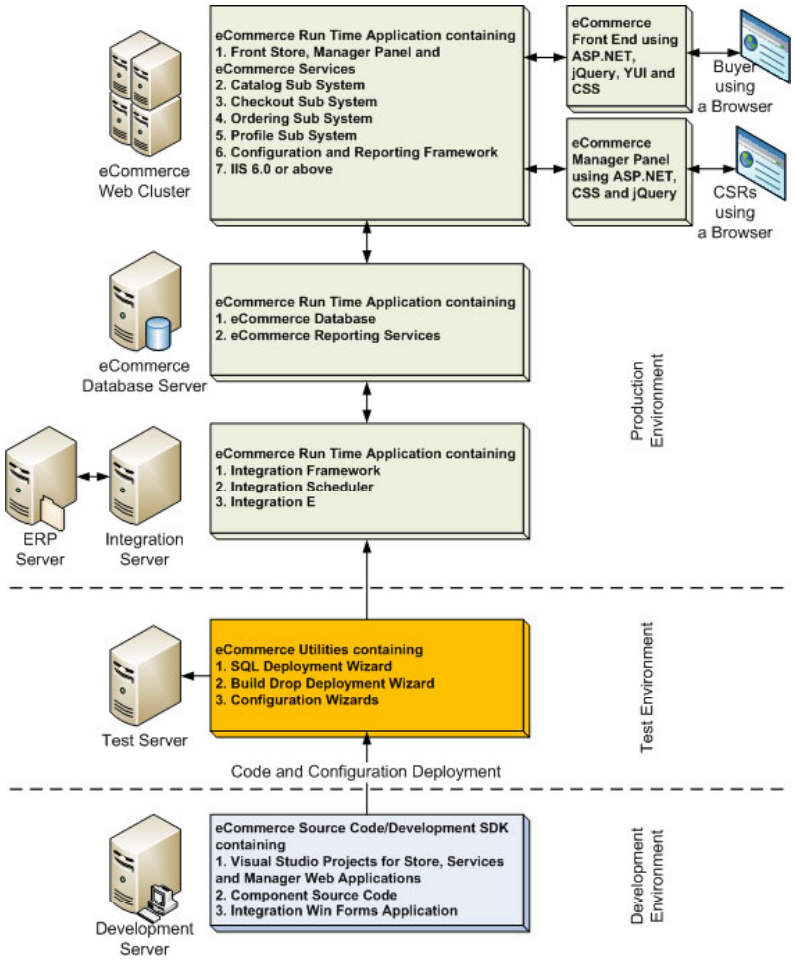
CPSC 875

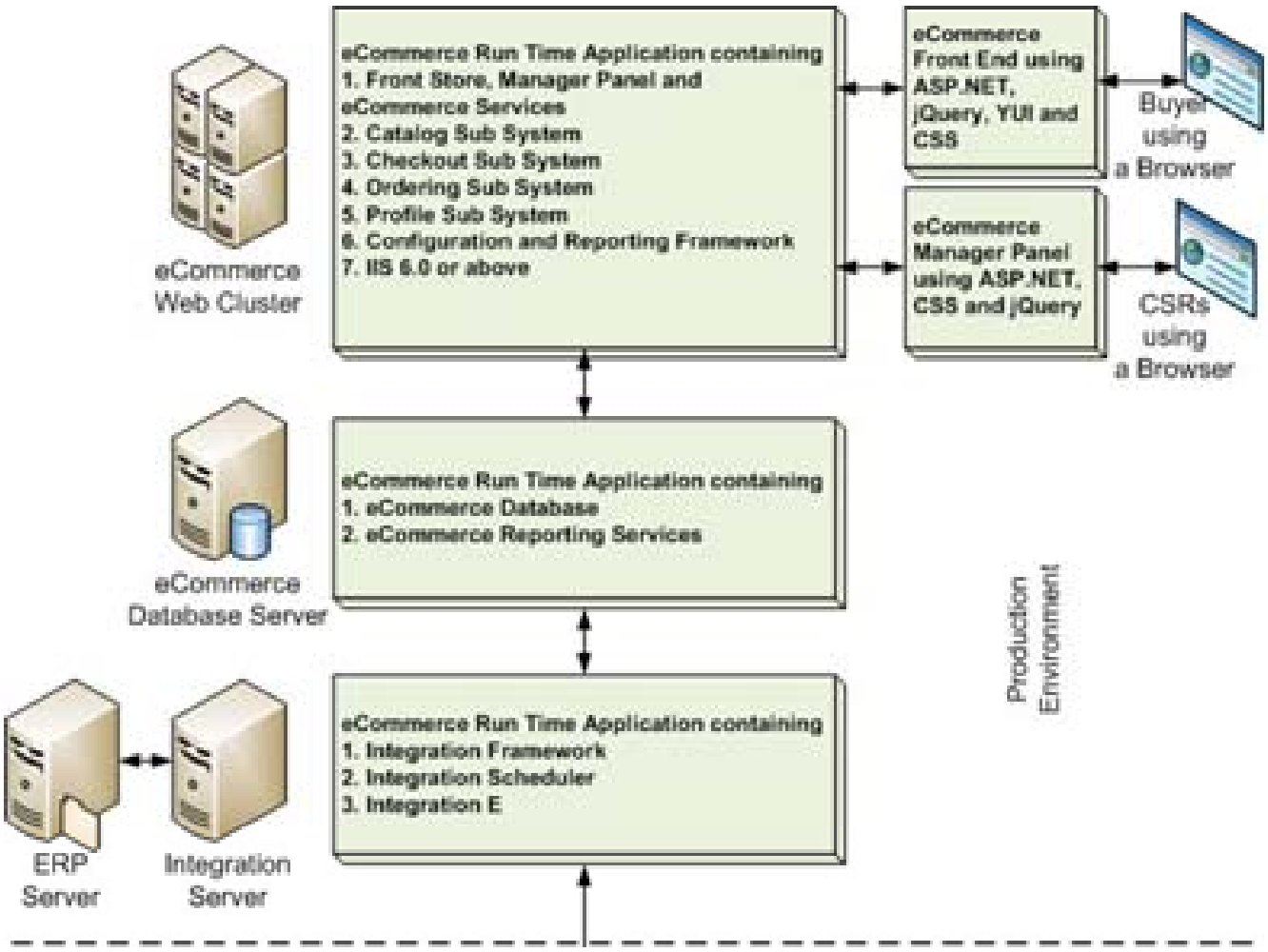
John D. McGregor

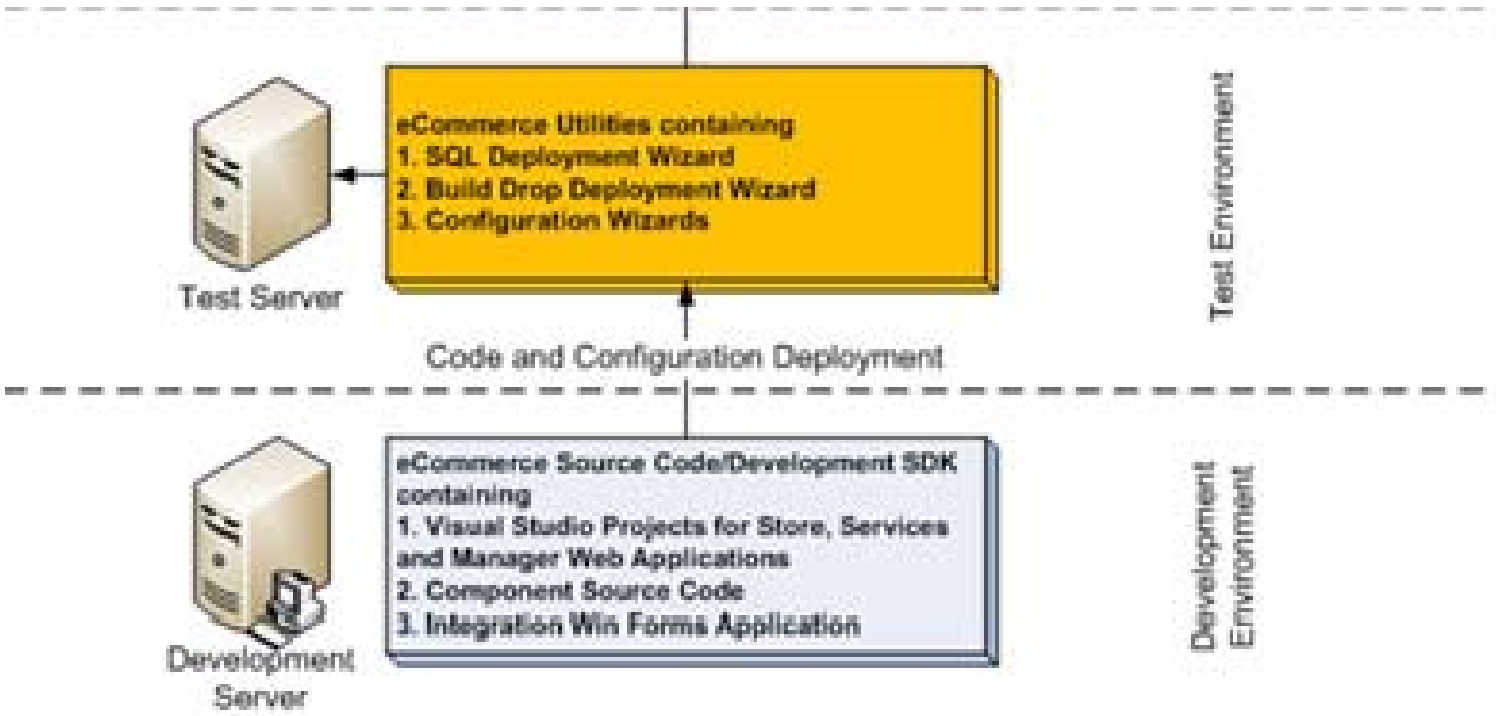
Risk, Uncertainty, and Options

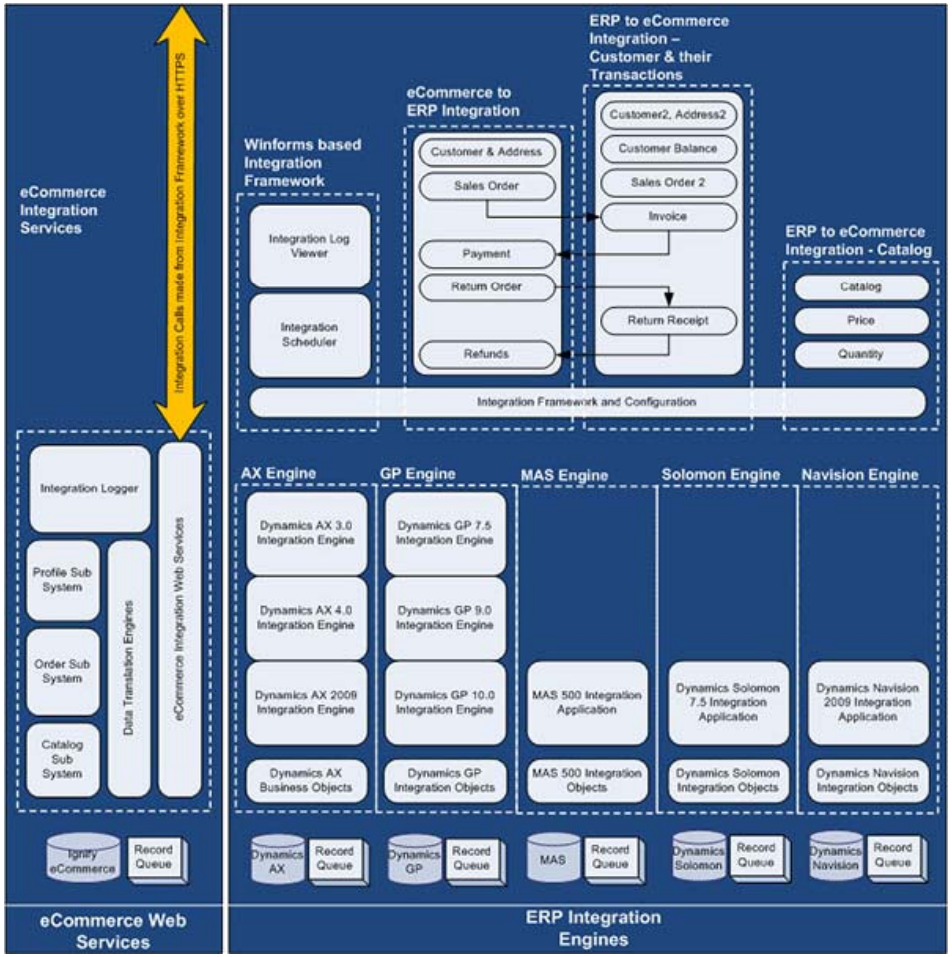
Dynamic environment



http://www.ignify.com/Ignify-eCommerce-Technical-Architecture.asp


eCommerce Run Time Application containing
1. Front Store, Manager Panel and eCommerce Services
2. Catalog Sub System
3. Checkout Sub System
4. Ordering Sub System
5. Profile Sub System
6. Configuration and Reporting Framework
7. IIS 6.0 or above
eCommerce Front End using ASP.NET, jQuery, YUI and CSS
Buyer using a Browser
eCommerce Manager Panel using ASP.NET, CSS and jQuery
CSRs using a Browser
eCommerce Web Cluster
eCommerce Run Time Application containing
1. eCommerce Database
2. eCommerce Reporting Services
eCommerce Database Server
eCommerce Run Time Application containing
1. Integration Framework
2. Integration Scheduler
3. Integration E
ERP Server
Integration Server
Production Environment


eCommerce Utilities containing
1. SQL Deployment Wizard
2. Build Drop Deployment Wizard
3. Configuration Wizards
Test Server
Test Environment
Code and Configuration Deployment
eCommerce Source Code/Development SDK containing
1. Visual Studio Projects for Store, Services and Manager Web Applications
2. Component Source Code
3. Integration Win Forms Application
Development Server
Development Environment


eCommerce Integration Services
Integration Calls made from Integration Framework over HTTPS
Integration Logger
Profile Sub System
Order Sub System
Catalog Sub System
Data Translation Engines
eCommerce Integration Web Services
Ignity eCommerce
Record Queue
eCommerce Web Services
Winforms based Integration Framework
Integration Log Viewer
Integration Scheduler
eCommerce to ERP Integration
Customer & Address
Sales Order
Payment
Return Order
Refunds
ERP to eCommerce Integration – Customer & their Transactions
Customer2, Address2
Customer Balance
Sales Order 2
Invoice
Return Receipt
ERP to eCommerce Integration - Catalog
Catalog
Price
Quantity
Integration Framework and Configuration
AX Engine
GP Engine
MAS Engine
Solomon Engine
Navision Engine
Dynamics AX 3.0 Integration Engine
Dynamics AX 4.0 Integration Engine
Dynamics AX 2009 Integration Engine
Dynamics AX Business Objects
Dynamics GP 7.5 Integration Engine
Dynamics GP 9.0 Integration Engine
Dynamics GP 10.0 Integration Engine
Dynamics GP Integration Objects
MAS 500 Integration Application
MAS 500 Integration Objects
Dynamics Solomon 7.5 Integration Application
Dynamics Solomon Integration Objects
Dynamics Navision 2009 Integration Application
Dynamics Navision Integration Objects
Dynamics AX
Record Queue
Dynamics GP
Record Queue
MAS
Record Queue
Dynamics Solomon
Record Queue
Dynamics Navision
Record Queue
ERP Integration Engines

Risk-1

- An event that could happen and if it did happen would cause a loss of value is a risk.
- There are two major lines of mitigation:
  - Reduce the probability of occurrence
  - Reduce the cost if it does occur
- There is a risk that the packet processing software will miss its performance target resulting in loss of packets, unclear speech, and misunderstandings. Mitigation: simulate the architecture, measure actions required to process packets, propose alternative design, simulate and see if fewer actions are needed.

Risk-2

- One mitigation tactic is to design experiments.
- More viable for software than hardware
- To avoid impacting the project schedule the entire region that might be affected by a design decision can be factored into a module that can be replaced.
- Then different designs can be created, measured, and the winner inserted without the rest of the design being impacted.

Risk - 3

- Every experiment is the basis for an option
- We can afford to run several experiments
- But we need a notion of value
- There is no point in spending more than the design is worth
- Also there is the cost of a production quality implementation once the choice is made

Acknowledgement

Quality-Attribute-Based Economic Valuation of Architectural Patterns

Ipek Ozkaya
Rick Kazman
Mark Klein

May 2007

TECHNICAL REPORT
CMU/SEI-2007-TR-003
ESC-TR-2007-003

Software Architecture Technology Initiative

http://www.sei.cmu.edu/reports/07tr003.pdf

Definitions

An *option*, a financial concept, is the right, but not the obligation, to take an action in the future when there is uncertainty. *Real options* apply this concept to real-life investments such as manufacturing plants, information technology investments, and product research and development

A central idea in our work is that an architectural decision, such as the application of a pattern, is analogous to a *financial derivative*. In financial markets, a derivative is a financial instrument whose value depends on, or derives from, the values of basic underlying assets [Hull 2006]. For example, a stock option is a derivative whose value is dependent on the price of the stock. The stock is the underlying asset for the option. We will show how quality attributes and their expected utility in the face of uncertainty act as underlying assets for the valuation of design decisions, similar to the valuation of financial derivatives.

Patterns

Architectural patterns are intended to aid in creating architectures that meet quality attribute requirements[1] [Buschmann 1996, Bass 2003]. Such patterns are employed by architects to achieve a desired quality attribute behavior, which, in turn, imparts utility to the architecture. We can analyze those patterns to estimate the future value of a system in the face of uncertainty based on the utility of the quality attributes that can be achieved from the patterns' application and based on a prediction of how much the market will value that level of utility.

Questions addressed

- How can architectural patterns be evaluated for their real option values?
- Based on such an evaluation, can suggestions about *when* to apply which pattern be generated in order to support architectural evolution?
- Can the valuation of architectural patterns reveal key insights for architectural decision making with respect to quality attributes and business goals?

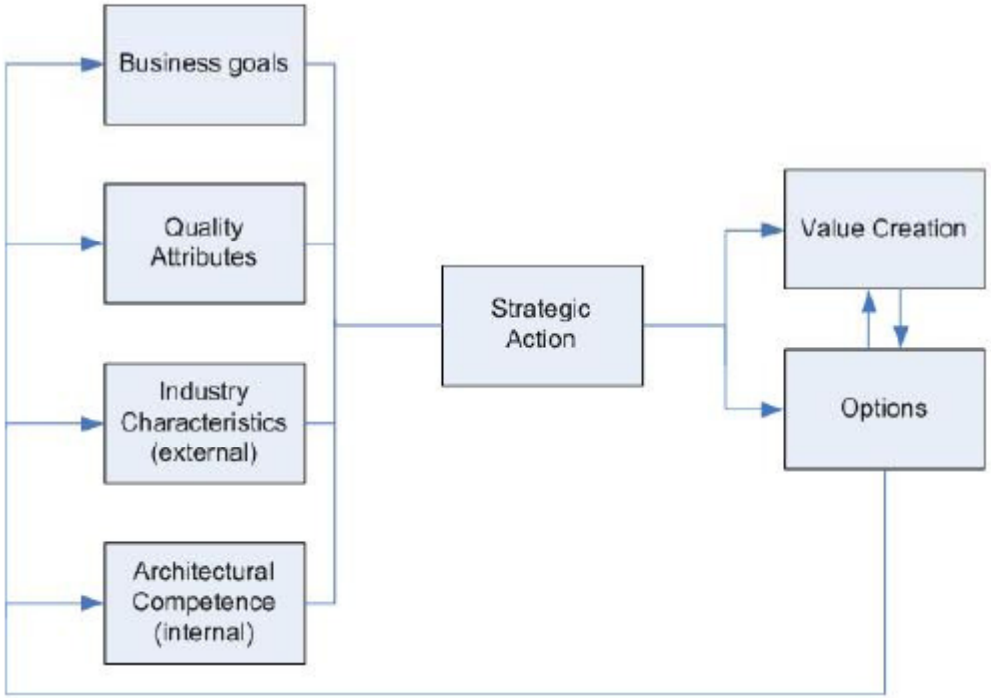
Components of real options

The components of real options analysis include

- the decision to be made
- a characterization of the uncertainty
- the decision rule

The decision rule is a simple mathematical expression that indicates when the decision should be made and helps identify the critical parameters that architects need to observe during the decision-making process.

For example, in the case above, what is the value of making the architecture more modifiable or perform better? If the system's modifiability is poor and new features take a long time to implement, the utility of increased modifiability might be substantially higher. If systems based on this architecture already have adequate performance, the marginal utility of performance improvements may be low. But even this information is insufficient to make a truly informed architectural decision. The key to such a decision is to understand the *potential future value* of increased modifiability or performance, the *costs* of achieving that potential value (in terms of architectural strategies pursued today), and the *likelihood* that such an increase in modifiability or performance will be needed.



Adapted from Credit Suisse First Boston's Strategy Formulation and Real Options Model into Architecting [Maubousin 1999]

Figure 1: Framework for Options Analysis in Software Architecting

Approaches to real options

- Black-Scholes-Merton, which involved work originally done by Black-Scholes [Black 1972] and later extended by Merton [Merton 1998]
- binomial analysis [Amram 1999, Hull 2006, Cox 1979]
- Baldwin and Clark's more recent Net Option Valuation (NOV) model [Baldwin 2000]

Challenges - 1

estimation of volatility: Volatility is used as a measure of price/value fluctuation, which may also be the key uncertainty input. Volatility is challenging because, depending on the context of the problem, it may refer to an historic or implied value [Hull 2006]. For example, in the context of a stock option, volatility may refer to an *historic* value, as a measure of past fluctuations in the stock price. That is an observed value and can be calculated as the standard deviation of the log of price returns. Volatility can be calculated by using a current option price and solving for volatility as output; this is referred to as *implied* volatility. To cal-

Challenges - 2

price of an option as a function of an underlying replicating portfolio: Options may be valuable, but they incur an up-front cost to acquire. The ability to realize this value depends on how well uncertainty is managed and how well opportunities (that arise as more information becomes available) are evaluated and acted on. For financial stocks, this up-front cost to

Challenges - 3

exercise strategy: A financial option can typically be exercised only once, either at a given deadline if European or any time up until a predetermined time if American. When talking about real options in design projects, such as software architecting, a design strategy that provides some design flexibility may actually permit options being exercised multiple times. Modifiability is an example of such a strategy: it allows for multiple additions and, hence by definition, multiple equally fruitful opportunities for exercising the option. The experiment-

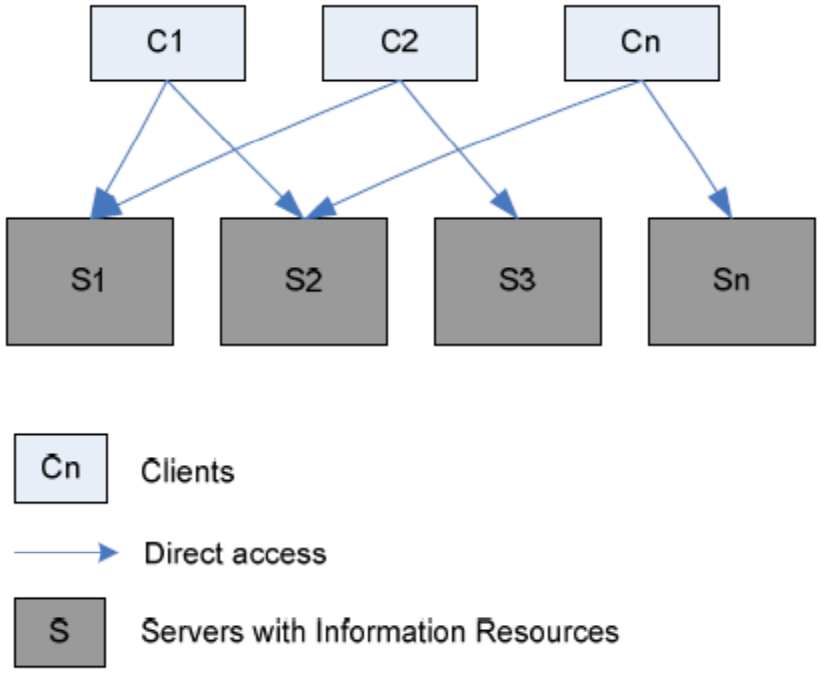
Real options analysis is an applicable valuation technique when the following conditions exist:

- **uncertainty**: There is uncertainty about what may happen in the future.
- **business goal**: The uncertainty is important for managing or achieving a business goal.
- **new information**: The business should be in a position to exploit new information when it becomes available. The assumption is that new information reduces uncertainty and opens up different opportunities.
- **action today**: Recall that an option is defined as the right, but not the obligation to make a decision in the future. To obtain this right (i.e., to hold the option), a stakeholder must take some deliberate value-driven action today.
  - **possibility of future design choices**: The action taken today opens up some future design choices that may not be as feasible today.
  - **possibility of future value**: The future choices create opportunities of value.

The model problem that we will use as a running example is a city information system (CIS) Web site, provided by BizCo. This is an application that operates in an inherently distributed environment. BizCo makes its profit based on the number of hits its Web site gets. Users retrieve information using a Web browser. *Information resources* host data about city events, places, and the like. BizCo's management believes that more information resources carried in the system will attract more users.

BizCo's current system uses the simple client-server architecture shown in Figure 2. In order to access an information resource in this system, its location must be hard-coded and the service must be executed in the server. While this design is simple and has acceptable latency, it makes the addition of new resources cumbersome. When the information provider receives a request from a client, it runs the appropriate service and returns the information. The system is expected to evolve over time through the addition of new information resources. That addition may also result in an increase in the number of users, which in turn may cause the availability of the system to become a critical quality concern.

Architecture for example



CIS as a real options problem

- **uncertainty:** The ability to find new resources willing to contribute information is a source of uncertainty. BizCo's marketing department predicts that more information resources will bring more users, but it cannot accurately predict the demand—another source of uncertainty. More users will mean that availability requirements will become more stringent. Currently, the system has an availability goal of no more than five hours of downtime per day.
- **business goal:** BizCo would like to increase its market share to 10% of the CIS market. Doing so requires increasing the number of new information resources, which is predicted to increase the number of users.
- **new information:** BizCo is looking into several new collaborations for providing a more diverse information pool. For the sake of simplicity, we assume that any new information resources become available monthly.
- **action today:** BizCo is seeking answers to the following questions: How and when should it evolve the current architecture in response to new user and information-source demands? The action to be taken today involves two considerations:
  - **possibility of future design choices:** BizCo would like to know how to stage architectural decisions and determine which ones bring more value relative to its business goals.
  - **possibility of future value:** If BizCo sticks with the existing client-server architecture, management believes that the company will not be able to respond to future demands, such as including different cities' information. Hence, it may lose market share.

Uncertainty in modifiability: The uncertainty for modifiability is the number of new information resources that will need to be added. If a new information resource needs to be added every month, with the current architecture, BizCo does not expect to be able to incorporate it into the system in a timely fashion. Hence, the product would lose market share.

Uncertainty in availability: The uncertainty for availability is how critical downtime will be for users. Currently, the system is down from 3 AM – 8 AM for regular maintenance. When new in- be acceptable downtime and may need to be reduced to a maximum of one hour per day, or even less.

Table 1: Tactics Identified to Address Quality Attribute Concerns

Quality Attribute	**Tactics**	**Patterns to achieve tactic**
Modifiability	Prevention of ripple effects: hide information, use an intermediary	Client-Server, Proxy, Broker, Client-Dispatcher-Server
Performance	Resource demand: reduce computational overhead Resource management: maintain multiple copies	Proxy, Shared Memory
Availability	Fault detection: ping/echo Fault recovery: active redundancy	Client-Server, Proxy, Broker, Client-Dispatcher-Server,

Table 2: Broker, Proxy, and Client-Dispatcher-Server Patterns

	Benefits	Liabilities
BROKER	Clients and servers do not need to know each others' locations, since all requests are handled through the broker.	Slower runtime performance
	Changeability and extensibility of components	Low fault tolerance
	Reusability of existing services	
PROXY	Decoupling of clients from location of servers (with better runtime in exchange for some loss of flexibility)	Efficiency: slower due to indirection
	Efficiency at lower cost by using "load on demand" strategy	Loading on demand does not work when originals are highly dynamic
CLIENT-DISPATCHER-SERVER	Exchangeability of servers	Efficiency: slower due to indirection and explicit connection establishment
	Allows for deferring decisions and preparing for later conversion to a distributed system	Dispatcher is a central component, and changes to the interface of this component creates modifiability vulnerabilities.
	Fault tolerance: new servers can be activated without impact	
CLIENT-SERVER	Enables encapsulation of the handling of shared resources on the behalf of multiple clients with a simple structure	Clients need to explicitly know the location of the servers to access services, which hinders scalability and modifiability when new servers need to be added.
	Enables processes that reside on the servers to be modified independent of the clients	Slower runtime performance due to communication between clients and servers

Elicited values

Table 3: Quality Attribute Responses

	Modifiability	Performance	Availability
	person-day	seconds	hr downtime
Client-Server	7	0.5	5
Client-Dispatcher-Server	2	3	1
Proxy	2	2	1
Broker	1	5	3

A financial (stock) option is the right, but not the obligation to buy (or sell) some amount of stock at some point (or interval of time) in the future. Determining the value of a stock option is based in part on the value of the stock [Hull 2006]. In real options analysis, the value of the project's expected cash flow is similar to the *stock price* in a financial option. We use a different approach. We associate *utility* with different aspects of the system such as levels of functionality and quality. Total system utility is our stock price equivalent. The cost of making the change enabled by the option (such as adding a new component) is equivalent to the *exercise price*, and the time until the opportunity disappears is the *time to expiration*. *Volatility* represents project value fluctuation, and finally the *risk-free interest rate* is assumed to be a known market value [Hull 2006, Amram 1999, Favaro 1998] (or possibly a known organization-specific value).

We map these components to our analysis as follows. The *decision* BizCo needs to make is which design pattern to embed in the architecture and when. *This decision is evaluated based on the value of the options created by the selected pattern.* The cost of switching to a new architecture pattern is similar to the premium to be paid in buying a stock option—that is, the cost of the option, which we refer to as the *switching cost.* The *exercise price* is the maintenance cost that needs to be spent in each alternative for adding new information resources. The time interval *t* in our example is three months.[2] The risk-free interest rate *r* is 6% per year or 0.5% per month.

In the CIS example, we use total, system-wide utility value instead of stock price. The system value is determined by summing several dimensions of utility obtained from the quality attributes the system achieves.

$$S = V_S + v_{IP} + v_A \qquad (1)$$

where

- S is total system value.
- v_{IP} is the utility value from modifying the system with information providers.
- v_A is the utility value from having more availability.
- V_s denotes the rest of the current system's value prior to making any changes.
- V_{IP} is the expected outcome associated with modifiability.
- V_A is the expected outcome associated with availability.

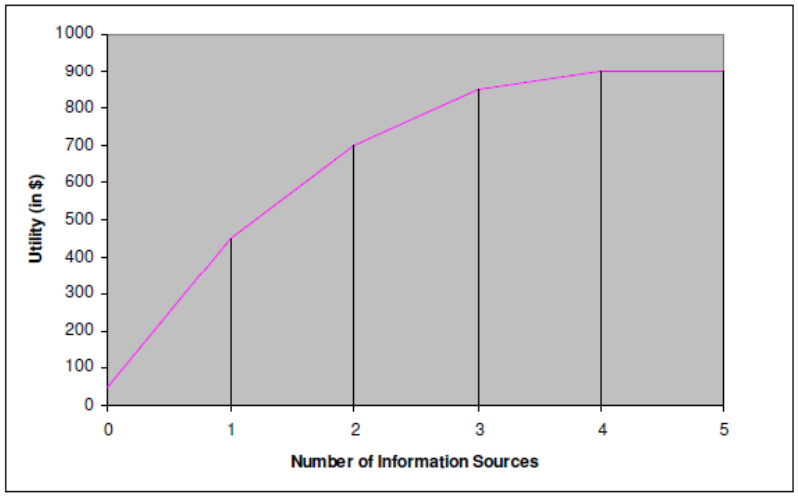


Figure 3: Utility Versus New Information Resources in Time

Figure 4 shows the predicted favorable and unfavorable outcomes at each time period using a binomial tree. For example, in Figure 4, at cell G, after three months, $850 is the utility associated with three information providers. This value is assumed to be elicited from the stakeholders and charted as a utility graph, as represented in Figure 3. The associated graph in each figure charts utility as a function of each uncertain variable. (A more accurate depiction of the utility curve, would be utility surface, where utility is a function of time and the uncertain variable. We are actually plotting a line on that surface.) The explanations at the leaf nodes describe the state observed at the end of the given three-month period with the paths reaching to that particular node. For simplicity, we graph utility in terms of dollars.

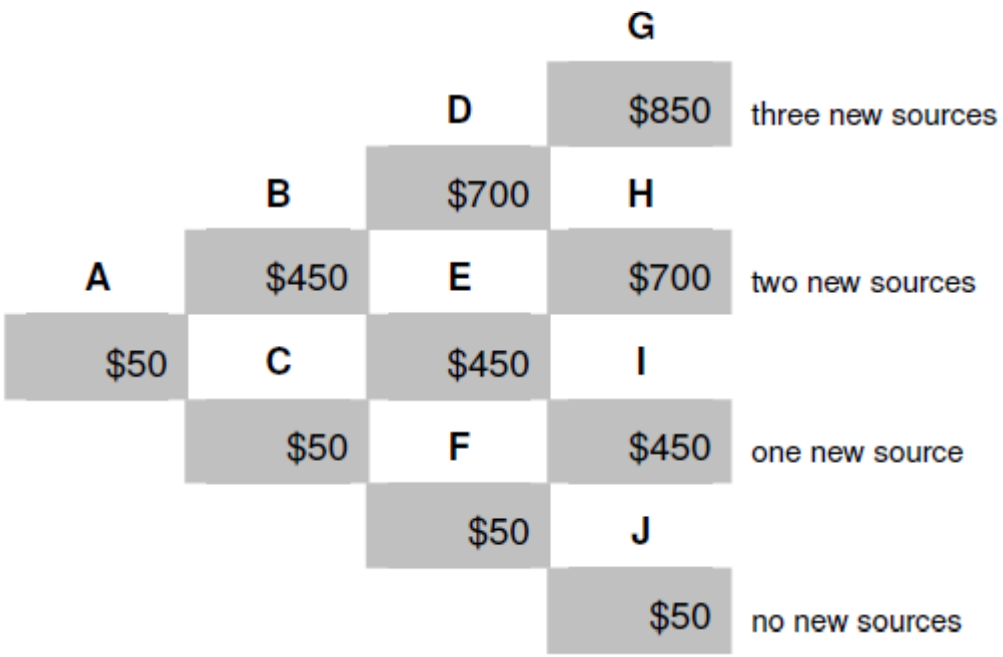


Figure 4: Modifiability – Uncertainty in the Number of New Information Resources

formula

options pricing model. For a two-step binomial tree, consisting of the starting point and the states one level further in time, the option price formula is

$$f = (pf_u + (1-p)f_d) / (1+r) \qquad (2)$$

where

- f is the option price.
- Subscript *u* indicates that the value of the system (in our case, the stock equivalent) goes up.
- Subscript *d* indicates that the value of the system goes down.
- *uu* in a multi-period timeline would refer to two subsequent favorable outcomes.
- *ud* in a multi-period timeline would refer to one favorable and one unfavorable outcome and so on.
- r is the risk-free rate of return.
- p is a coefficient as defined in Equation 3.

$$p = (1+r - d) / u-d \qquad (3)$$

Here, u is the factor by which the stock price increases, and d is the factor by which the stock price decreases. In our case, the u and the d values are calculated based on the predicted system values we elicit from stakeholders. There are more general formulas for n-level binomial trees.

3.6.1 Option Value: Modifiability

First, we consider modifiability to be the only quality attribute of concern in making the pattern choices. Figure 7 shows our calculations. The underlined values are the favorable and unfavorable outcome coefficients, *u* and *d*, that we calculate from the given system values. The p values are calculated based on Equation (3). The end nodes of each tree are calculated using Equations (5) or (6). The upper cells represent the system value, generated using formula (1). The shaded cells are the value of the option, based on (2).

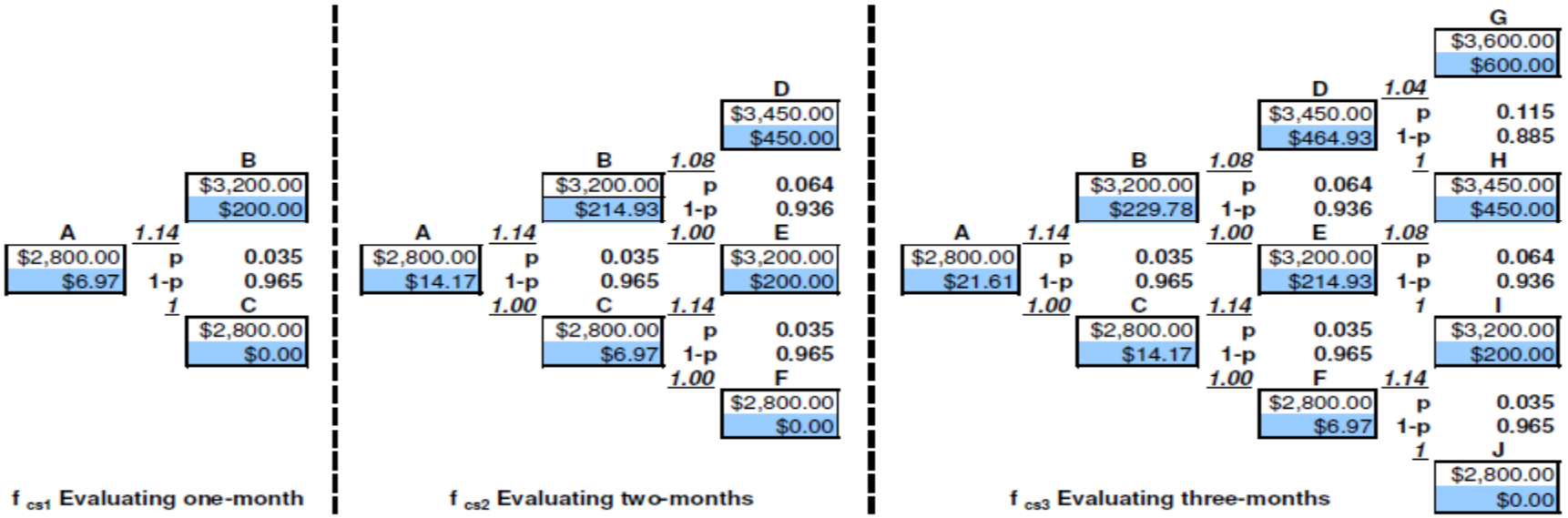


Figure 7: Evaluation of the Options in Client-Server Pattern with New Information Resources

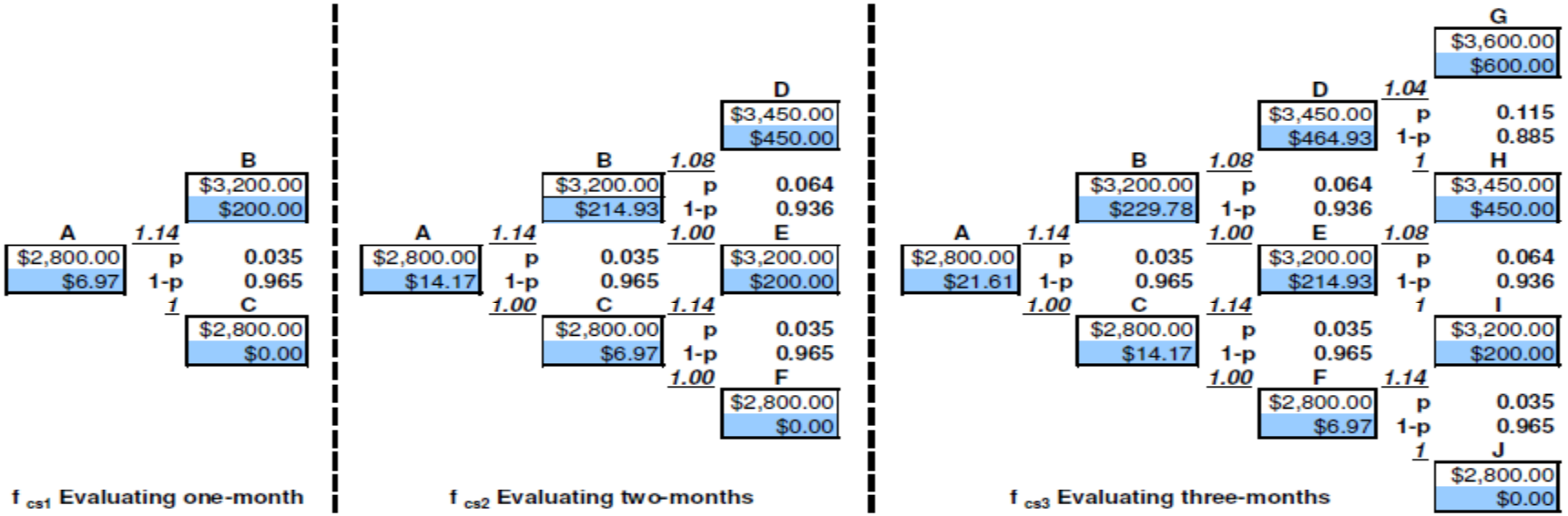


Figure 7: Evaluation of the Options in Client-Server Pattern with New Information Resources

To see in detail how these values are computed, consider cell D in the evaluation of the two-month case in Figure 7. The upper value in this cell is calculated as

$$S_{uu} = V_S + V_{IPcs} \quad (2) = 2750 + 700 = 3450$$

The lower value is calculated as

$$f_{uu} = \max(0, S_{uu} - C_{csm})$$

The option value of the Client-Server system, f_{cs}, is then calculated as

$$f_{cs} = f_{cs1} + f_{cs2} + f_{cs3}$$

$$f_{cs} = 6.97 + 14.17 + 21.61 = 42.75 \quad (9)$$

The same logic is used in calculating the option values of the Client-Dispatcher-Server, Proxy, and Broker designs. We use the matching maintenance costs for each design that we presented in Table 4. Figure 8 shows the corresponding valuation for Client-Dispatcher-Server or Proxy.

3.6.2 Option Value: Modifiability and Availability

Next, we take into consideration not only the utility value the patterns have as a result of modifiability but also the utility they do or do not exploit due to availability. The reasoning we follow is the same as with the calculations we demonstrated in Section 3.6.1. However, in considering the patterns, we also take into consideration the utility and maintenance costs due to availability. The system values of the different alternatives would be generated using $S = V_s + v_{IP} + v_A$ for the Client-Dispatcher-Server and Proxy designs. The availability utility to be gained from the current Client-Server and Broker designs are below the desired quality-attribute-response level expected: hence, they do not provide added benefit. This is represented by not considering the availability utility value as $S = V_S + v_{IP}$. An architectural pattern with vulnerability in respect to a certain quality attribute may still bring some utility value or may actually cause the overall design to lose value. Here, we make a simplification and assume v_A as zero for these cases where the design choices have vulnerabilities with respect to availability.

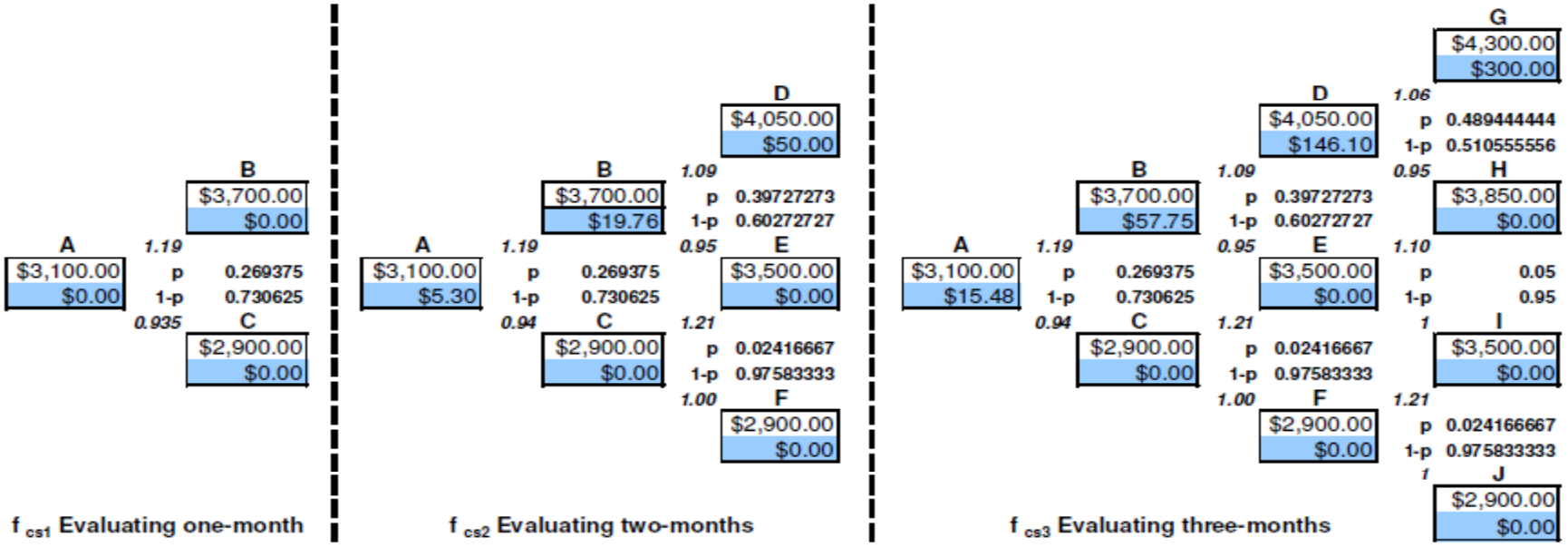


Figure 10: Evaluation of the Options in Client-Server Pattern with New Information Resources and Availability

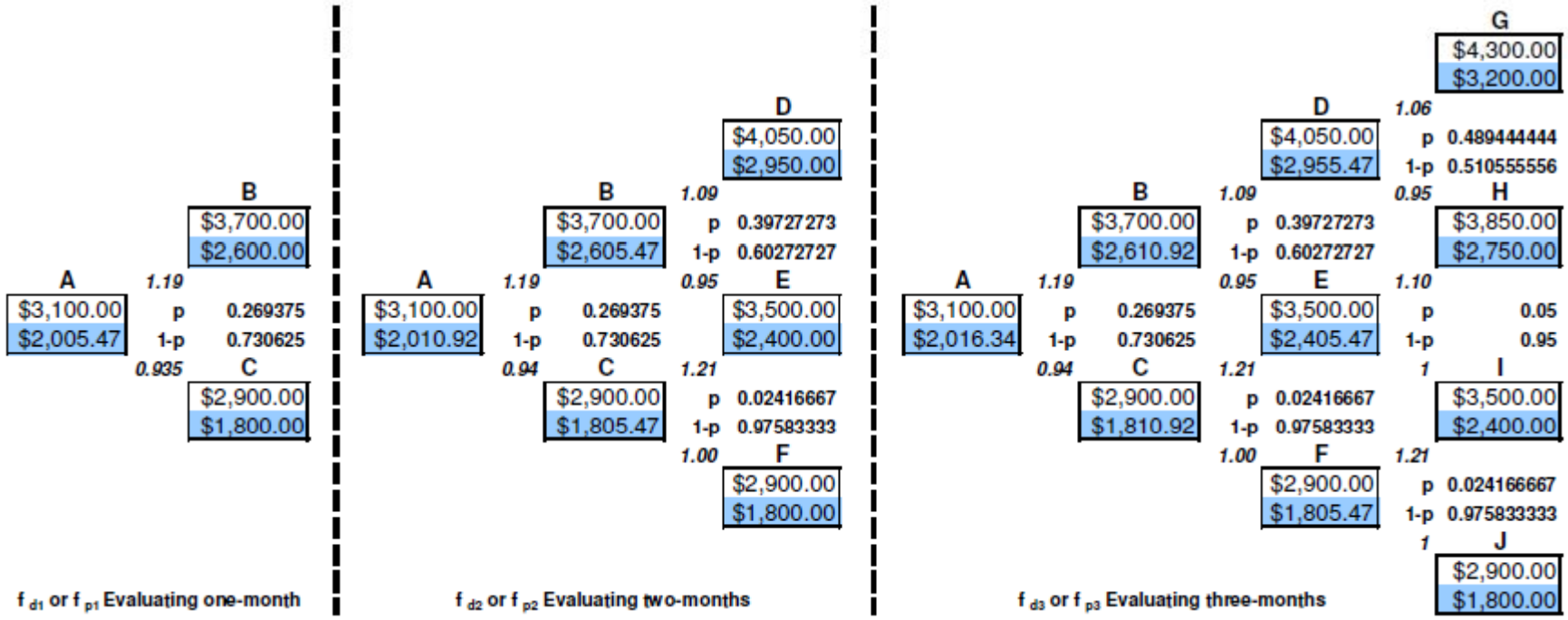


Figure 11: Evaluation of the Options in Client-Dispatcher-Server and Proxy Patterns with New Information Resources and Availability

The total option value of the Client-Dispatcher-Server or Proxy designs increases as a result of their strength in dealing with availability. This value evaluates as

$$f_p = f_d = f_{d1} + f_{d2} + f_{d3}$$

$$f_p = f_d = 2005.47 + 2010.92 + 2016.34 = 6032.73$$

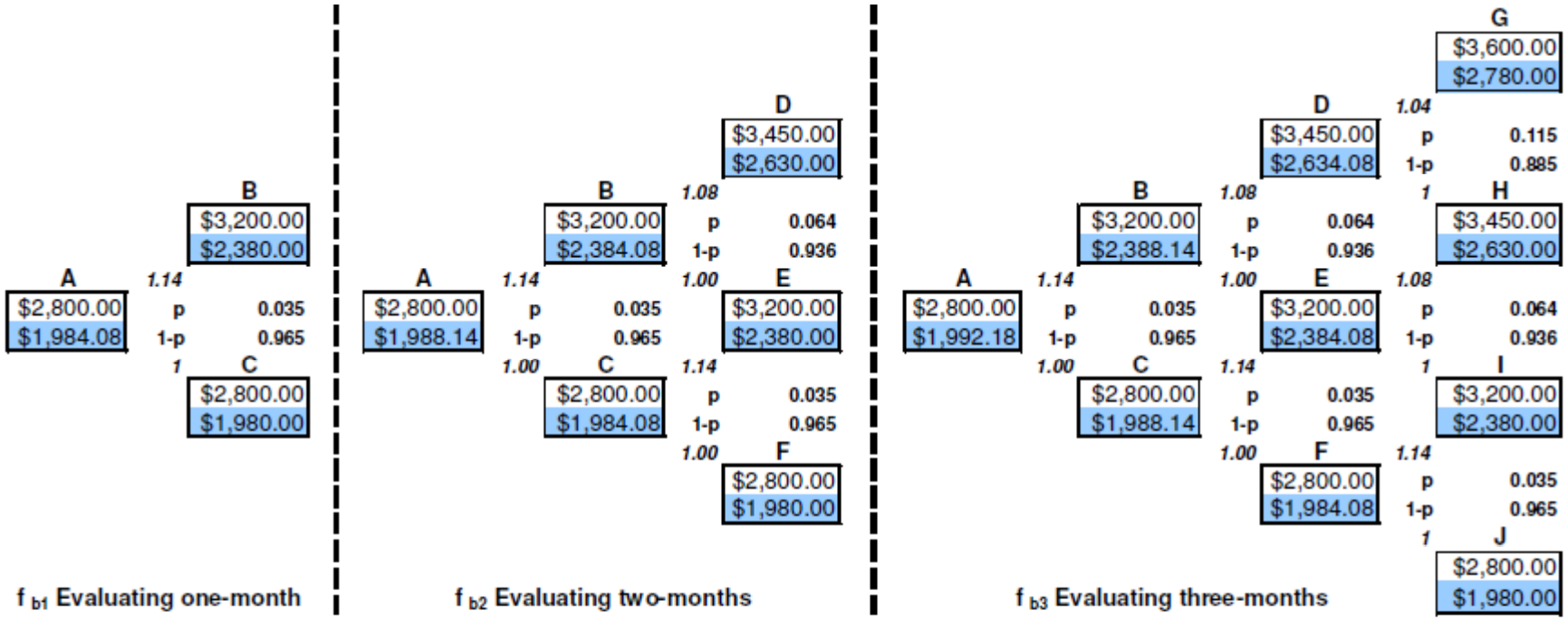


Figure 12: Evaluation of the Options in Broker Pattern with New Information Resources and Availability

The total option value in this case is

$$f_b = f_{b1} + f_{b2} + f_{b3}$$

$$f_b = 1984.08 + 1988.14 + 1992.18 = 5964.40$$

In this scenario, the company is better off *not* changing to a Broker design because the Client-Dispatcher-Server or Proxy choices can better accommodate the importance of availability in the market. The values of those choices are higher than that of the broker, despite its advantage in modifiability. This conclusion also holds when the switching costs (presented in Table 4) are taken into consideration. Spending $4500 to switch to a Broker design creates value (5964.40–4500 = 1464.40), but switching to Client-Dispatcher-Server or Proxy is *more* advantageous (6032.73 – 4100 = 1932.73).

So here is what you are going to do

- Examine your architecture for points of uncertainty
- Develop an operational profile
- Return to a previous decision about a pattern to use
- Consider the alternative patterns
- Describe the considerations about each pattern as they relate to operational profile and uncertainty