the results of this observation show that the volatility parameter is critical for the valuation of the options. In real situations, the performance analyst/architect may inspect available performance benchmarks, screen historical load-trends to predict future ones, or use prototypes of partial implementations to collect performance indices. Consequently, volatility can be then empirically extracted. The analyst can make use of the sensitivity analysis for better understanding of the impact of throughput on the value added when uncertainty in the likely future load dominates.

3. RELATED WORK

In this Section, we provide a quick overview of closely related research on: (i) the use of real options in software design and engineering; (ii) related research on architectural evaluation, and (iii) ongoing research on the "coupling" of software architecture and middleware.

The use of real options in software engineering. Economics approaches to software design appeal to the concept of static Net Present Value (NPV) as a mechanism for estimating value [Boe00]. These techniques, however, are not readily suitable for strategic reasoning of software development as they fail to factor flexibility [Boe00; Erd99]. The use of strategic flexibility to value

software design decisions has been explored in, for example, [Erd99; Erd02; Erd00; Sul96; Sul99; Sul01] and real options theory has been adopted to value the strategic flexibility: [Bal01] studied the flexibility created by modularity in design of components (of computer systems) connected through standard interfaces. [Sul96; Sul99; Sul01] pioneered the use of real options in software engineering. [Sul96; Sul99] suggested that real options analysis can provide insights concerning modularity, phased projects structures, delaying of decisions and other dynamic software design strategies. [Sul99] formalized that option-based analysis, focusing in particular on the flexibility to delay decisions making. An interesting approach that has inspired the early stages of our work is that of [Sul01]. [Sul01] extended [Bal01] that is developed to account for the influence of modularity on the evolution of the computer industry. [Sul01] use the model developed in [Bal01] to treat the "evolovability" of software design using the value of strategic flexibility. Specifically, they argued that the structure and value of modularity in software design creates value in the form of real options. A module creates an option to invest in a search for a superior replacement and to replace the currently selected module with the best alternative discovered, or to keep the current one if it is still the best choice. The value of such an option is the value that could be realized by the optimal experiment-and-replace policy. Knowing this value can help a designer to reason about both investment in modularity and how much to spend searching for alternatives. [Erd99] describes how strategic flexibility in software development, involving COTS components, can be valued using real options. An interesting use of real options theory is that of [Erd02]. [Erd02] uses real options to value the inherent flexibility in the Extreme Programming (XP), where they have considered XP as a lightweight process that is well positioned to respond to change and future opportunities; hence, creating more value than a heavy-duty process that tends to freeze development decisions.

Architectural evaluation. Interested reader may refer to [Bah03b] in which we provide a comprehensive survey on architectural evaluation methods. In short, we have distinguished between two classes of software architecture evaluation methods: (i) general-purpose methods(e.g., ABAS[Kle99], ATAM[Kaz98], SAAM[Kaz94]) that evaluate software architectures for qualities that need to be met by the system (e.g. performance, security, and modifiability) and (ii) an emerging class of methods that explicate evaluation for stability and evolution. Existing methods to architectural evaluation have ignored any economic considerations, with CBAM [Kaz01] being the notable exception. The evaluation decisions using these methods tend to be driven by ways that are not connected to, and usually not optimal for value creation. Factors such as flexibility, time to market, cost and risk reduction often have higher impacts on value creation [Boe00]. Hence, flexibility is in the essence. In our work, we link flexibility to value, as a way to make the value of scalability tangible.

Relating CBAM to our work, the following distinctions can be made: with the motivation to analyse the cost and benefits of architectural strategies, where an architecture strategy is subset of changes gathered from stakeholders, CBAM does not appeal to the analysis of the value of scalability or the architectural strategies responsible for realising scalability in an architecture. Further, CBAM does not tend to capture the long-term and the strategic value of the specified strategy. ArchOptions, in contrast, views flexibility as a strategic architectural quality that adds to the architecture values in the form of *growth options.* When CBAM complements ATAM [Kaz98] to reason about qualities related to change such as modifiability, CBAM does not supply rigorous predictive basis for valuing such impact. Plausible improvements of the existing CBAM include the adoption of real options theory to reason about the value of postponing investment decisions. CBAM uses real options theory to calculate the value of option to defer the investment into an architectural strategy. The delay is based on cost and benefit information. In the context of the real options theory, CBAM tends to reason about the *option to delay* the investment in a specific strategy until more information becomes available as other strategies are met. In contrast, we uses real options to value the flexibility provided by the architecture to expand in the face of evolutionary requirements; henceforth, referred to as the options to expand or growth options.

On the "coupling" of software architectures and middleware. There is only very little work on the "coupling" of middleware and software architectures. Notable exceptions include [Jaz95; Gal97; Sul97; Ore98; DiN99; Met00; Med03; Den04].

[Jaz95] explores the relationship between software architectures and component technologies. [Gal97] have looked at an existing component framework, the C ++ standard library, and identified the architectural style induced. [Sul97] claims that for a system to be implemented in a straightforward manner on top of a middleware, the corresponding architecture has to be compliant with the architectural constraints imposed by the middleware. [Ore98] discuss the importance of complementing *component interoperability models* with explicit architectural models. [DiN99] devised the term *middleware-induced architectural styles* and used Architecture Definition Languages (ADLs) to describe the assumptions and constraints that middleware infrastructures impose on the architecture of system. [Met00] proposed a classification framework of software connectors and described types of services provided by connectors for enabling and facilitating component interactions. They aim at building implementation topologies (e.g., bridging of middleware) that preserve the properties of the original architecture. [Med03] stated the idea of "coupling" the modelling power of software architectures with the implementation support provided by middleware. They have investigated the possibility of defining systematic mappings between architectures and middleware. [Den04] measured performance attributes of an architecture based on the early available implementation support provided by the middleware.

In summary, research effort on the relation between software architectures and middleware has been motivated by pragmatic needs. The effort has revolved on issues such as investigating the compliancy of architectural styles with middleware; capabilities that the middleware and the architecture can bring when "coupled" to understand quality attributes of the system such as performance; mapping between middleware and software architectures; and semantics and syntactical issues related to the mapping process. As it has been noted in several occasions [Emm00b; Emm02], research on software architectures has over-emphasized functionality and not sufficiently addressed how global properties and non-functional requirements are achieved in an architecture, where these requirements cannot be attributed to individual components or connectors. Moreover, no notable effort has been devoted on understanding the economics of non-functional requirements in relation to both architecture and middleware, when coupled. Our use of architectural flexibility and its value is novel and only a step toward such an understanding using a *value-based* [EDSR 1-8] reasoning.

4. CONCLUSIONS

We have reported on how ranges in which scalability requirements change can inform the selection of distributed components technology and subsequently the selection of application server products. As the exact method for analyzing scalability is subject to debate, we have identified two views to the analysis of scalability and have focused the analysis on throughput for valuing scalability. We have applied ArchOptions, a real options-based model, to the problem of valuing the flexibility of the architecture in scaling up two versions of a given architecture, each induced with a distinct middleware: one with CORBA and the other with J2EE. The results show that the application of real options theory is said to be superior to the application of traditional techniques, such as Net Present Value, for estimating value in software. The contribution demonstrates that using value-based reasoning, we can value scalability and support the development (evolution) of software systems that need to adapt to the inevitable evolving requirements.

5. REFERENCES

[Bah03a] Bahsoon, R. and Emmerich, W.: Evaluating Software Architectures: Development, Stability, and Evolution. In: Proc. of IEEE/ACS Computer Systems and Applications, IEEE CS Press (2003a) 47-57

[Bah03b] Bahsoon, R. and Emmerich, W.: ArchOptions: A Real Options-Based Model for Predicting the Stability of Software Architecture. In: Proc. of the 5th Workshop on Economics-Driven Software Engineering Research, with the 25th Int. Conf. on Software Engineering, IEEE CS (2003b) 35-40

[Bah03c] Bahsoon, R.: Evaluating Software Architectures for Stability: A Real Options Approach. In: Proc. of the Doctoral Symposium of the 25th Int. Conference on Software Engineering, IEEE CS Press (2003)

[Bah04a] Bahsoon, R. and Emmerich, W.: Evaluating Architectural Stability with Real Options Theory. In: Proc. of the 20th IEEE Int. Conf. on Software Maintenance, IEEE CS Press (2004a) 443-447

[Bah04b] Bahsoon, R. and Emmerich, W.: Applying ArchOptions to Value the Payoff of Refactoring. In: Proc. of the Sixth Workshop on Economics-Driven Software Engineering Research, with the 26th Int. Conf. on Software Engineering, IEE Press (2004b) 66-70

[Bah05a] Bahsoon, R.: Evalauting Architectural Stability with Real Options Theory, PhD thesis, U. of London, UK (2005)

[Bah05b] Bahsoon, R., Emmerich, W., and Macke, J.: Using ArchOptions to Select Stable Middleware-Induced Architectures. In: IEE Proceedings Software, Special issue on Relating Requirements to Architectures, IEE Press 152(4) (2005) 176-186

[Bald01] Baldwin, C. Y., and Clark, K.B.: Design Rules - The Power of Modularity. MIT Press (2001)

[Bla73] Black, F., and Scholes, M.: The Pricing of Options and Corporate Liabilities. Journal of Political Economy. U. of Chicago Press (1973) 637-654

[Boe00] Boehm, B., and Sullivan, K. J.: Software Economics: A Roadmap. In: A. Finkelstein (ed.): The Future of Software Engineering. ACM Press (2000) 320-343

[Coo01] Cook, S., Ji, H., and Harrison, R.: Dynamic and Static Views of Software Evolution. In: Int. Conf. on Software Maintenance, Florence, Italy. IEEE CS (2001) 592-601

[Den04] Denaro, G., Polini A., and Emmerich W.: Performance Testing of Distributed Component Architectures. In: S. Beydeda and V. Gruhn (eds.), Building Quality into COTS Components - Testing and Debugging, Springer (2004) 294-314

[DiN99] Di Nitto, E., and Rosenblum, D.: Exploiting ADLs to Specify Architectural Styles Induced by Middleware Infrastructures. In: Proceedings of the 21st Int. Conference on Software Engineering, ACM Press (1999) 13-22

[EDS99-06] EDSER 1-8: Proceedings of the Workshops on Economics-Driven Software Engineering Research: In conj. with the 21st through 28th International Conference on Software Engineering (1999 - 2006)

[Emm00a] Emmerich, W.: Engineering Distributed Objects. John Wiley & Sons, Chichester, UK (2000a)

[Emm00b] Emmerich, W.: Software Engineering and Middleware: A Road Map. In: A. Finkelstein (ed.), Future of Software Engineering, ACM Press (2000b) 117-129

[Emm02] Emmerich, W.: Distributed Component Technologies and their Software Engineering Implications. In: Proc. of the 24th Int. Conf. on Software Engineering, Orlando, Florida, ACM Press (2002) 537-546

[Erd00] Erdogmus, H.: Value of Commercial Software Development under Technology Risk. The Financier 7(2000)

[Erd02] Erdogmus, H., Boehm, B., Harrison, W., Reifer, D. J., and Sullivan, K. J.: Software Engineering Economics: Background, Current Practices, and Future Directions.Tutorial Summary. In: Proc. of 24th Int. Conf. on Software Engineering, ACM Press (2002) 683-684

[Erd99] Erdogmus, H., and Vandergraaf. J: Quantitative Approaches for Assessing the Value of COTS-Centric Development. In: the Proc. of the Sixth International Symposium on Software Metrics (METRICS' 99), Boca Raton, FL, IEEE CS Press (1999) 279-290

[Fin00] Finkelstein, A.: Architectural Stability. http://www.cs.ucl.ac.uk/staff/a.finkelstein/talks.html (2000)

[Gal97] Gall, H., Jazayeri, M., Klösch, R., and Trausmuth, G.: The Architectural Style of Component Programming. COMPSAC, IEEE CS Press (1997) 18-27

[Jaz95] Jazayeri, M.: Component Programming - a Fresh Look at Software Components, In: 5th European Software Engineering Conference. Lecture Notes in Computer Sc, Springer (1995) 457-478

[Kaz01] Kazman, R., Asundi, J., and Klein, M.: Quantifying the Costs and Benefits of Architectural Decisions. In: Proc. of 23rd Int. Conf. on Software Engineering, IEEE CS Press (2001) 297-306

[Kaz94] Kazman, R., Abowd, G., Bass, and L., Webb, M.: SAAM: A Method for Analyzing the Properties of Software Architectures. In: Proc. of 16th Int. Conf. on Software Engineering, IEEE CS (1994) 81-90

[Kaz98] Kazman, R., Klein, M., Barbacci, M., Lipson, H., Longstaff, T., and Carrière, S.J.: The Architecture Tradeoff Analysis Method. In: Proc. of 4rth. Int. Conf. on Engineering of Complex Computer Systems IEEE CS Press (1998) 68-78

[Kle99] Klein, M., and Kazman, R.: Attribute-Based Architectural Styles.CMU/SEI-99-TR-22, Software Engineering Institute(1999)

[Med03] Medvidovic N., Dashofy E., and Taylor R.: On the Role of Middleware in Architecture-based Software Development. Int. Journal of Software Engineering and Knowledge Engineering, 13(4) (2003) 229-306

[Med97] Medvidovic, N., and Taylor, R.: A Framework for Classifying and Comparing Architecture Description Languages. In: Proc. of 6th. European Software Engineering Conf., with the Fifth ACM SIGSOFT Symp. on the Foundations of Software Engineering, ACM Press (1997)60-76

[Meh00] Mehta, N., Medvidovic, N., and Phadke, S.: Towards a Taxonomy of Software Connectors. In: Proc. of the 22nd International Conference on Software Engineering, ACM Press (2000) 178-187

[OMG00] Object Management Group: The Common Object Request Broker: Architecture and Specification, 2.4 ed., OMG (2000)

[Orei98] Oreizy, P., Medvidovic, N., Taylor, R., and D. Rosenblum, D.: Software Architecture and Component Technologies: Bridging the Gap. In Digest of the OMG-DARPA-MCC Workshop on Compositional Software Architectures, Monterey, CA (1998)

[Sul01] Sullivan, K.J., Griswold, W., Cai, Y., and Hallen, B.: The Structure and Value of Modularity in Software Design. In: the Proceedings of the ninth ESEC/FSE, Vienna, Austria (2001) 99-108

[Sul96] Sullivan, K. J.: Software Design: The Options Approach. In: the Proc. of the Second Int. Software Architecture Workshop. Joint Proceedings of the SIGSOFT '96 Workshops, San Francisco, CA (1996) 15–18

[Sul97] Sullivan, K. J., Socha, J., and Marchukov, M.: Using Formal Methods to Reason about Architectural Standards. In: Proc. of the 19th Int. Conf. on Software Engineering, ACM Press (1997) 503-513

[Sul99] Sullivan, K. J.: Chalasani, P., Jha, S., and Sazawal, V.: Software Design as an Investment Activity: A Real Options Perspective. Real Options and Business Strategy: Applications to Decision-Making. In: Trigeorgis L. (ed.) Risk Books (1999) 215-260

[Sun] Sun Microsystems Inc.: Duke's bank application, http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Ebank.html

[Sun02] Inc Sun MicroSystems Inc: Enterprise JavaBeans Specification v2.1 (June 2002)

[Tri95] Trigeorgis, L.: Real options in Capital Investment: Models, Strategies, and Applications. Praeger Westport, London (1995)