universität
wien

DIPLOMARBEIT

Titel der Diplomarbeit

Web Services Implementation of Option Valuation

Band 1 von 3 Bänden

Verfasser

Alexander Hopfgartner

angestrebter akademischer Grad

Magister der Sozial- und Wirtschaftswissenschaften (Mag.rer.soc.oec.)

Wien, 2008

Studienkennzahl lt. Studienblatt:	A 175
Studienrichtung lt. Studienblatt:	Wirtschaftsinformatik
Betreuer:	Univ.-Prof. Dipl.-Ing. Dr. Engelbert Dockner

ZUSAMMENFASSUNG

Die Diplomarbeit beschreibt die am häufigsten verwendeten Modelle und Methoden für die Bewertung von europäischen und amerikanischen Standardoptionen, sowie einige der bekanntesten exotischen Optionen. Diese Modelle bilden die Basis zur Entwicklung und Realisierung von Webservices welche die Bewertung durchführen.

Im ersten Abschnitt wird der Begriff *Option* definiert, und das Modell für Vermögenswerte, welches die Basis bildet, umrissen. Der nächste Teil ist der fundamentalen partiellen Differenzialgleichung und deren Ableitung gewidmet, die die Entwicklung aller Derivate beschreibt, deren Profit von einem einzelnen zugrunde liegenden Vermögenswert abhängt. Danach werden die binomialen und trinomialen Modelle von Vermögensgegenständen beschrieben und die effiziente Bewertung von europäischen beziehungsweise amerikanischen Optionen anhand von binomialen und trinomialen Bäumen gezeigt. Als nächstes wird die Konstruktion von trinomialen Bäumen im Einklang mit den Marktpreisen europäischer Standardoptionen und deren Nutzung zur Bewertung von pfadabhängigen exotischen Optionen, wie Barrier- oder Lookback-Optionen, dargestellt.

Der zweite Abschnitt befaßt sich mit Webservices und einer Definition des Begriffs *Webservice*. Es folgt eine ausführliche Beschreibung der Architektur für Webservices aus Anwendersicht sowie der notwendigen Interaktionen und Werkzeuge. Danach wird ein grundlegender Ansatz für die Entwicklung von Webservices aus Entwicklersicht von Dienstanbietern und Dienstanwendern beschrieben. Dieser erklärt den Lebenszyklus, die Operatoren, sowie die Interaktionen und Anwendungen, die notwendig sind, um Webservices im Allgemeinen zu realisieren.

Der dritte Abschnitt zeigt die Anwendung dieser allgemeinen Konzepte und Werkzeuge unter Verwendung der Java™-Technologie. Der Prozess der Entwicklung und der Implementierung der Webservices für die Preiskalkulation wird erklärt und mittels kurzer Code-Beispiele verdeutlicht. Abschließend werden Beispiele dieser Webservices zur Optionsbewertung mit ihren Ergebnisbäumen bzw. –gittern angeführt.

ABSTRACT

This diploma thesis describes the most commonly used models and methods for pricing standard European and American options, as well as some of the best known exotic options. These models are the base to develop and implement Web Services that manage the valuations.

In the first part the term *option* is defined and the model for asset prices forming the basis is outlined. The fundamental partial differential equation is derived that describes the evolution of all derivatives whose payoff depends on a single underlying asset. Then the binomial and the trinomial models of asset prices are described and it is shown, how European and American derivatives can be priced efficiently in binomial and trinomial trees, respectively. Also, it is described how trinomial trees can be constructed to be consistent with the market prices of standard European options and shown how they can be used to price exotic path-dependent options such as barrier or look-back options.

The second part introduces Web Services and the term *Web Service* is defined. A detailed description of the architecture for Web Services from the operator perspective, as well as interactions and artifacts is given. Then a basic approach for developing Web Services from the point of view of the developer of service providers and service requestors is described. It explains the development lifecycle, operators, interactions and application development patterns necessary to implement Web Services in general.

In the third part the development approach relates these common concepts and tools to their application using Java™ technology. The process of creating and implementing the pricing Web Services is explained and short code samples are given where necessary. Finally, valuation examples of the pricing Web Service implementation are shown with their output trees or lattices, respectively.

NOTATION

Symbol	**Description**
t	time
K	strike or exercise price of option
T	maturity date of option usually current date will be 0 and so T will also be time to maturity
S	asset price
μ	drift of S
σ	volatility of S usually instantaneous standard deviation of returns
r	instantaneous continuously compounded interest rate
δ	continuous dividend yield on an asset
τ	time discrete cash dividend paid
$C()$	European call price
$P()$	European put price
$E[]$	expectation operator
dS	infinitesimal increment in asset S
dt	infinitesimal increment of time
dz	infinitesimal increment in a standard Wiener process during dt
x	natural logarithm of S *(ln(S))*
v	risk neutral drift of x
Δt	small increment of time
Δx	small increment in x
u	size of proportional upward move of stochastic variable, or subscript indicating upward move of a stochastic variable
d	size of proportional downward move of stochastic variable, or subscript indicating upward move of a stochastic variable

m	subscript indicating a central move of a stochastic variable
p	probability of transition in a tree subscripted by u, m and d to indicate the direction of the transition
N	number of time steps in tree
N_j	number of nodes above and below current level of asset price
$N(x)$	Standard cumulative normal distribution function evaluated at x
i	time step index
j, k	usually a state variable level index
H	barrier level
X_{rebate}	cash rebate associated with barrier option
Q	state price
$F_{i,j,k}$	path dependent variable value k at node i, j
$n_{i,j}$	number of path dependent variable values at node i, j
$exp()$	e^x

TABLE OF CONTENTS

PART THREE: JAVA WEB SERVICES IMPLEMENTATION

FIGURES

TABLES

1 Options Introduction

1.1 Genesis and History

The year 1973 is often called the birth of options. With the establishment of the Chicago Board of Options Exchange (CBOE) and the introduction of options traded to stock exchange, a central institution for the trade with standardized options was present for the first time. Additionally the option clearing corporation which was founded in the same year served as intermediary between the contracting parties in the option business.

However, options are much older and have a long and well documented history. Already the ancient Greeks knew about options and how to make their money with options. Thus Malkiel and Quandt [1] report of a philosopher named Thales, who earned a fortune with option contracts on the use of olive presses. In addition, in Amsterdam around 1600, both call and put options on tulip bulb were traded [2]. The first mention of options in the United States dates back to the year 1792 at the same time as the New York Stock Exchange was established [3]. In Austria futures and options are traded on the 'Österreichische Termin-und Optionenbörse' (OTOB).

1.2 Options Definition

Options are one of the main types of derivatives which are financial instruments whose values depend on the value of the underlying.

'In finance, the underlying of a derivative is an asset, basket of assets, index, or even another derivative such that the cash flows of the (former) derivative depend on the value of this underlying. There must be an independent way to observe this value to avoid conflicts of interest' [4].

An asset is a probable future economic benefit obtained or controlled by a person or company as a result of a past transaction or event [5].

In order to be able to make a valuation from standardized options and exotic options, it is essential to understand their nature.

The definition of an option is:

'**Options are financial instruments that convey the right, but not the obligation, to engage in a future transaction on some underlying security, or in a futures contract**' [6].

It is upon the option holder's choice to exercise the option, whether the party who sold the option must fulfill the terms of the contract.

Call options provide the right to buy a specified quantity of an asset at a set strike price at some date on or before expiration.

Put options provide the right to sell a specified quantity of an asset at a set strike price at some date on or before expiration.

It can be seen from this definition that the price of an option is thus affected by a number of factors:

- The present price of the underlying asset.
- The strike price.
- The time up to the maturity (expiration date).
- The volatility or standard deviation of the underlying asset.
- The interest rate.

1.3 Classification

As a classification for the bulk of different options generally the style of an option is used, which is usually defined by the dates on which the option may be exercised.

Therefore the following style categories exist:

- European options - may be exercised only on maturity (expiration date).
- American options - may be exercised on any trading day on or before expiration date.
- Bermudan options - may be exercised only on fixed dates on or before expiration date.
- Barrier options – require that the underlying asset must reach some trigger level before the exercise can occur.

Additionally the payoff of the option is used for categorization.

For example European and American options - as well as others where the payoff is calculated similarly - are referred to as ‘vanilla options’. Options where the payoff is calculated differently are categorized as ‘exotic options’. Exotic options can pose challenging problems in valuation and hedging.

2 Black-Scholes World

One of the most important sizes in evaluation models for options is the underlying. Independently of all other parameters the price of the asset to which the option refers, finally determines the value of the option mainly. However, a substantial basic assumption of a multiplicity of option evaluation models is that the exact value of the underlying does not let itself predict - not even by historical course time series or existing evaluation models [7]. However conclusions on the average value and the variance of course changes can be made by evaluation models or by means of these historical course time series. Thus a probability distribution of future asset values can be calculated.

2.1 Black-Scholes Model

Almost all option evaluation models are based on a simple model for price movements of the underlying, i.e. the random walk[1].

The model looks as follows:

At time *t* the asset has the value *S*. Within a small time interval *dt* it changes the value at *S* to *S* + *dS* (see Figure 2.1).

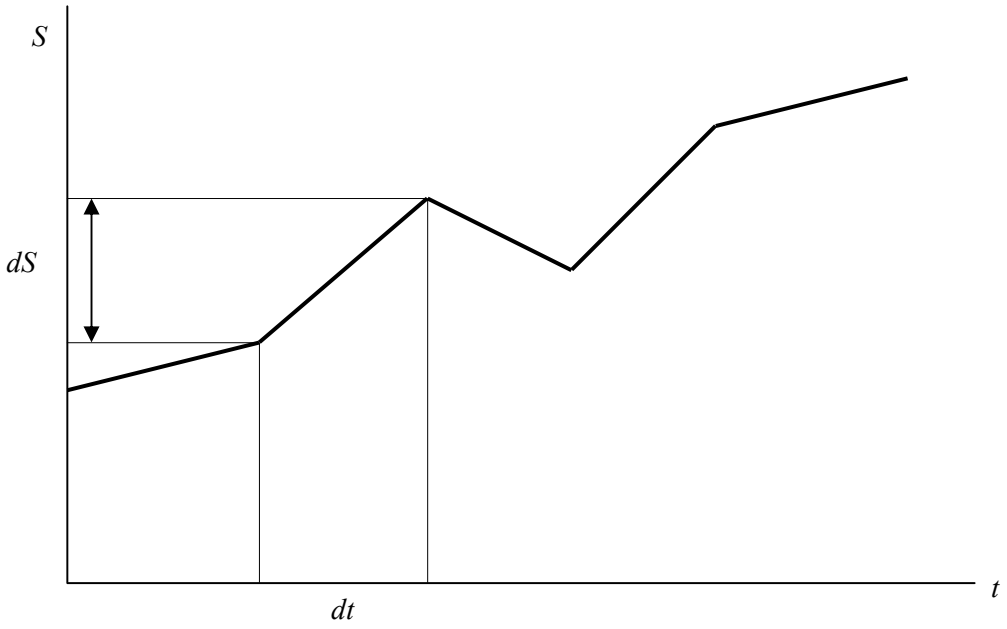

Figure 2.1: Development of an Underlying During Time

That results in a relative profit of *dS/S* for the appropriate period. If an average value from all relative yields during a longer time period is calculated, the average growth rate[2] μ of the asset is obtained, also known as drift or trend.

[1] The random walk hypothesis is a financial theory stating that stock market prices evolve according to a random walk and thus the prices of the stock market cannot be predicted (Wikipedia, 2008).
[2] In simple models μ is assumed to be constant, or a function of *S* and *t* in more complex models.

Moreover random changes in the price of the asset must be considered due to external effects - e.g. unexpected messages. This happens via a random number *dz* which is weighted by means of volatility σ - i.e. the standard deviation [7].

A Wiener[3] process *dz* has the following key properties:

- *dz* is normally distributed with mean zero and variance *dt* or standard deviation $\sqrt{dt}$
- the values of *dz* over two different, non-overlapping increments of time are independent

If these two components are joined the stochastic differential equation is formed.

$$\frac{dS}{S} = \mu\, dt + \sigma\, dz \qquad (2.1)$$

It mathematically describes the model for the asset price. The right side of the equation contains a deterministic part $\mu\, dt$ and a part determined by the randomness of $\sigma\, dz$.

However, (2.1) does not describe a single deterministic path for example of a share. In fact many different evolvements, that are time series, can be generated with formula (2.1), where each item represents a possible future course path. From these different evolvements interesting and important information is gained concerning the probability of the distribution of the share quotation at a specific time. As a result skewed and bell-shaped probability density functions are obtained, as in Figure 2.2.

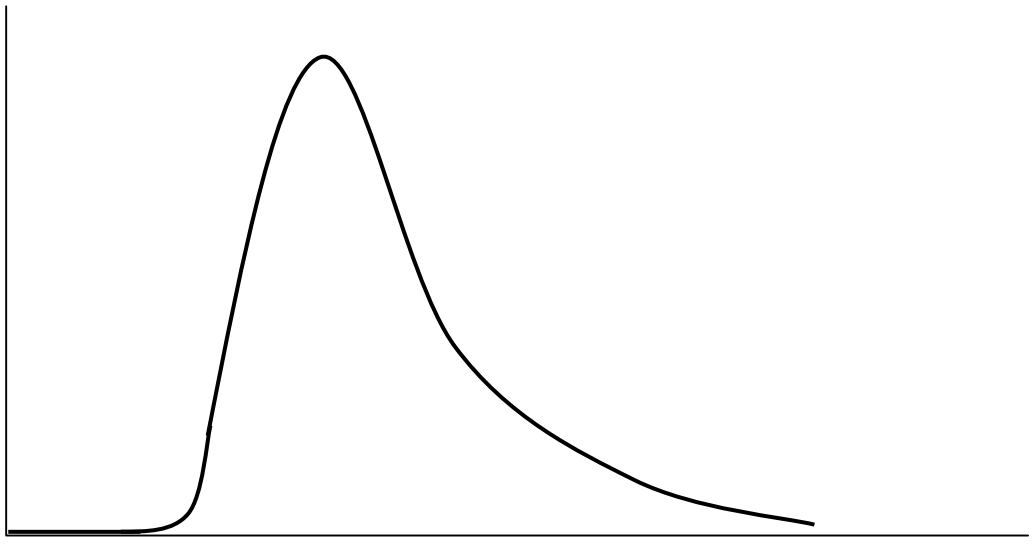

Figure 2.2: Probability Density Function of the Random Walk

[3] The random variable, or equivalently the change *dz*, is called a Wiener or Brownian motion process.

If the share price obeys the model described by (2.1), the probability density function of the random walk is log-normally distributed.

The discrete model of the random walk works very well for quite large time intervals. However, if used in the material life, the discrete, mathematical model is changed in to a continuous model where the size of the time interval converges towards zero - $dt \to 0$.

2.2 Derivation of the Black-Scholes Partial Differential Equation

The transformed mathematical model by means of Itô's lemma[4] appears as follows:

$$df = \left(\mu S \frac{\partial f}{\partial S} + \tfrac{1}{2} \sigma^2 S^2 \frac{\partial^2 f}{\partial S^2} + \frac{\partial f}{\partial t} \right) dt + \sigma S \frac{\partial f}{\partial S} dz \qquad (2.2)$$

This formula represents a substantial part during the derivation of the Black-Scholes option valuation formula [8].

Starting point for the development of the Black-Scholes formula is the assumption that there is an option, whose value C depends only on two sizes - i.e. the course of the underlying S and the time t. At this time it is not yet necessary to specify whether the option is a call or a put [7]. In accordance with the formula (2.2) deduced above the random walk, which the option C follows, can be defined as

$$dC = \left(\mu S \frac{\partial C}{\partial S} + \tfrac{1}{2} \sigma^2 S^2 \frac{\partial^2 C}{\partial S^2} + \frac{\partial C}{\partial t} \right) dt + \sigma S \frac{\partial C}{\partial S} dz \qquad (2.3)$$

In a further step a portfolio is designed, composed of an option C and a not yet specified quantity - Δ - of the underlying. The value of this portfolio is

$$\Pi = C - \Delta S \qquad (2.4)$$

And the change within one time period is

$$d\Pi = dC - \Delta\, dS \qquad (2.5)$$

[4] In mathematics, Itô's lemma is used in Itô stochastic analysis to find the differential of a function of a particular type of stochastic process.

The random walk of this portfolio obeys the following equation:

$$d\Pi = \left(\mu S\frac{\partial C}{\partial S} + \tfrac{1}{2}\sigma^2 S^2\frac{\partial^2 C}{\partial S^2} + \frac{\partial C}{\partial t} - \mu\Delta S\right)dt + \sigma S\left(\frac{\partial C}{\partial S} - \Delta\right)dt \qquad (2.6)$$

If Δ - which is not yet specified - is replaced with,

$$\Delta = \frac{\partial C}{\partial S} \qquad (2.7)$$

in the above formula (2.6) the random component dz can be eliminated and the equation simplifies to

$$d\Pi = \left(\frac{\partial C}{\partial t} + \tfrac{1}{2}\sigma^2 S^2\frac{\partial^2 C}{\partial S^2}\right)dt \qquad (2.8)$$

Investing amount Π into a portfolio without risk would gain a profit of $r\Pi dt$ within one time period dt.

If the right side of equation (2.8) would be larger than the gained profit of the portfolio without risk, an investor or arbitrageur could make a guaranteed profit without risk by borrowing an amount Π and invest in the portfolio. The profit would be larger than the costs of borrowing.

Also in the contrary case the arbitrageur could make a guaranteed profit without risk by selling the portfolio from formula (2.8) and investing amount Π in the bank. The market forces of supply and demand as well as arbitrageurs ensure that there is no profit without risk or only very briefly. Hence the profit from the portfolio without risk and the portfolio from formula (2.8) must be equal or approximately equal. Mathematically that means

$$r\Pi\, dt = \left(\frac{\partial C}{\partial t} + \tfrac{1}{2}\sigma^2 S^2\frac{\partial^2 C}{\partial S^2}\right)dt \qquad (2.9)$$

By replacing formula (2.4) and formula (2.7) and a division through dt we finally get the Black-Scholes partial differential equation:

$$\frac{\partial C}{\partial t} + \tfrac{1}{2}\sigma^2 S^2\frac{\partial^2 C}{\partial S^2} + rS\frac{\partial C}{\partial S} - rC = 0 \qquad (2.10)$$

All derivatives which depend only on the price of the underlying S and the time t, must comply with this very general formula. Therefore this equation is suitable also for options which look at first sight complicated, for example exotic options.

The following assumptions[5] are the basis to derive the equation [7]:

- The price of the asset is log-normally distributed.
- The risk-free interest rate r and the volatility σ are well-known functions depending on the time over the lifecycle of the option
- there are no transaction costs for hedging the portfolio
- the underlying asset does not pay a dividend during the lifecycle of the option
- there are no arbitrage opportunities
- the trade of the underlying asset takes place continuously
- short selling is permitted and the asset is arbitrarily divisible

2.3 Black-Scholes Formula

While partial differential Black-Scholes equation was derived in chapter 2.2, this chapter shows how to solve the equation. In order to get a clear solution, the boundaries and final conditions must be defined in a first step.

For a European call with a current value of $C(S, t)$, a strike price K and an expiration date T the final condition at time $t = T$ is

$$C(S, T) = \max(S - K, 0) \tag{2.11}$$

This final condition corresponds to the pay-off profile at the expiry date and is well-known with accuracy.

For the boundary conditions the two extreme cases are examined, if the price of the underlying becomes zero, $S = 0$, and that value grows infinitely, $S \to \infty$. If the price of the underlying becomes zero, $S = 0$, the formula (2.11) shows that also dS is always zero and therefore the value of the underlying itself can never change. Therefore if $S = 0$ the option is worthless, even in the long run. Thus, the first lower boundary condition is

$$C(0, t) = 0 \tag{2.12}$$

If the value of the asset rises immensely it becomes more likely that the option is exercised and the height of the exercise price becomes less important. Therefore, if $S \to \infty$ the value of the option converges to the value of the underlying. Thus, the second upper boundary condition is

[5] Some of these assumptions can be dismissed by modifications in the model.

$$C(S,\ t) \approx S \quad \text{if} \quad S \to \infty \tag{2.13}$$

Figure 2.3 illustrates the conditions (2.11), (2.12) and (2.13) resulting in the shaded area of possible option values.

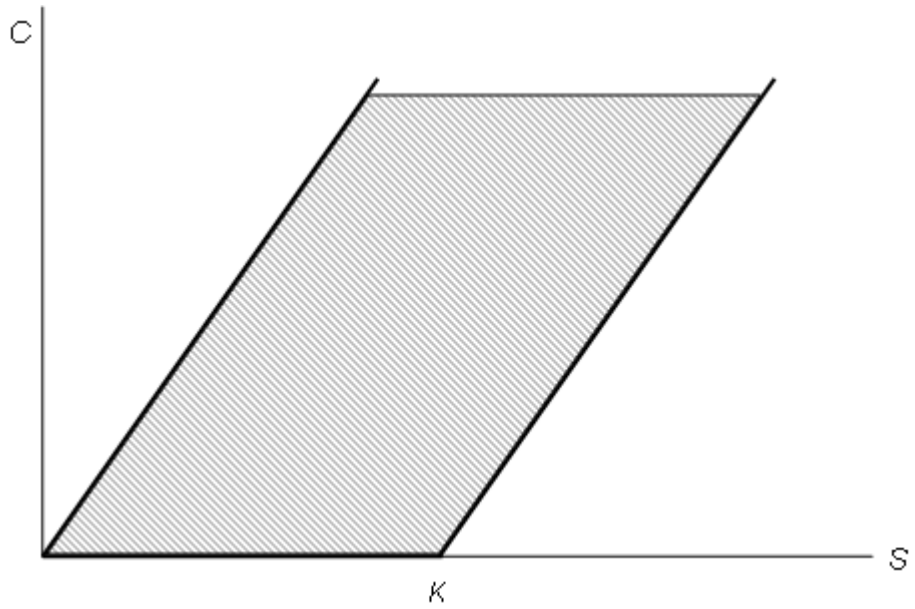

Figure 2.3: Boundary Conditions for a European call option

With these conditions - (2.11), (2.12) and (2.13) - the partial differential equation can be solved and the well-known Black-Scholes formula[6] for a European call option is obtained [8]:

$$C(S,\ t) = S\ N(d_1) - K\ e^{-r(T-t)}\ N(d_2) \tag{2.14}$$

where

$$N(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{1}{2}y^2}\, dy \tag{2.15}$$

$$d_1 = \frac{\log(S/K) + \left(r + \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}} \tag{2.16}$$

$$d_2 = \frac{\log(S/K) + \left(r - \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}} = d_1 - \sigma\sqrt{T-t} \tag{2.17}$$

[6] See [8] for the exact derivation.

For a European put option with the value *P (S, t)* the final condition is again identical to the pay-off profile at expiration date:

$$P(S,\ T)\ =\ \max(K\ -\ S,\ 0) \tag{2.18}$$

For the boundary condition again the two extreme cases $S = 0$ and $S \to \infty$ are considered. As already mentioned above the value of S remains always zero in the case $S = 0$. Therefore the pay-off of the put option at time T, the exercise price K, is accurately determined. In order to compute the value of the put option at time t, the exercise price K must be discounted only. Thus, the first upper boundary condition results as the present cash value of the exercise price K

$$P(0,\ t)\ =\ K e^{-r(T\ -\ t)} \tag{2.19}$$

In the second case where $S \to \infty$, the exercise of the put option is very unlikely and thus the option gets worthless. The lower boundary condition is

$$P(S,\ t)\ \to\ 0\quad \text{wenn}\quad S\ \to\ \infty \tag{2.20}$$

Figure 2.4 illustrates the conditions (2.18), (2.19) and (2.20) resulting in the shaded area of possible option values.

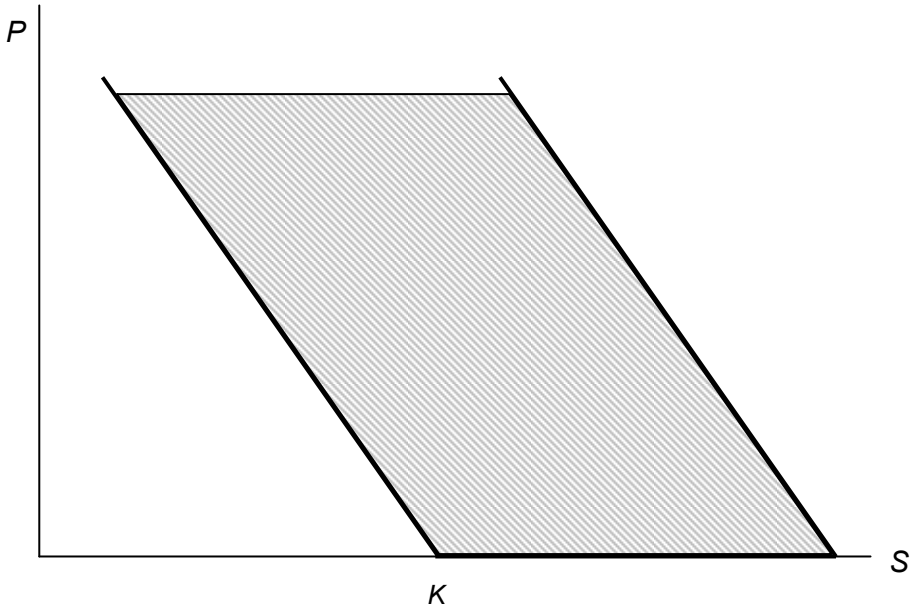

Figure 2.4: Boundary Conditions for a European put option

With these conditions - (2.18), (2.19) and (2.20) - the partial differential equation can be solved and obtains the well-known Black-Scholes[7] formula for a European put option [8].

$$P(S,\ t) = K\ e^{-r(T-t)}\ N(-d_2) - S\ N(-d_1) \tag{2.21}$$

where

$$N(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{1}{2}y^2}\, dy \tag{2.22}$$

$$d_1 = \frac{\log(S/K) + \left(r + \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}} \tag{2.23}$$

$$d_2 = \frac{\log(S/K) + \left(r - \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}} = d_1 - \sigma\sqrt{T-t} \tag{2.24}$$

2.4 Numerical Techniques

Due to their simplicity the Black-Scholes formulas are widely used by market participants. However, they are only applicable for standard European call and put options and not for pricing something more complicated such as American options.[8]

In order to value American-style options with their early exercise opportunities numerical techniques such as binomial and trinomial trees and finite difference methods must be used [7].

For exotic options especially path-dependent options, of which look-backs and barriers are probably the best known, again much more computationally intensive numerical techniques have to be applied. The pay-off of these options at maturity is some known function of the path that the asset takes before the maturity date, which can be hardly put in to a single valuation formula.

[7] See [8] for the exact derivation.
[8] To some extent the Black-Scholes formulae can be also adapted for pricing of other than standard options which reduces accuracy.

3 Binomial Model

One of the most common and best known numerical techniques for valuing options is the binomial model, especially for American-style options.

While European call or put options can be valued by using the Black-Scholes formula, for American call or put options the analytical approach is not applicable. Also for options on assets that pay dividends where the price of these options has no closed-form solution, numerical procedures must be used to solve the Black-Scholes partial differential equation.

Since early exercise of American options can be optimal depending on the level of the underlying asset, the binomial model allows handling this matter. Furthermore several extensions to price more complex options such as exotic options are possible.

3.1 Basics of the Binomial Model

In the binomial model the underlying asset price is expected to follow a binomial process. That means that the asset price can only change to one of two possible values at any time and so the asset price has a binomial distribution. During a time period Δt a considered asset with a current price of S can move - following a multiplicative binomial process - up to a new level uS or down to a new level dS. The average behavior and volatility of the asset are specified by the parameters u and d. Furthermore at the end of the time period Δt a considered call option on this asset matures, which also is shown in Figure 3.1. These are the first two branches of a binomial tree starting from its root - representing today - and evolving out in time by one time step [8] and [10].

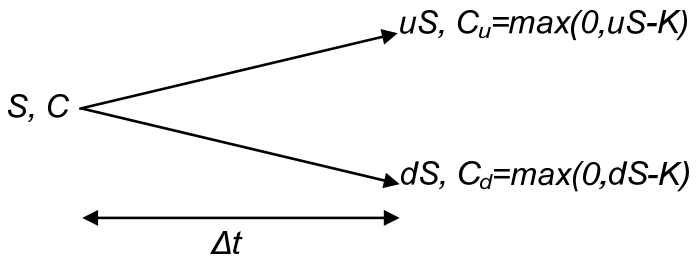

Figure 3.1: Simple Binomial Model of a Call Option and its Underlying Asset

Similar to the Black-Scholes model, a riskless portfolio can be set up consisting of Δ units of the underlying asset and a short position of one call option. The value of the portfolio needs to be the same regardless of whether the asset price goes up or down over the period Δt:

$$-C_u + \Delta uS = -C_d + \Delta dS \tag{3.1}$$

Rearranging the equation to:

$$\Delta = \frac{C_u - C_d}{(u-d)S} \tag{3.2}$$

Since this portfolio is riskless it must earn the riskless rate of interest r (continuously compounded).

$$(-C_u + \Delta uS) = e^{r\Delta t}(-C + \Delta S) \tag{3.3}$$

Substituting from equation (3.2) into equation (3.3) for Δ and rearranging for the call price at the start of the period C, gets

$$C = e^{-r\Delta t}\left(\frac{e^{r\Delta t} - d}{u-d}C_u + \frac{u - e^{r\Delta t}}{u-d}C_d\right) \tag{3.4}$$

Defining

$$p = \frac{e^{r\Delta t} - d}{u-d}$$

and substituting into equation (3.4) for p, the following simpler form is achieved:

$$C = e^{-r\Delta t}(pC_u + (1-p)C_d) \tag{3.5}$$

The above formula (3.5) shows the pricing of a call option with one period to maturity. For the valuation of a put option just the pay-off condition has to be changed, meaning the values of C_u and C_d for a put.

$$C_u = \max(0, K - uS)$$
$$C_d = \max(0, K - dS)$$

Note that the actual probabilities of the stock moving up or down are never used in deriving the option price, just as for the Black-Scholes model. This fact implies that the option price is independent of the expected return of the stock and therefore independent of the risk preferences of investors. This allows to interpret p and $(1-p)$ as risk-neutral probabilities and equation (3.4) as taking discounted expectations of future pay-offs under the risk-neutral probabilities. Calculating the risk-neutral probabilities directly from the asset price is quite simple - as assumed the return is the riskless rate.

Analogous to equation (3.5) the expected value of the underlying asset S at the end of the time period Δt can be written as

$$E(S_{\Delta t}) = uSp + dS(1-p)$$

Rearranging the equation to

$$E(S_{\Delta t}) = Sp(u-d) + dS$$

Substituting p

$$p = \frac{e^{r\Delta t} - d}{u - d}$$

in to the above formula, reduces to

$$E(S_{\Delta t}) = Se^{r\Delta t} \qquad (3.6)$$

Up to now the binomial model has comprised just one time step but it can be expanded to use more steps [11].

In Figure 3.2 the appropriate binomial tree for an option which matures in four periods of time is shown [9]. Each state in the tree is a node with two labels named i which indicates the number of time steps and j for the number of upward movements of the asset price, both measured from the beginning. So the asset price at node (i, j) is $S_{ij}=Su^i d^{i-j}$ and the option price is going to be C_{ij}. For the lowest node at every time step j is always zero. Generally it is assumed that the N^{th} time step corresponds to the maturity date of the option. When all $S_{N,j}$ values are computed the value of the option at the maturity date is simply the known pay-off, e.g. for a call option

$$C_{N,j} = \max(0, S_{N,j} - K) \qquad (3.7)$$

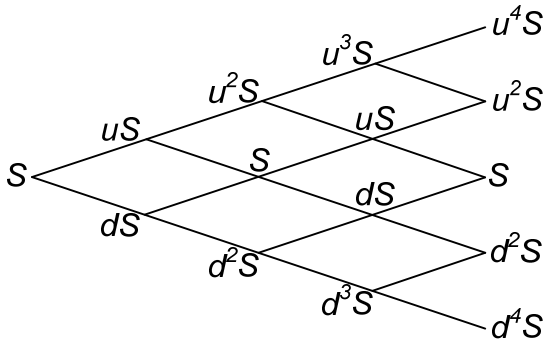

Figure 3.2: Four-step Binomial Tree for an Underlying Asset

As with the one period example the value of the option at any node in the tree before maturity is the discounted expected future value. The generalized formula is defined as

$$C_{i,j} = e^{-r\Delta t}(pC_{i+1,j+1} + (1-p)C_{i+1,j}) \tag{3.8}$$

To compute the value of the option at every node at time step *N-1* the equations (3.7) and (3.8) can be used. Reapplying equation (3.8) working backwards through the tree, the value of the option at every node in the tree can be computed.

The valuation of a European put option is quite straight forward, just the pay-off structure at maturity (N^{th} time step) changes to the known formula.

$$C_{N,j} = \max(0, K - S_{N,j}) \tag{3.9}$$

The backward calculation remains the same as for the European call option, using equation (3.8)

Due to the fact that American style options can be exercised not only at the maturity date but at every time step, the computation has to include the possibility of early exercise. Thus at every node there has to be a comparison between the value of the option if exercised and the value if not exercised. The option value at that node is then the greater of the two. For example an American put option

$$C_{i,j} = \max(e^{-r\Delta t}(pC_{i+1,j+1} + (1-p)C_{i+1,j}), K - S_{i,j}) \tag{3.10}$$

See chapter 12.6.1 for pricing examples.

3.2 Generalization of the Binomial Model

When constructing a binomial tree the behavior of the real asset price should be represented [8]. In order to reach this the parameters u, d and p should be chosen to match the mean and variance of the underlying asset during the time interval Δt. In the risk-neutral world the expected return from a stock is the risk-free rate r. Thus the expected value of the asset price at the end of the time interval Δt is

$$Se^{r\Delta t} = pSu + (1-p)Sd \tag{3.11}$$

or

$$e^{r\Delta t} = pu + (1-p)d \tag{3.12}$$

The variance of the proportional change in the asset price for interval Δt is $\sigma^2 \Delta t$. Therefore it follows

$$pu^2 + (1-p)d^2 - \left[pu + (1-p)d^2\right] = \sigma^2 \Delta t \tag{3.13}$$

This can be reduced to

$$e^{r\Delta t}(u+d) - ud - e^{2r\Delta t} = \sigma^2 \Delta t \tag{3.14}$$

by substituting from equation (3.12) for p (in a stochastic process the variance of a variable X defines as $E(X^2) - [E(X)]^2$).

For the three parameters p, u, and d two conditions – equation (3.12) & (3.13) have been set. The third condition used is

$$u = \frac{1}{d} \tag{3.15}$$

From these three conditions the values for the parameters are given by

$$p = \frac{a-d}{u-d} \tag{3.16}$$

$$u = e^{\sigma\sqrt{\Delta t}} \tag{3.17}$$

$$d = e^{-\sigma\sqrt{\Delta t}} \tag{3.18}$$

where

$$a = e^{r\Delta t} \tag{3.19}$$

to construct an appropriate binomial tree in a risk-neutral world.

The only problem with this formulation is that the approximation is only good over a small time interval. You cannot freely choose arbitrarily large time steps. To obtain a more general and flexible formulation the model is reformulated in terms of the natural logarithm of the asset price *(x = ln(S))*.

The natural logarithm of the asset price under GBM is normally distributed with a constant mean and variance. Applying Itô's lemma the continuous time risk-neutral process for x can be shown to be

$$\begin{aligned} dx &= \nu dt + \sigma dz \\ \nu &= r - \frac{1}{2}\sigma^2 \end{aligned} \tag{3.20}$$

Figure 3.3 shows the discrete time binomial model for x.

The variable x can either go up with a probability of p_u to a level of $x + \Delta x_u$ or down with a probability of $p_d = 1 - p_u$ to a level of $x + \Delta x_d$. This is known as the additive binomial process.

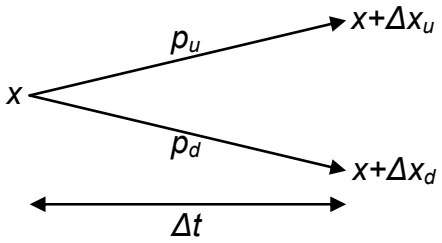

Figure 3.3: Simple Binomial Model of the Natural Logarithm of an Asset

Equating the mean and variance of the binomial process for x with the mean and variance of the continuous time process over the time interval Δt leads to:

$$\begin{aligned} \mathrm{E}[\Delta x] &= p_u \Delta x_u + p_d \Delta x_d = \nu \Delta t \\ \mathrm{E}[\Delta x^2] &= p_u \Delta x_u{}^2 + p_d \Delta x_d{}^2 = \sigma^2 \Delta t + \nu^2 \Delta t^2 \end{aligned} \tag{3.21}$$

As known $p_u + p_d = 1$ or $p_d = 1 - p_u$ just trivially substitute and obtain two equations in three unknowns.

The third condition is to set the jump sizes to be equal, which leads to

$$\begin{aligned} p_u(\Delta x) + p_d(-\Delta x) &= \nu\Delta t \\ p_u\Delta x^2 + p_d\Delta x^2 &= \sigma^2\Delta t + \nu^2\Delta t^2 \end{aligned} \tag{3.22}$$

and gives

$$\begin{aligned} \Delta x &= \sqrt{\sigma^2\Delta t + \nu^2\Delta t^2} \\ p_u &= \frac{1}{2} + \frac{1}{2}\frac{\nu\Delta t}{\Delta x} \end{aligned} \tag{3.23}$$

The disadvantage of this method is that its convergence is quite complicated. Moreover unsatisfying is that the error can actually increase with an increase in the number of time steps. The finite difference methods can solve this problem.

3.3 Additive Binomial Model

The structure of the general additive binomial model is similar to that of the multiplicative model [9]. As before the nodes in the tree are identified by a pair of indices (i,j), where $j = 0,1,\ldots,i$. So every node is i periods in the future and the asset has made j upwards moves to reach that node. Therefore the price of the underlying at node (i,j) is

$$S_{i,j} = \exp(x_{i,j}) = \exp(x + j\Delta x_u + (i-j)\Delta x_d) \tag{3.24}$$

After constructing the tree (see Figure 3.4) the value of the option at the maturity date can be calculated at the N^{th} time step. Working backwards through the tree each option price $C_{i,j}$ at every node is given by

$$C_{i,j} = e^{-r\Delta t}(\Delta x_u C_{i+1,j+1} + \Delta x_d C_{i,j+1}) \tag{3.25}$$

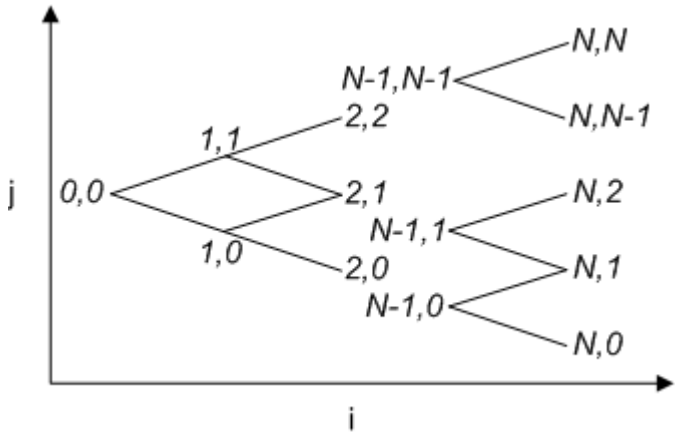

Figure 3.4: General Additive Binomial Tree

See chapter 12.6.2 for pricing examples.

3.3.1 Pricing Underlying Asset Paying a Continuous Dividend Yield

If the underlying asset for the construction of a binomial tree is a stock or a stock index that pay dividends the model has to be adapted.

In the case of a continuous dividend yield - which is mainly used for options on stock indices – the valuation is quite straight forward. In order to take into account the continuous dividend yield, just replace r by $(r - \delta)$ wherever it appears in the formulas[9]. For the variables Δx, p_u and ν the general additive formula with equal jump sizes changes to

$$\begin{aligned} \Delta x &= \sqrt{\sigma^2 \Delta t + \nu^2 \Delta t^2} \\ p_u &= \frac{1}{2} + \frac{1}{2}\frac{\nu \Delta t}{\Delta x} \\ \nu &= r - \delta - \frac{1}{2}\sigma^2 \end{aligned} \qquad (3.27)$$

Processing of the valuation is therefore the same as with an underlying asset which pays no dividend:

1. Constructing the tree and calculating the asset prices at each node
2. Valuating the several option prices at maturity date
3. Working backwards through the tree to calculate the price of the option today $(T=0)$

[9] Just as the stochastic differential equation changes to

$$dS = (r - \delta)Sdt + \sigma S dz \qquad (3.26)$$

in the Black-Scholes world.

3.3.2 Pricing Underlying Asset Paying a Known Discrete Cash Dividend

In the common case of a known cash dividend on the asset the situation becomes more difficult. Then the binomial tree gets non-recombining for nodes after the ex-dividend date. Figure 3.5 shows a binomial tree for an asset paying a cash amount D at a time τ where the condition $k\Delta t < \tau < (k+1)\Delta t$ is satisfied.

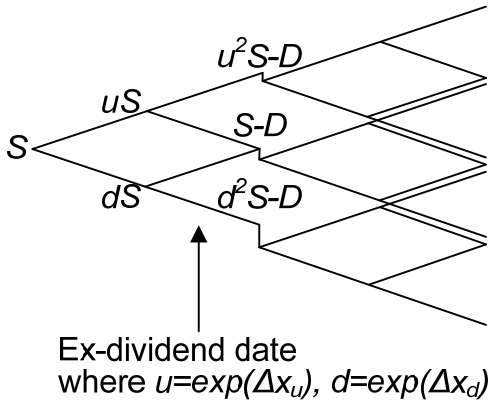

Figure 3.5: Binomial Tree with Adjustment for a Known Discrete Cash Dividend

For the time prior to the dividend date the tree nodes remain unchanged. Otherwise the value of the asset at node (i,j) becomes

$$S\exp(\Delta x_u)^j \exp(\Delta x_d)^{i-j} - D \qquad (3.28)$$

The number of nodes increases dramatically - at time $(k+m)\Delta t$ there are $m(k+1)$ nodes rather than $k+m+1$.

To handle this problem and obtain a recombining tree a particular assumption about the volatility of the asset price is made. It is supposed that the asset price S_t has two components. The uncertain part that is, $\tilde{S}_t$ and the certain part that is the present value of the future dividend stream. The value of $\tilde{S}_t$ is given by

$$\tilde{S}_t = S_t \text{ when } t > \tau \qquad (3.29)$$

and

$$\tilde{S}_t = S_t - De^{-r(\tau-t)} \text{ when } t \le \tau \qquad (3.30)$$

The volatility of $\tilde{S}_t$ is defined as $\tilde{\sigma}$ and assumed to be constant. The binomial tree parameters p_u, p_d, Δx_u, Δx_d are calculated in the usual way, but with σ replaced by $\tilde{\sigma}$. The binomial tree is constructed in the same way as before, where the value of the asset is

$$\tilde{S}_t \exp(\Delta x_u)^j \exp(\Delta x_d)^{i-j} + De^{-r(\tau - t)} \text{ when } t = i\Delta t < \tau$$

and

$$\tilde{S}_t \exp(\Delta x_u)^j \exp(\Delta x_d)^{i-j} \text{ when } t = i\Delta t > \tau.$$

See chapter 12.6.2.3 for a pricing example.

3.4 Binomial Model and Path-Dependent Options

Path-dependence with options explains how the payoff structure of such an option is limited or affected by the evolvement of the stock price of an underlying asset in course of time, even though past stock prices may no longer be relevant. So in this context path-dependence is used to mean simply ‘history matters’ e.g. the asset price on a fixing date or was the asset price above or below a specific level during an observation period (barrier option).

The binomial tree modeling even some exotic options - e.g. path-dependent options especially barrier options - can be priced using this method. Barrier options differ from standard options by a predetermined level *H,* the barrier level. If the asset price falls below or rises above the barrier the option knocks out and pays off nothing, or knocks in and starts to exist.

Due to the problems with accuracy, convergence and the simplicity of the tree structure binomial trees are not ideal. However, the simple tree structure gives the basic idea how to price those options.

Figure 3.6 gives an example how to price an American down-and-out call option and shows following three paths of the development of the asset price:

- Path 1 does not go below the barrier level and finishes above the strike price and therefore pays off.
- Path 2 does not go below the barrier level, but finishes below the strike price and therefore pays off zero.
- Path 3 goes below the barrier and therefore pays off nothing even though it finishes above the strike price.

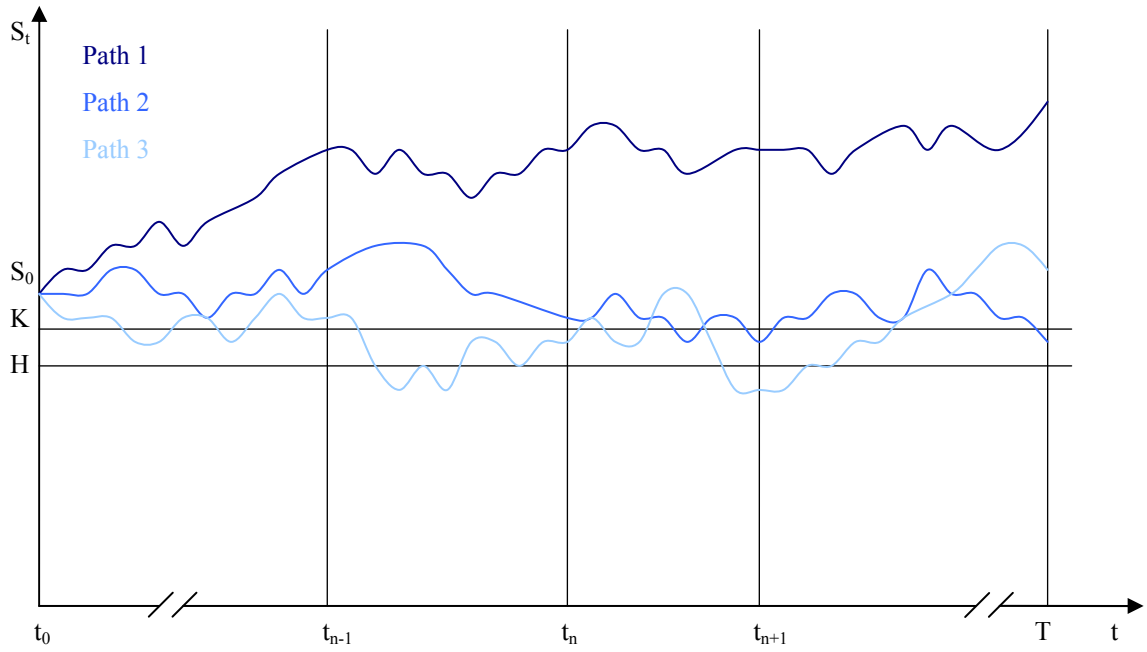

Figure 3.6: Different Asset Paths for a Down-and-Out Call Option

The pricing of this option in a binomial tree is similar to the early exercise problematic. At every node the value of the underlying asset must be compared to the barrier level *H* and if it is triggered - below the barrier - the option price at this node is set to zero.

See chapter 12.6.2.4 for a pricing example.

4 Trinomial Trees and Finite Difference Models

The binomial model showed some inefficiency regarding accuracy and convergence. To obtain more significant option prices the binomial model can be further adapted to a trinomial and implied tree structure respectively (based on [9]).

4.1 Trinomial Tree Model

The risk-neutral model of an underlying asset paying a continuous dividend yield has the following stochastic differential equation

$$dS = (r - \delta)Sdt + \sigma Sdz \tag{4.1}$$

Again it is more convenient to work in terms of $x = ln(S)$ which leads to

$$dx = \nu dt + \sigma dz$$

where (4.2)

$$\nu = r - \delta - \frac{1}{2}\sigma^2$$

Figure 4.1 shows a trinomial model of an asset which, over a small time interval Δt, can go up by Δx - the space step - stay the same or go down by Δx, with the probabilities p_u, p_m and p_d respectively[10].

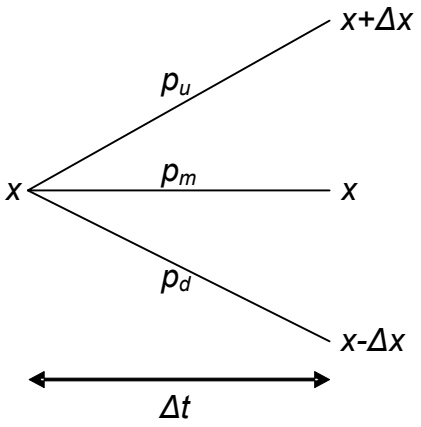

Figure 4.1: Simple Trinomial Tree Model of an Underlying Asset

[10] Δx cannot be chosen independently of Δt and a good choice is $\Delta x = \sigma\sqrt{3\Delta t}$.

The relationship between the continuous time process and the trinomial process of the drift and volatility parameters stated by Δx, p_u, p_m and p_d are shown in the following formulas:

$$E[\Delta x] = p_u(\Delta x) + p_m(0) + p_d(-\Delta x) = \nu\Delta t \tag{4.3}$$

$$E[\Delta x^2] = p_u(\Delta x^2) + p_m(0) + p_d(\Delta x^2) = \sigma^2\Delta t + \nu^2\Delta t^2 \tag{4.4}$$

$$p_u + p_m + p_d = 1 \tag{4.5}$$

Solving equations (4.3) to (4.5) gives

$$p_u = \frac{1}{2}\left(\frac{\sigma^2\Delta t + \nu^2\Delta t^2}{\Delta x^2} + \frac{\nu\Delta t}{\Delta x}\right) \tag{4.6}$$

$$p_m = 1 - \frac{\sigma^2\Delta t + \nu^2\Delta t^2}{\Delta x^2} \tag{4.7}$$

$$p_d = \frac{1}{2}\left(\frac{\sigma^2\Delta t + \nu^2\Delta t^2}{\Delta x^2} - \frac{\nu\Delta t}{\Delta x}\right) \tag{4.8}$$

The one step trinomial model as shown in Figure 4.1 can be extended to form a complete trinomial tree (Figure 4.2) where i represents the time step and j represents the level of the asset price relative to the initial asset price. Thus at a certain node $(i\ j)$ there is $t = i\Delta t$, and $S_{i,j} = S\ exp(j\Delta x)$ and the option price is $C_{i,j}$. The values of the option at maturity ($T = N\Delta t$) is given by the known pay-off, for example for a call option

$$C_{N,j} = \max(0, S_{N,j} - K) \tag{4.9}$$

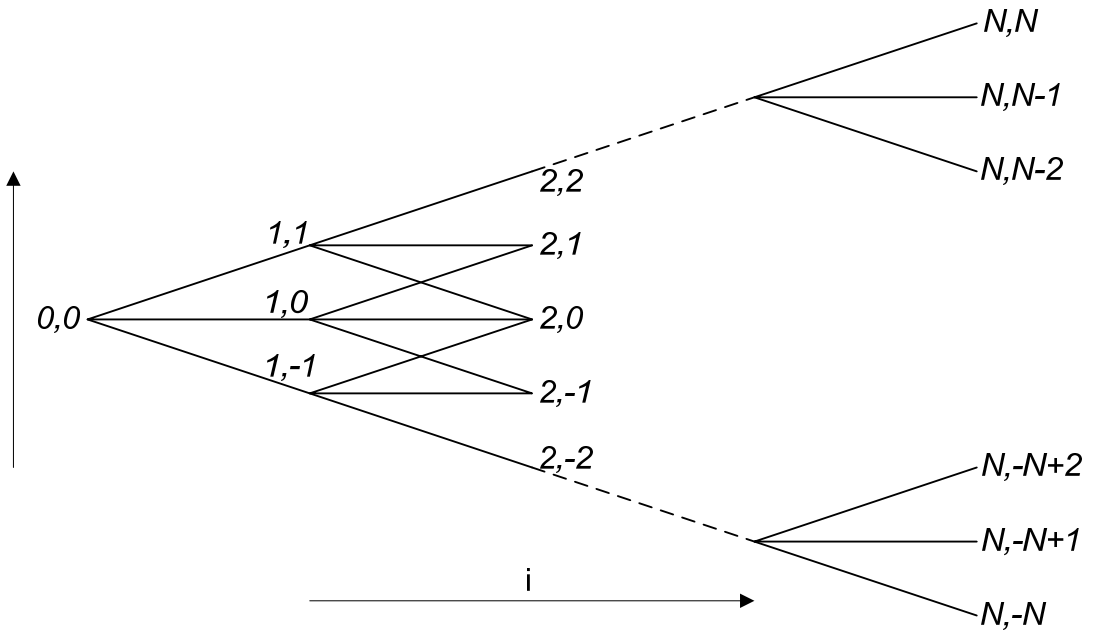

Figure 4.2: Structure of the Trinomial Tree

Again we can compute option values as discounted expectations in a risk-neutral world. The option values of earlier nodes are computed by discounting their predecessors with the corresponding probabilities.

$$C_{i,j} = e^{-r\Delta t}(p_u C_{i+1,j+1} + p_m C_{i+1,j} + p_d C_{i+1,j-1}) \qquad (4.10)$$

Although much more data has to be computed within a trinomial tree it shows advantages over the binomial tree:

- much better approximation to the continuous time process for the same number of time steps
- easy to work with because of its more regular lattice and higher flexibility
- comfortable extension to time-varying drift and volatility parameters

See chapter 12.6.3.1 for a pricing example.

4.2 Finite Difference Models

4.2.1 Explicit Finite Difference Models

A related model approach to solve the problem of option valuation, that takes the advantages of a trinomial tree into account, are finite difference methods. The idea behind is simplifying the Black-Scholes partial differential equation (4.11) by replacing the partial differentials with finite differences [12] and [13].

$$-\frac{\partial C}{\partial t} = \frac{1}{2}S^2\sigma^2\frac{\partial^2 C}{\partial S^2} + (r-\delta)S\frac{\partial C}{\partial S} - rC \qquad (4.11)$$

Again it is more convenient to work in terms of $x = ln(S)$ which leads to

$$-\frac{\partial C}{\partial t} = \frac{1}{2}\sigma^2\frac{\partial^2 C}{\partial x^2} + \nu\frac{\partial C}{\partial x} - rC \qquad (4.12)$$

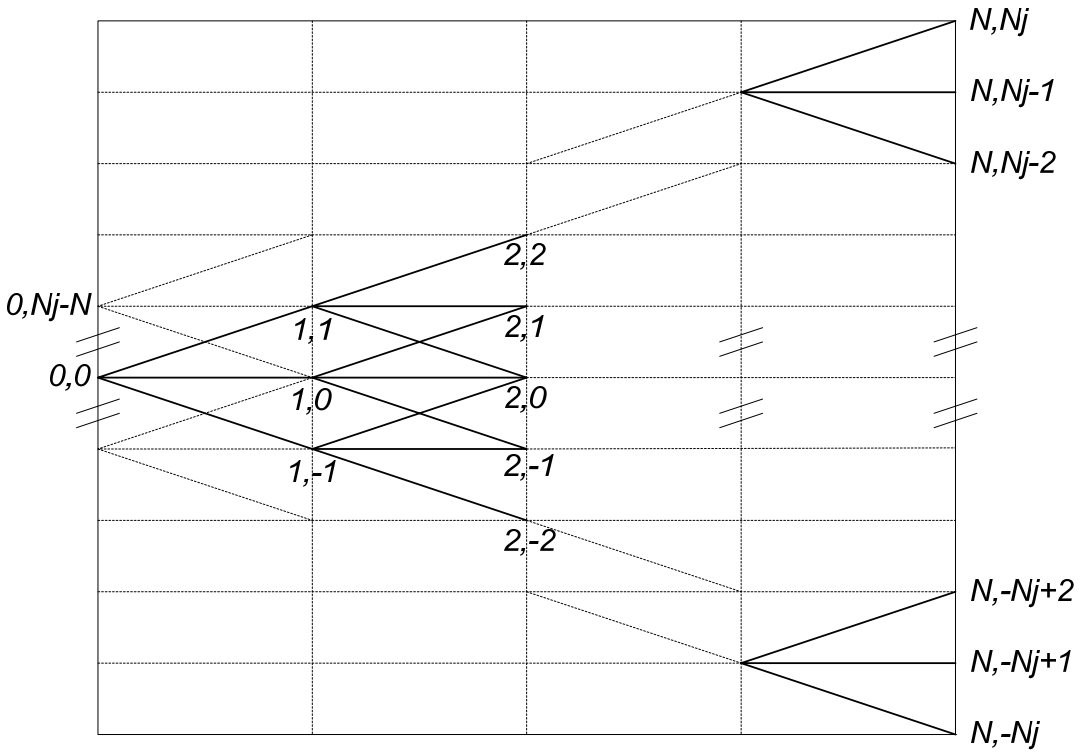

Figure 4.3: Lattice for Finite Difference Approach

An approximation in equation (4.12) is used to obtain the explicit finite difference method – for $\partial C/\partial t$ a forward difference is used and for $\partial^2 C/\partial x^2$ and $\partial C/\partial x$ central differences are used. Therefore, the terms of the lattice are

$$-\frac{C_{i+1,j} - C_{i,j}}{\Delta t} = \frac{1}{2}\sigma^2\frac{C_{i+1,j+1} - 2C_{i+1,j} + C_{i+1,j-1}}{\Delta x^2} + \nu\frac{C_{i+1,j+1} - C_{i+1,j-1}}{2\Delta x} - rC_{i+1,j} \qquad (4.13)$$

which can be rearranged to

$$C_{i,j} = p_u C_{i+1,j+1} + p_m C_{i+1,j} + p_d C_{i+1,j-1} \qquad (4.14)$$

$$p_u = \Delta t\left(\frac{\sigma^2}{2\Delta x^2} + \frac{\nu}{2\Delta x}\right) \tag{4.15}$$

$$p_m = 1 - \Delta t\frac{\sigma^2}{\Delta x^2} - r\Delta t \tag{4.16}$$

$$p_d = \Delta t\left(\frac{\sigma^2}{2\Delta x^2} - \frac{\nu}{2\Delta x}\right) \tag{4.17}$$

Equation (4.14) is equivalent to the discounted expectations approach. This can be demonstrated by taking a slightly different approximation (to the partial differential equation) for the value at node *(i,j)* rather than *(i+1,j)* in the last term of equation (4.13)

$$-\frac{C_{i+1,j} - C_{i,j}}{\Delta t} = \frac{1}{2}\sigma^2\frac{C_{i+1,j+1} - 2C_{i+1,j} + C_{i+1,j-1}}{\Delta x^2} + \nu\frac{C_{i+1,j+1} - C_{i+1,j-1}}{2\Delta x} - rC_{i,j} \tag{4.18}$$

Which can be rewritten as

$$C_{i,j} = \frac{1}{1 + r\Delta t}(p_u C_{i+1,j+1} + p_m C_{i+1,j} + p_d C_{i+1,j-1}) \tag{4.19}$$

$$p_u = \frac{1}{2}\Delta t\left(\frac{\sigma^2}{\Delta x^2} + \frac{\nu}{\Delta x}\right) \tag{4.20}$$

$$p_m = 1 - \Delta t\frac{\sigma^2}{\Delta x^2} \tag{4.21}$$

$$p_d = \frac{1}{2}\Delta t\left(\frac{\sigma^2}{\Delta x^2} - \frac{\nu}{\Delta x}\right) \tag{4.22}$$

, where $1/(1+r\Delta t)$ is an approximation of $1/e^{r\Delta t}$. Therefore the explicit finite difference method is equivalent to approximating the diffusion process by a discrete trinomial process.

The relationship between the lattice values in equation (4.14) is shown in Figure 4.4.

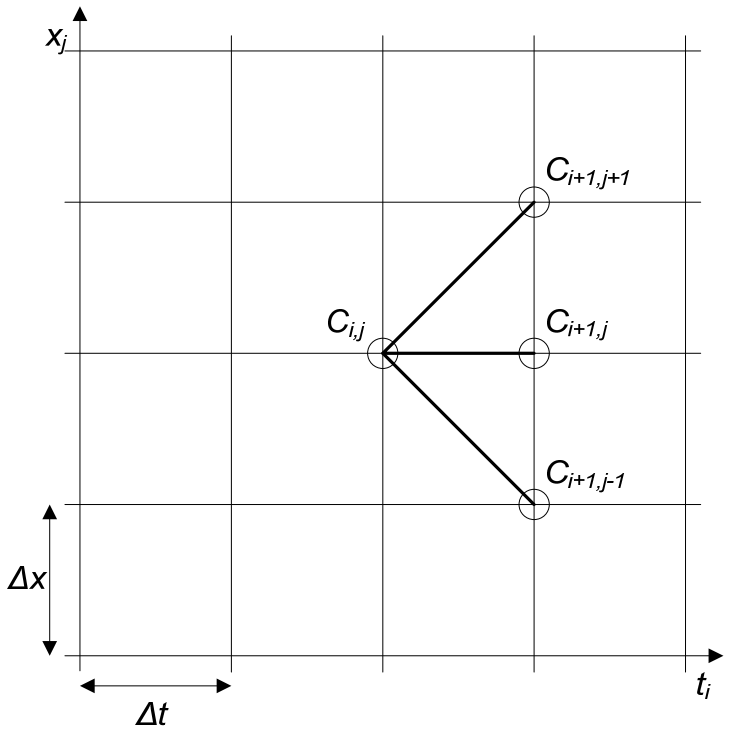

Figure 4.4: Structure of the Explicit Finite Difference Model

See chapter 12.6.3.2 for pricing examples.

Because the accuracy of this method is $O(\Delta x^2 + \Delta t)$ the error can be halved when $\Delta x^2 + \Delta t$ is halved. Therefore the time step must be halved, but the space step only needs to be reduced by a factor of $1/\sqrt{2}$.

To ensure stability and convergence of the finite difference method the following conditions must be fulfilled:

- the probabilities p_u, p_m and p_d have to be positive
- the condition $\Delta x \geq \sigma\sqrt{3\Delta t}$ has to be satisfied
- the convergence condition, that is the discretization error – i.e. the difference between the exact solution of the partial differential equation and the solution of the finite difference equation – must tend to zero as space and time steps tend to zero
- the stability condition, that is the round-off error – i.e. the difference between the solution of the finite difference equation and the numerically computed solution – must be small and remain bounded

4.2.2 Implicit Finite Difference Models

Again it is more convenient to work in terms of $x = ln(S)$ which leads as before to the well know Black-Scholes partial differential equation.

$$-\frac{\partial C}{\partial t}=\frac{1}{2}\sigma^2\frac{\partial^2 C}{\partial x^2}+\nu\frac{\partial C}{\partial x}-rC \tag{4.12}$$

Transforming the equation (4.12) by replacing the space derivatives with central differences at time step *i* rather than at *i+1* gives

$$-\frac{C_{i+1,j}-C_{i,j}}{\Delta t}=\frac{1}{2}\sigma^2\frac{C_{i,j+1}-2C_{i,j}+C_{i,j-1}}{\Delta x^2}+\nu\frac{C_{i,j+1}-C_{i,j-1}}{2\Delta x}-rC_{i,j} \tag{4.23}$$

Which can be rearranged to

$$p_u C_{i,j+1}+p_m C_{i,j}+p_d C_{i,j-1}=C_{i+1,j} \tag{4.24}$$

$$p_u=-\frac{1}{2}\Delta t\left(\frac{\sigma^2}{\Delta x^2}+\frac{\nu}{\Delta x}\right) \tag{4.25}$$

$$p_m=1+\Delta t\frac{\sigma^2}{\Delta x^2}+r\Delta t \tag{4.26}$$

$$p_d=-\frac{1}{2}\Delta t\left(\frac{\sigma^2}{\Delta x^2}-\frac{\nu}{\Delta x}\right) \tag{4.27}$$

The relationship between the lattice values is shown in Figure 4.5.

The equation (4.24) for each node *(i,j)* with $j = -N_j+1,\ldots, N_j-1$ cannot be solved individually for the option values at time step *i.* Therefore they must be considered, together with the boundary conditions,

$$C_{i,N_j}-C_{i,N_{j-1}}=\lambda_U \tag{4.28}$$

$$C_{i,-N_{j+1}}-C_{i,-N_j}=\lambda_L \tag{4.29}$$

to be a system of $2N_j+1$ linear equations which implicitly determine the $2N_j+1$ option values at time step *i.* The boundary condition parameters λ_U and λ_L are determined by the type of option being valued, for example for a call we have

$$\lambda_U=S_{i,N_j}-S_{i,N_{j-1}} \tag{4.30}$$

$$\lambda_L=0 \tag{4.31}$$

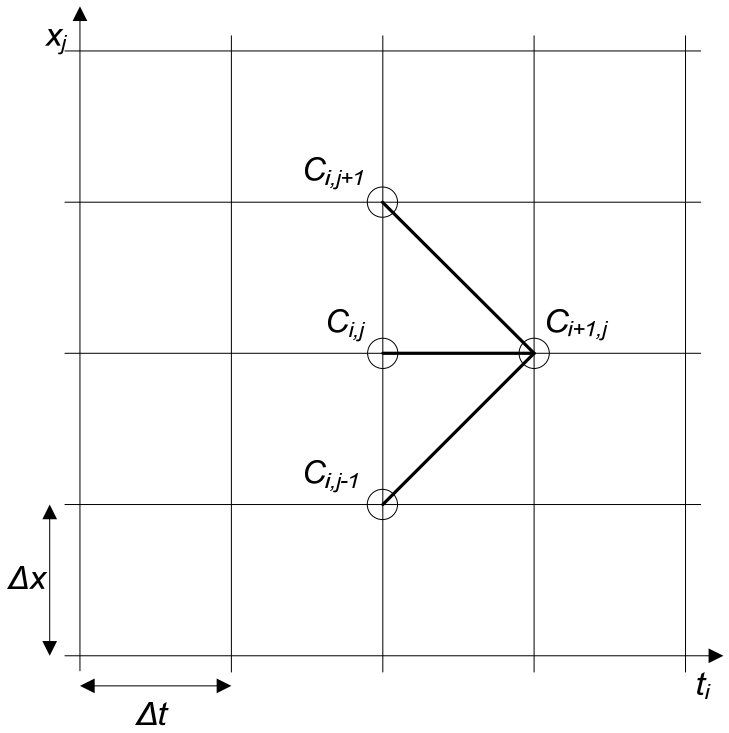

Figure 4.5: Structure of the Implicit Finite Difference Model

The equation set has a special structure which is called tri-diagonal. Each equation has two variables in common with the equation above and below. When writing the equation set in matrix form the tri-diagonal structure can clearly be seen:

$$\begin{vmatrix} 1 & -1 & 0 & \dots & \dots & \dots & 0 \\ p_u & p_m & p_d & 0 & \dots & \dots & 0 \\ 0 & p_u & p_m & p_d & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & p_u & p_m & p_d & 0 \\ 0 & \dots & \dots & 0 & p_u & p_m & p_d \\ 0 & \dots & \dots & \dots & 0 & 1 & -1 \end{vmatrix} \begin{vmatrix} C_{i,Nj} \\ C_{i,Nj-1} \\ C_{i,Nj-2} \\ \dots \\ C_{i,-Nj+2} \\ C_{i,-Nj+1} \\ C_{i,-Nj} \end{vmatrix} = \begin{vmatrix} \lambda_U \\ C_{i+1,Nj-1} \\ C_{i+1,Nj-2} \\ \dots \\ C_{i+1,-Nj+2} \\ C_{i+1,-Nj+1} \\ \lambda_L \end{vmatrix}$$

Figure 4.6: Matrix Form of Tri-Diagonal Equation Set

Solving this tri-diagonal matrix equation can be done very efficiently. Beginning with the boundary condition equation $j = -N_j$ this equation is rewritten to obtain

$$C_{i,-N_j} = C_{i,-N_{j+1}} - \lambda_L \qquad (4.32)$$

Substituting *(j = -N_j+1)* into the equation above

$$p_u C_{i,-N_{j+2}} + p_m C_{i,-N_{j+1}} + p_d (C_{i,-N_{j+1}} - \lambda_L) = C_{i+1,-N_{j+1}} \tag{4.33}$$

is obtained, which can be rearranged to

$$p_u C_{i,-N_{j+2}} + p'_m C_{i,-N_{j+1}} = p' \tag{4.34}$$

where

$$p'_m = p_m + p_d \qquad \text{and} \qquad p' = C_{i+1,-N_{j+1}} + p_d \lambda_L$$

Thus the original equation with three unknowns has become equation (4.34) with only two unknowns.

Equation (4.34) can be rewritten to

$$C_{i,-N_{j+1}} = \frac{p' - p_u C_{i,-N_{j+2}}}{p'_m} \tag{4.35}$$

Substituting *(j = -N_j+2)* into the equation for

$$p_u C_{i,-N_{j+3}} + p'_m C_{i,-N_{j+2}} = p' \tag{4.36}$$

is obtained, where

$$p'_m = p_m - \frac{p_u}{p'_{m,-N_{j+1}}} p_d \qquad \text{and} \qquad p' = C_{i+1,-N_{j+2}} - \frac{p'_{-N_{j+1}}}{p'_{m,-N_{j+1}}} p_d$$

(the added subscripts to the *p's* indicate the application to the equation for *j* = -N_j+*1*).

This process of substitution can be repeated all the way up to *j* = N_j-*1* obtaining:

$$p_u C_{i,N_j} + p'_m C_{i,N_{j-1}} = p' \tag{4.37}$$

Using equation (4.35) and the boundary condition equation for $j = N_j$

$$C_{i,N_j} - C_{i,N_{j-1}} = \lambda_U \tag{4.38}$$

can be solved for both $C_{i,Nj}$ and $C_{i,Nj-1}$. To obtain $C_{i,Nj-2}$ the next equation down for $j = N_j\text{-}2$ and $C_{i,Nj-1}$ are used. This process called back-substitution can be repeated all the way down to $j = -N_j$, thus solving the complete tri-diagonal system of equations (Figure 4.6).

See chapter 12.6.3.3 for a pricing example.

While the accuracy of the implicit finite difference method has the same order as the explicit finite difference method - $O(\Delta x^2+\Delta t)$ - more importantly, it is unconditionally stable and convergent. Thus gives more freedom to trade-off accuracy for speed by decreasing the time steps because there is no need to worry about a stability and convergence condition. The values of p_u, p_m and p_d can no longer be interpreted as probabilities, p_u and p_d will typically be negative while p_m will be greater than one. But, it can be proofed that the implicit finite difference approximation is equivalent to a generalized discrete stochastic process where the asset price may jump to every node in the lattice at the next time step.

4.2.3 Crank-Nicolson Finite Difference Models

A further refinement of the implicit finite difference method is the Crank-Nicolson method. It replaces the space and time derivatives with finite differences centered at an imaginary time step at $(i+1/2)$ and is also called a fully centered method. The Crank-Nicolson finite difference equation looks as follows:

$$-\frac{C_{i+1,j}-C_{i,j}}{\Delta t} = \frac{1}{2}\sigma^2\left(\frac{(C_{i+1,j+1}-2C_{i+1,j}+C_{i+1,j-1})+(C_{i,j+1}-2C_{i,j}+C_{i,j-1})}{2\Delta x^2}\right)$$
$$+\nu\left(\frac{(C_{i+1,j+1}-C_{i+1,j-1})+(C_{i,j+1}-C_{i,j-1})}{4\Delta x}\right)-r\left(\frac{C_{i+1,j}+C_{i,j}}{2}\right) \tag{4.39}$$

Which can be rearranged to

$$p_u C_{i,j+1} + p_m C_{i,j} + p_d C_{i,j-1} = -p_u C_{i+1,j+1} - (p_m - 2)C_{i+1,j} - p_d C_{i+1,j-1} \tag{4.40}$$

$$p_u = -\frac{1}{4}\Delta t\left(\frac{\sigma^2}{\Delta x^2}+\frac{\nu}{\Delta x}\right) \tag{4.41}$$

$$p_m = 1+\Delta t\frac{\sigma^2}{2\Delta x^2}+\frac{r\Delta t}{2} \tag{4.42}$$

$$p_d = -\frac{1}{4}\Delta t\left(\frac{\sigma^2}{\Delta x^2} - \frac{\nu}{\Delta x}\right) \tag{4.43}$$

The right-hand side of equation (4.40) consists of known option values and the known constant coefficients p_u, p_m, p_d and can therefore be considered a known constant. Together with the boundary conditions,

$$C_{i,N_j} - C_{i,N_{j-1}} = \lambda_U \tag{4.44}$$

$$C_{i,-N_{j+1}} - C_{i,-N_j} = \lambda_L \tag{4.45}$$

the set of equations (4.40) – (4.43) for $j = -N_j+1,...,N_j-1$ build again a tri-diagonal system of equations. The solution of these equations can be efficiently done very similar to the implicit finite difference method above.

The relationship between the lattice values in equation (4.40) is illustrated in Figure 4.7.

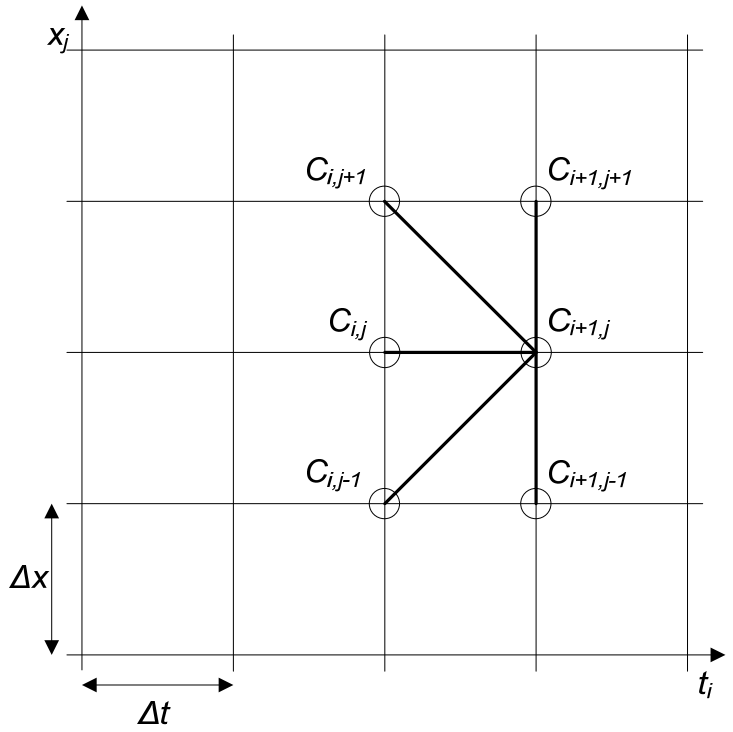

Figure 4.7: Structure of the Crank-Nicolson Finite Difference Model

The accuracy of the Crank-Nicolson method is

$$O\left(\Delta x^2 + \left(\frac{\Delta t}{2}\right)^2\right)$$

and is unconditionally stable and convergent. However, this method converges much faster than the implicit or explicit finite difference methods.

Again, the values of p_u, p_m and p_d can no longer be interpreted as probabilities, p_u and p_d will typically be negative while p_m will be greater than one. But, it can be proofed that the Crank-Nicolson finite difference approximation is equivalent to a generalized discrete stochastic process where the asset price may jump to every node in the lattice at the next time step.

See chapter 12.6.3.4 for a pricing example.

5 Implied Trees and Exotic Options

The methods discussed so far are only applicable for standard options and some specific exotic options such as down-and-out. Furthermore the market's expectation of the future in terms of market prices of standard European options is not covered as all parameters are time constant.

Generalizing the binomial and trinomial trees by making previously constant parameters (such as the probabilities) time dependent and implying these time-dependent parameters is the idea behind implied trees. In this way it is recognized that the real market is incomplete without the standard options and so the standard options should be treated as fundamental securities which prices are observed in the market (based on [9]).

5.1 Basics of the Implied Tree Model

The structure of the implied trinomial tree will be very similar to that of the constant coefficient trinomial tree.

Here, at each node there is a state price Q_{ij}[11], which is interpolated and/or extrapolated market data to obtain the required strike and maturity needed. Furthermore instead of a single set of transition probabilities p_u, p_m and p_d a different set of transition probabilities $p_{u,i,j}$, $p_{m,i,j}$, and $p_{d,i,j}$ for every node *(i j)* is used. The value of an option at node *(i,j)* will be $C_{i,j}$ as before, and time step N will correspond to the maturity date.

This has the following advantages without complicating the tree-building procedure:

- the time steps Δt_i can be different
- the asset price levels can vary with the time step
- convenient modeling of the tree ensuring that time steps fall on key dates required for the exotic options

[11] See Black-Scholes chapter for the derivation of state prices.

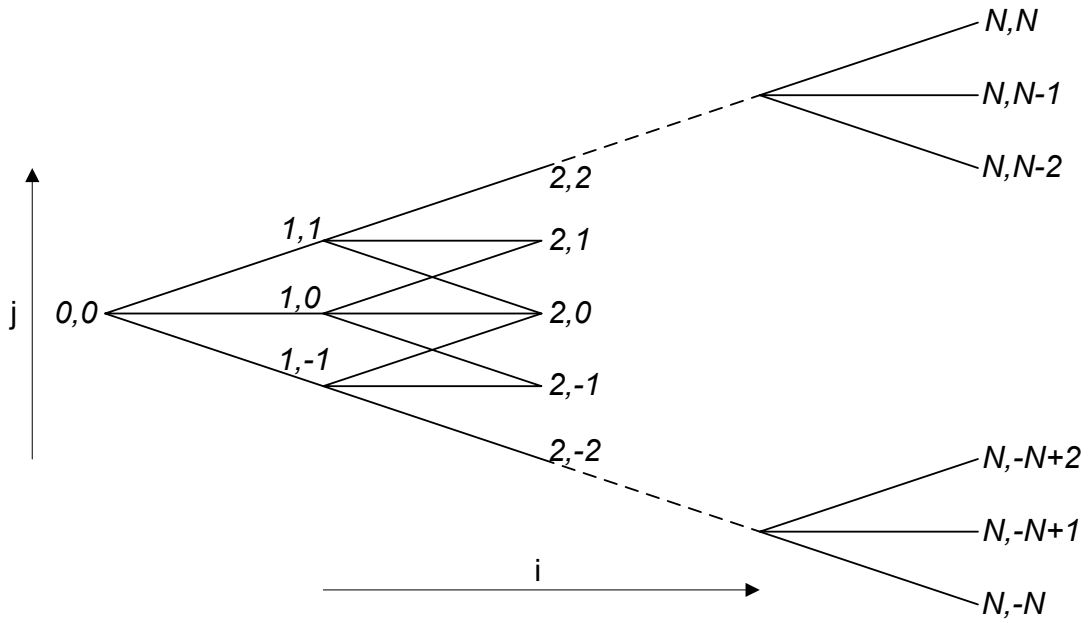

Figure 5.1: Structure of the Implied Trinomial Tree

5.2 Implied State Prices

The state prices for the nodes at time step N in the tree should be computed such that they are consistent with the market prices of standard European call and put options. Consider the highest node in the tree *(N,N)* at time step *N*. The price of an European call with strike price $S_{N,N-1}$ - asset price at the next node down -, and with a maturity date at time step N is

$$c(S_{N,N-1}, N\Delta t) = (S_{N,N} - S_{N,N-1})Q_{N,N} \tag{5.1}$$

because for all the nodes below *(N,N)* the pay-off of the call option is zero. Equation (5.1) can be rewritten to give the state price $Q_{N,N}$ at node *(N,N)* in terms of the known call price, asset price and strike price. The price of an European option with a strike price equal to the asset price $S_{N,N-2}$ at node *(N,N*-2), is given by

$$c(S_{N,N-2}, N\Delta t) = (S_{N,N-1} - S_{N,N-2})Q_{N,N-1} + (S_{N,N} - S_{N,N-2})Q_{N,N} \tag{5.2}$$

The only unknown in equation (5.2) is $Q_{N,N-1}$ since $Q_{N,N}$ was previously computed. Working down the nodes at time step N to the middle of the tree computing the state prices the option price, for node *(N,j),* is given by

$$c(S_{N,j-1}, N\Delta t) = (S_{N,j} - S_{N,j-1})Q_{N,j} + \sum_{k=j+1}^{N}(S_{N,k} - S_{N,j-1})Q_{N,k} \tag{5.3}$$

where everything is known except $Q_{N,J}$. From the bottom node of the tree working upwards to the central node of the tree this procedure can be started using put option prices, because of the iterative nature of the calculations numerical errors can build up in the state prices using call prices. This method can be applied to every time step in the tree.

5.3 Implied Transition Probabilities

From the calculated state prices at every node in the tree, where the local no-arbitrage relationships must hold, the transition probabilities are obtained. Assuming that the transition probabilities for all the nodes above node *(i,j)* are already computed, the transition probabilities for node *(i,j)* can be computed according to the following conditions:

$$e^{-r\Delta t}(p_{d,i,j} + p_{m,i,j} + p_{u,i,j}) = e^{-r\Delta t} \tag{5.4}$$

The first condition requires that the transition probabilities sum up to one.

$$p_{d,i,j} + p_{m,i,j} + p_{u,i,j} = 1 \tag{5.5}$$

The second condition is that the asset price at node *(i,j)* has to be equal to its local discounted expected value over the next time step.

$$S_{i,j} = e^{-r\Delta t}(p_{d,i,j}S_{i+1,j-1} + p_{m,i,j}S_{i+1,j} + p_{u,i,j}S_{i+1,j+1}) \tag{5.6}$$

Finally, the forward evolution equation for the state price at node *(i+1,j+1)* is:

$$Q_{i+1,j+1} = e^{-r\Delta t}(p_{d,i,j+2}Q_{i,j+2} + p_{m,i,j+1}Q_{i,j+1} + p_{u,i,j}Q_{i,j}) \tag{5.7}$$

Given the transition probabilities for all the nodes above *(i,j)* equation (5.7) can be rewritten to:

$$p_{u,i,j} = \frac{e^{r\Delta t}Q_{i+1,j+1} - p_{d,i,j+2}Q_{i,j+2} - p_{m,i,j+1}Q_{i,j+1}}{Q_{i,j}} \tag{5.8}$$

As well for $p_{m,i,j}$ and $p_{d,i,j}$

$$p_{m,i,j} = \frac{e^{r\Delta t} S_{i,j} - S_{i+1,j-1} - p_{u,i,j}(S_{i+1,j+1} - S_{i+1,j-1})}{(S_{i+1,j} - S_{i+1,j-1})} \tag{5.9}$$

$$p_{d,i,j} = 1 - p_{m,i,j} - p_{u,i,j} \tag{5.10}$$

The relationship diagram between the state prices and transition probabilities is shown in Figure 5.2.

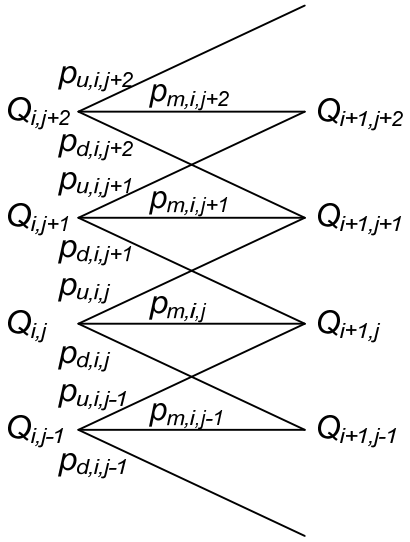

Figure 5.2: State Prices and Transition Probabilities

For the highest node *(i,i)* at time step *i,* and for node *(i,i-1)* equation (5.8) reduces to

$$p_{u,i,i} = \frac{e^{r\Delta t} Q_{i+1,i+1}}{Q_{i,i}} \tag{5.11}$$

and

$$p_{u,i,i-1} = \frac{e^{r\Delta t} Q_{i+1,i} - p_{m,i,i} Q_{i,i}}{Q_{i,i-1}} \tag{5.12}$$

using equations (5.9) and (5.10) to obtain $p_{m,i,j}$ and $p_{d,i,j}$ respectively $p_{m,i,i-1}$ and $p_{d,i,i-1}$.

Starting at the top of the tree the transition probabilities can be solved in an iterative manner working downwards. Again to avoid numerical errors for the lower part of the tree the process is stopped at the central node. For the lower half of the tree $p_{d,i,j}$

is directly obtained from the forward evolution of the state prices, and then $p_{m,i,j}$ and $p_{u,i,j}$ are obtained by solving the remaining two equations simultaneously.

Therefore, equations (5.8) – (5.12) become

$$p_{d,i,j} = \frac{e^{r\Delta t} Q_{i+1,j-1} - p_{u,i,j-2} Q_{i,j-2} - p_{m,i,j-1} Q_{i,j-1}}{Q_{i,j}} \tag{5.13}$$

$$p_{m,i,j} = \frac{e^{r\Delta t} S_{i,j} - S_{i+1,j+1} - p_{d,i,j}(S_{i+1,j-1} - S_{i+1,j+1})}{(S_{i+1,j} - S_{i+1,j+1})} \tag{5.14}$$

$$p_{u,i,j} = 1 - p_{m,i,j} - p_{d,i,j} \tag{5.15}$$

$$p_{d,i,-i} = \frac{e^{r\Delta t} Q_{i+1,-i-1}}{Q_{i,-i}} \tag{5.16}$$

$$p_{d,i,-i+1} = \frac{e^{r\Delta t} Q_{i+1,-i} - p_{m,i,-i} Q_{i,-i}}{Q_{i,-i+1}} \tag{5.17}$$

To ensure that the transition probabilities remain positive it is necessary that the explicit finite difference method stability condition ($\Delta x \geq \sigma \sqrt{3 \Delta t}$) is satisfied at every node.

$$\mathrm{var}[\Delta x] = \sigma^2_{local} \Delta t = \mathrm{E}[\Delta x^2] - (\mathrm{E}[\Delta x])^2 \tag{5.18}$$

A simple and robust way to meet this condition is to set the space step as follows:

$$\Delta x = \sigma_{\max} \sqrt{3 \Delta t} \tag{5.19}$$

Where σ_{max} is the maximum implied volatility from the standard options to which the tree is being fitted.

See chapter 12.6.4.1 for a pricing example.

5.4 Exotic Options and Implied Trees

Plain Vanilla options share certain characteristics such as one underlying asset or the fact that the payoff depends only on the underlying asset at maturity. Further they are defined as a call option or a put option and the payoff is determined as the difference between the asset price and the strike price.

The particular feature of exotic options is to soften the restrictions of vanilla options. So the payoff additionally can be dependent of the average asset price on different fixing dates (asian option). In another case the payoff is dependent on whether the asset price was above or below a specific level during an observation period (barrier option). Here the path taken by these exotic options is of prime importance and therefore they are called path-dependent options. Also it is possible that the payoff is determined on the weighted average of several underlying assets (basket option) or the option is not predefined as a call or put option (chooser option). A vast number of other exotic options exist which are not categorized in to an own community.

5.4.1 Pricing Barrier Options

The difference between standard options and barrier options is that they appear or disappear only if the underlying asset price hits a predetermined level - H - the barrier [14].

There are three parameters to be defined:

- barrier level: defines if the barrier is below or above the current asset price - down or up
- barrier condition: defines whether the option disappears or appears when the barrier is crossed - out or in
- option type: as for standard options - call or put

A down-and-out call option for example, has the pay-off of a standard call option except if the underlying asset price goes down below the barrier level H then the option disappears and pays nothing. The pay-off of a down and out call can be expressed as follows:

$$\max(0, S_T - K)\big|_{\min(S_{t1},\ldots,S_{tm})>H}$$

In Table 5.1 all 8 possible parameter combinations are listed with a mathematical definition of their pay-off.

Name	pay-off
Down and out call	$\max(0,S_T - K) \mid \min(S_{t1} \ldots S_{tm})>H$
Up and out call	$\max(0,S_T - K) \mid \max(S_{t1} \ldots S_{tm})<H$
Down and in call	$\max(0,S_T - K) \mid \min(S_{t1} \ldots S_{tm})\leq H$
Up and in call	$\max(0,S_T - K) \mid \max(S_{t1} \ldots S_{tm})\geq H$
Down and out put	$\max(0,K - S_T) \mid \min(S_{t1} \ldots S_{tm})>H$
Up and out put	$\max(0,K - S_T) \mid \max(S_{t1} \ldots S_{tm})<H$
Down and in put	$\max(0,K - S_T) \mid \min(S_{t1} \ldots S_{tm})\leq H$
Up and in put	$\max(0,K - S_T) \mid \max(S_{t1} \ldots S_{tm})\geq H$
where $\mid_{\text{condition}}$ is the indicator function which has value one if condition is true and zero otherwise	

Table 5.1: Different Barrier Options

An example of three possible developments of the underlying asset price for an American down-and-out call option is given in Figure 3.6 – chapter 'The Binomial Model'.

Barrier options are generally cheaper than standard options because of the possibility that the option disappears or never appears. As the asset price becomes very low relative to the strike price the chances of it finishing in the money are very low, while with a standard option the buyer still pays for this chance. A standard variation on the barrier family are barrier options which pay a predetermined cash rebate (X_{rebate}) if an "out" option disappears or an "in" option never appears.

The procedure for calculating the barrier option price is quite similar to the pricing of standard options except that the barrier boundary condition is added. For a down-and-out call option this means, when stepping back through the tree at every node where the underlying asset price is below the barrier the option price is set equal to the rebate amount - which may be zero [9].

See chapter 12.6.4.2 for a pricing example.

5.4.2 Pricing Look-Back Options

The difference between standard options and look-back options is that either the final asset price or the strike price is set equal to the minimum or maximum asset price observed on one of a set of predetermined fixing dates, t_i; $i = 1,..., m$ [15].

There are two parameters to be defined:

- look-back condition: defines whether the asset price or the strike price is replaced - fixed strike or floating strike
- option type: as for standard options - call or put

In Table 5.2 all 4 possible parameter combinations are listed with a mathematical definition of their pay-off.

Name	pay-off
Fixed strike look-back call	$\max(0,\max(S_{t1} \ldots S_{tm})-K)$
Fixed strike look-back put	$\max(0,K-\min(S_{t1} \ldots S_{tm}))$
Floating strike look-back call	$\max(0, S_T-\min(S_{t1} \ldots S_{tm}))$
Floating strike look-back put	$\max(0,\max(S_{t1} \ldots S_{tm})- S_T)$

Table 5.2: Different Look-Back Options

A fixed strike call option for example, has the pay-off of a standard call option, except that the asset price at the maturity date is replaced by the maximum asset price that occurred over the set of fixings specified.

Figure 5.3 illustrates two possible paths of the underlying asset price:

- Path 1: the maximum level of the asset price at the fixing dates occurs at fixing date t_1 - this is below the strike price K and so the pay-off at T is zero.
- Path 2: the maximum occurs at fixing date t_3, - is above the strike price and so the pay-off is St_3-K even though the path finishes below the strike price.

For the floating strike look-backs, if the maturity date is a fixing date then they are not really options since they will always be exercised. That is the worst pay-off that can occur is zero if the price at maturity is the maximum or minimum of the observed prices.

Look-back options thus allow the holder to buy or sell the underlying asset for the best of the observed prices.

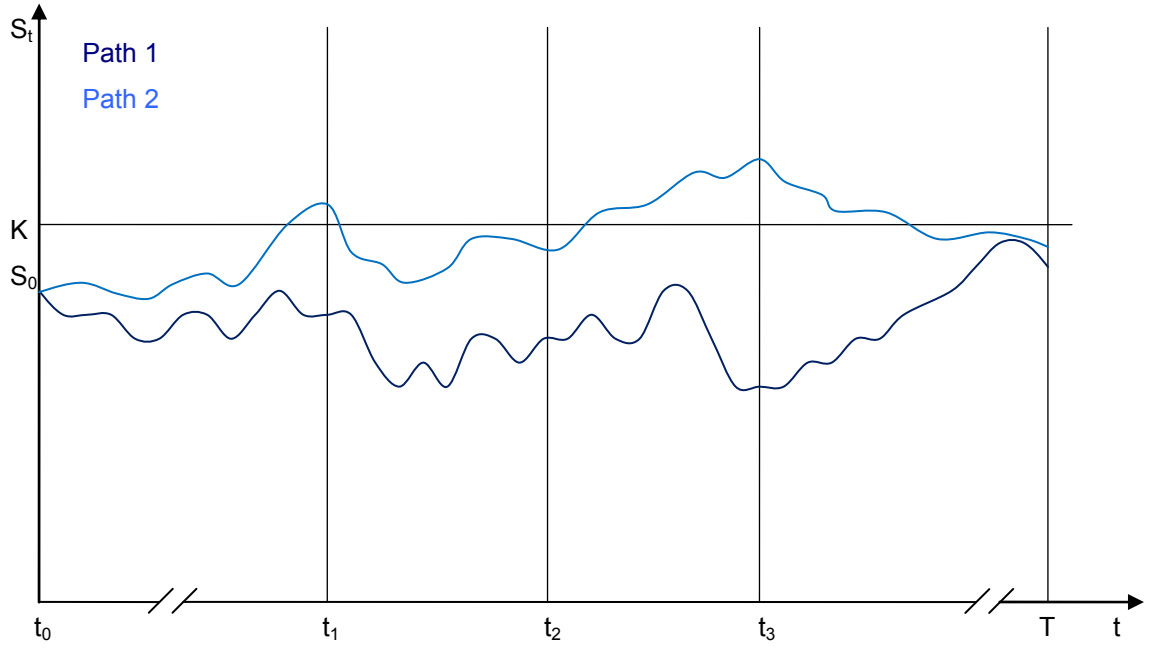

Figure 5.3: Fixed Strike Look-back Call Option Example Paths

Because of their path dependency the pricing of look-back options in trees is complicated. This means, the value of the look-back option at any node in the tree depends on the current maximum or minimum asset price, which in turn depends on the path the asset price took to reach that node. As there can be many different paths through a tree to a particular node (see Figure 5.4) the look-back option can have many different values at a particular node [9].

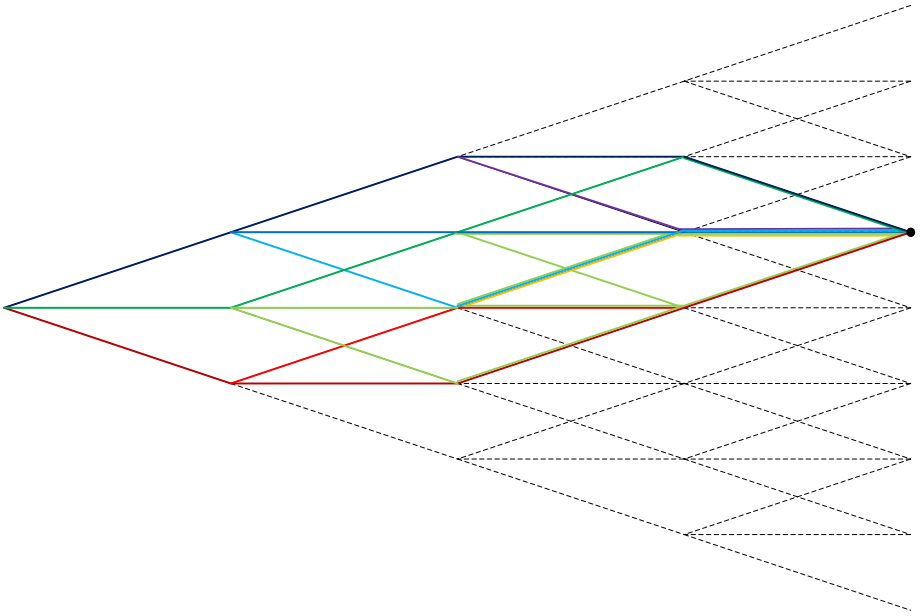

Figure 5.4: Different Paths to the same Node in a Trinomial Tree

To compute the price of a look-back option in a tree all the possible values of the maximum or minimum at each node must be considered. In general, the number of paths which reach a given node increases exponentially with the number of time steps to that node. Fortunately the number of maximum or minimum asset prices only increases linearly with the number of time steps, but this still increases significantly the amount of computation required.

The procedure is quite similar as for standard options. Step back through the tree in the usual way except that at every node the range - i.e. the minimum and maximum - of the possible maxima which can occur for every node in the tree must be determined first. This means, at every node the minimum and maximum possible maximum asset prices which could have occurred for all paths which reach the node are stored. Then choose an appropriate set of values of the maximum between the minimum and maximum possible for each node - the nodes which lie on the upper and lower edges of the tree have only one path passing through them, and therefore there will be only one maximum. The largest range of values will occur in the central section of the tree. Therefore the number of values considered should increase linearly with the number of time steps but also decrease linearly from the central nodes of the tree down to one at the edges of the tree.

Let $n_{i,j}$ be the number of values stored at node *(i,j)* and $F_{i,j,k}$, $k=1,\ldots, n_{i,j}$ be the values of the maximum, where $F_{i,j,1}$ is the minimum and $F_{i,j,\ n_{jj}}$ is the maximum. Figure 5.5 illustrates the structure of the nodes.

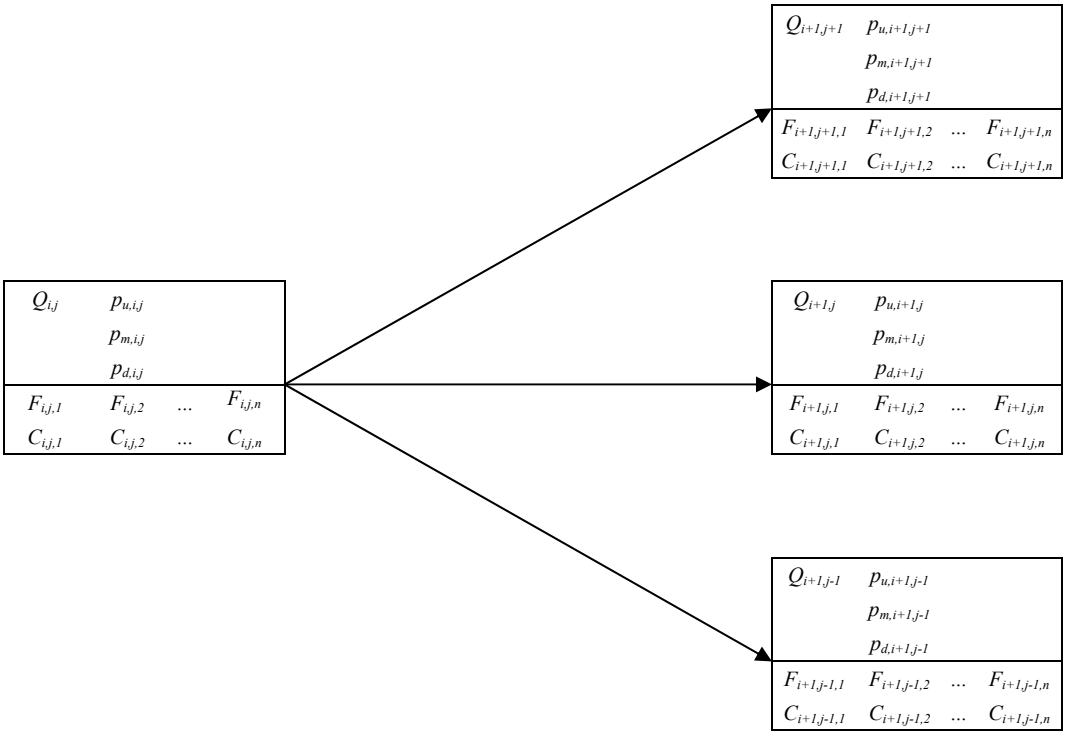

Figure 5.5: Structures of Nodes for the Valuation of a Path-Dependent Option

If $n_{i,j}$ is given by

$$n_{i,j} = 1 + \alpha(i - abs(j)) \tag{5.20}$$

where α is typically between one and five, $n_{i,j}$ will always be one at the edges of the tree *(j=i* and *j=-i)* and $1+\alpha i$ in the centre of the tree. Thus, α can be increased to improve the accuracy of the approximation by considering more values of the maximum, whilst keeping the computational effort required under control.

In order to find the range of values of the maximum step forward through the tree from *i=0* to *i=N*. Having found the range of maxima for all nodes up to time step *i-1,* then for any node *(i,j)* the minimum maximum must be the greater of the minimum maximum of the lowest node at time step *i-1* with a branch to the current node and the asset price at the current node:

$$F_{i,j,1} = \max(F_{i-1,j_i,1}, S_j) \tag{5.21}$$

where node $(i\text{-}1,j_i)$ is the lowest node with a branch to node *(i,j)*. Similarly, the maximum maximum must be the greater of the maximum maximum of the highest node at time step *i-1* with a branch to the current node and the asset price at the current node:

$$F_{i,j,n} = \max(F_{i-1,j_u,n}, S_j) \tag{5.22}$$

where node $(i\text{-}1,j_u)$ is the highest node with a branch to node (i,j).

A uniform spread for the set of $n_{i,j}$ values of the maximum over the range found at each node (i,j) is given by

$$F_{i,j,k} = F_{i,j,1} + \left(\frac{F_{i,j,n} - F_{i,j,1}}{n-1}\right)(k-1) \tag{5.23}$$

The value of the option at maturity at every node and for every value of the maximum can be set, once all the values of the maximum at every node have been computed.

$$C_{N,j,k} = \max(0, F_{i,j,k} - K), j = -N,..., N, k = 1,..., n \tag{5.24}$$

Finally, again step back through the tree computing discounted expectations and applying the early exercise condition at every node and for every value of the maximum

$$C_{i,j,k} = e^{-r\Delta t_i}(p_{u,i,j}C_{i+1,j+1,u} + p_{m,i,j}C_{i+1,j,m} + p_{d,i,j}C_{i+1,j-1,d}) \tag{5.25}$$

where $C_{i+1,j+1,u}$, $C_{i+1,j,m}$, $C_{i+1,j\text{-}1,d}$, are the values of the option at time step $i+1$, given the current maximum, for upward, middle and downward branches of the asset.

For the middle and downward branches the maximum will remain the same, it cannot be changed by the asset price decreasing.

$$F_{i+1,j,m} = F_{i,j,k} \quad F_{i+1,j-1,d} = F_{i,j,k} \tag{5.26}$$

For the upward branch the maximum is the greater of the current maximum and the asset price at the upward branch node

$$F_{i+1,j+1,u} = \max(F_{i,j,k}, S_{j+1}) \tag{5.27}$$

The maxima $F_{i+1,j+1,u}$ $F_{i+1,j,m}$ $F_{i+1,j\text{-}1,d}$ and therefore also the option values $C_{i+1,j+1,u}$, $C_{i+1,j,m}$ $C_{i+1,j\text{-}1,d}$ will not, in general, be stored at the upward, middle and downward

nodes and therefore must be obtained by interpolation. For example using linear interpolation having

$$C_{i+1,j+1,u} = C_{i+1,j+1,k_i} + \left(\frac{C_{i+1,j+1,k_u} - C_{i+1,j+1,k_i}}{F_{i+1,j+1,k_u} - F_{i+1,j+1,k_i}} \right) (F_{i+1,j+1,u} - F_{i+1,j+1,k_i}) \qquad (5.28)$$

where k_i and k_u are such that

$$F_{i+1,j+1,k_i} \le F_{i+1,j+1,u} \le F_{i+1,j+1,k_u} \quad \text{and} \quad k_u = k_i + 1.$$

That is, the two maxima which lie closest to either side of $F_{i+1,j+1,u}$ are found and a linear interpolation between these is done to obtain an estimate for $C_{i+1,j+1,u}$ and similarly for $C_{i+1,j,m}$ and $C_{i+1,j-1,d}$. This will always be possible because at every node the minimum and maximum possible values of the maximum are stored.

6 Web Services

6.1 Web Services Introduction

Web services are expected to revolutionize our life in the same way as the Internet has during the past decade or so. The key is that Web services provide a common protocol that Web applications can use to connect to each other over the Internet.

As a result of Web services the integration of applications is more easily and quickly than ever before. This integration takes places at a higher level in the protocol stack and is based on messages. The core issue of these messages is the emphasis of service semantics and less network protocol semantics which enables the possibility of loose integration of functions. These are ideal characteristics to connect business functions across the Web either between enterprises as well as within enterprises.

Web services are a technology for deploying and providing access to business functions over the Internet. There are several development platforms, tools, and kits to help building Web services [16].

6.2 Overview of Java Web Services

From a software architect's point of view, a Web service can be considered as a service-oriented architecture, which consists of a collection of services that communicate with each other (and end-user clients) through well-defined interfaces. One advantage of service-oriented architecture is that it allows the development of loosely coupled applications that can be distributed and accessed, from any client, across the network [17].

6.3 Web Services Definition

A Web service is **'a software system designed to support interoperable machine-to-machine interaction over a network'** (W3C) [18].

A Web service is **'an interface that describes a collection of operations that are network-accessible through standardized XML messaging'** (IBM) [19].

'Web Services are self-describing components that can discover and engage other Web services or applications to complete complex tasks over the Internet' (SUN) [20].

The main advantage of Web services is that they are built on existing industry standards. Web services are application components that are designed to support interoperable machine-to-machine interaction over a network. This interoperability is gained through a set of XML-based open standards, such as the Simple Object Access Protocol (SOAP), the Web Services Description Language (WSDL), and Universal Description, Discovery, and Integration (UDDI). These standards provide a common and interoperable approach for defining, publishing, and using Web services.

For example, the services are described in Extensible Markup Language (XML) and are communicated over the Hypertext Transfer Protocol (HTTP). This union is one way to form the new industry standard called Simple Object Access Protocol (SOAP). Publication of Web services is done via two standards, the Universal Description, Discovery, and Integration (UDDI), and Discovery (DISCO).

6.4 Web Services Properties

The Web service interface hides the complexity of the service implementation.

This fact provides the following advantages for the service:

- Independent usage (hardware/software platform, programming language)
- Loosely coupled
- Component oriented
- All terrain implementations

The Web service description takes place through standardized formal XML messaging.

The content of the Web service description covers the following information details:

- Interaction with the service.
- Message formats which detail the operation.
- Transport protocols.
- Location of the service.

7 Web Services Model

The Web service model is based on *interactions* between three **operators** - *service provider*, *service requestor* and *service registry*. The **interactions** include the, *publish*, *find* and *bind* operations. Together, the interactions and operators handle the Web services **artifacts** - the *Web service software module* and the *Web service description* (see Figure 7.1) [21].

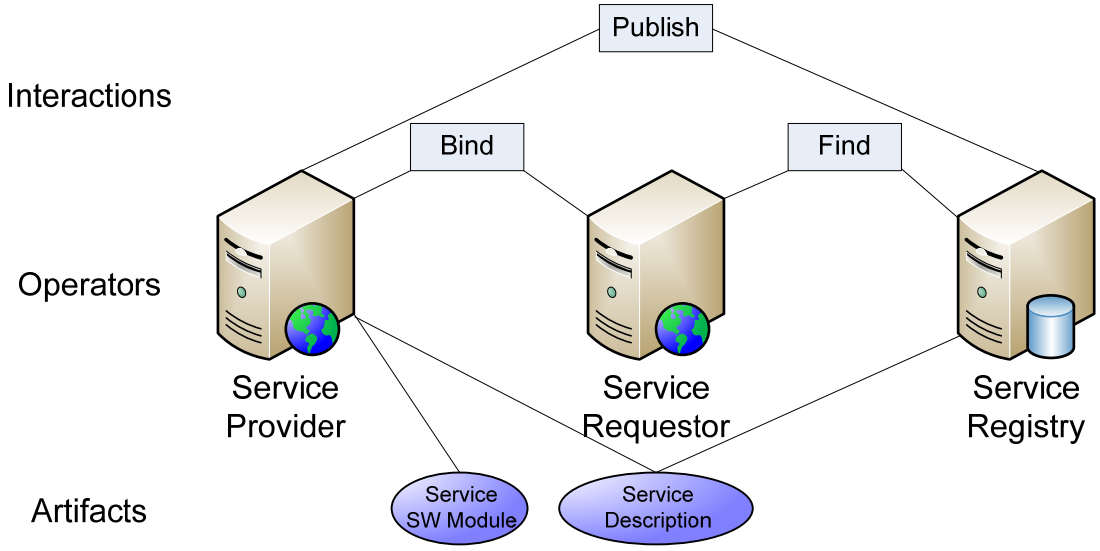

Figure 7.1: Interactions, Operators and Artifacts

A typical scenario would consist of the following steps:

1. The service provider hosts the service software module (implementation of a Web service).
2. The service provider defines a service description for the Web service.
3. The service provider publishes the service description to the service requestor and/or service registry.
4. The service requestor uses a find operation to retrieve the service description.
5. The service requestor uses the service description to bind with the service provider.
6. Finally, the service requestor invokes or interacts with the Web service implementation.

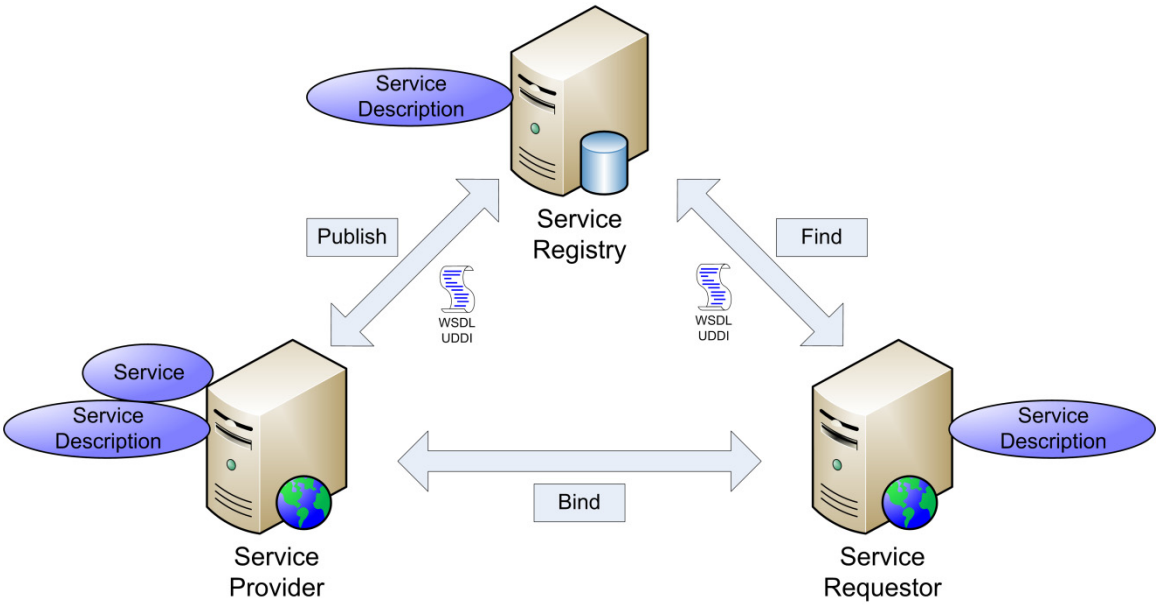

Figure 7.2: Web Services Model

7.1 Operators of the Web Services Model

Following operators are involved in the Web service model:

- **Service provider** is the owner of the service and the author of the Web service description. This operator (platform) hosts access to the service.
- **Service requestor** is any kind of business with certain functions which want to be satisfied or an application that is invoking or initiating an interaction with the service. So the service requestor can be anyone using a browser or a program without a user interface, for example another Web service.
- **Service registry** is a searchable data base where service providers publish their service descriptions. Service requestors can find services and obtain binding information from the service descriptions. Also other sources besides a service registry, such as a local file, FTP site, Web site or Discovery of Web services (DISCO) can obtain a service description. This means that the service registry is an optional operator in the Web service model (e.g. if the service requestor has the service description there is no need for the service registry).

7.2 Interactions of the Web Services Model

The following three elementary interactions must take place for an application to consume the Web service:

1. **Publish** the service description(s).
2. **Find** the service description(s).
3. **Bind** or invoke the service(s) based on the service description(s).

In detail, these interactions are:

- **Publish**. A service description is only accessible, when it is published so that the service requestor can find it. The location where the service description is published depends on the requirements of the application.
- **Find**. In the find operation, the service requestor retrieves a service description directly from the service provider or queries the service registry (see below) for the type of service required. This operation can occur in two different lifecycle phases - at design time to retrieve the service's interface description for program development, and at runtime to retrieve the service's binding and location description for invocation.
- **Bind**. In the bind operation the service requestor invokes or initiates an interaction with the service at runtime. Therefore it uses the binding details in the service description to localize, contact and invoke the service.

7.3 Artifacts of the Web Services Model

The Web service model contains the following artifacts:

- **Service**. A service is a software module deployed on a network-accessible platform to be invoked by or to interact with a service requestor. It is also possible that the service acts as a requestor, by referring to other Web services in its implementation.
- **Service Description**. The service description contains the details of the interface and implementation of the service, including data types, operations, binding information and network location. The service description is published either to a service requestor or to a service registry.

8 Web Services Architecture

This section handles the role of service description in the Web services architecture and service publication techniques supporting static as well as dynamic[12] Web services applications. Regarding to service publication, the mechanism of service discovery is shortly discussed as well [22].

8.1 Web Services Protocol Stack

To review the Web services architecture at first, a detailed look at a conceptual stack for Web services is taken. Included are the layers for choosing the network protocol, XML-based messaging and extended basic XML messaging with a service description.

In order to perform the three Web services operations publish, find and bind a Web services stack must incorporate standards at each level. Figure 8.1 shows a conceptual Web services protocol stack with the standards that apply at the corresponding layer. The lower layers provide capabilities that upper layers build on, whereas the vertical bars represent enterprise-class infrastructure requirements that must be fulfilled [19].

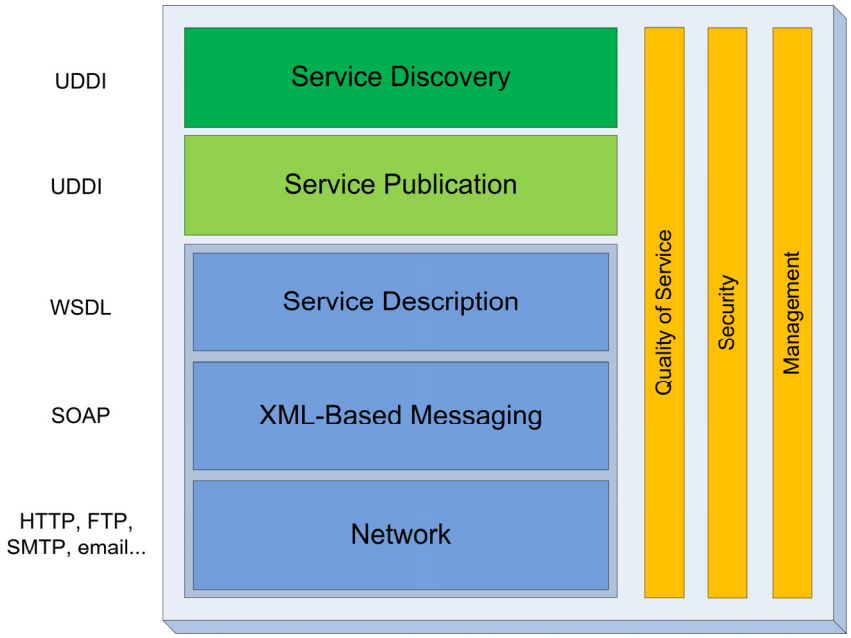

Figure 8.1: Web Services Protocol Stack

8.2 Network Layer

The base layer of the Web services stack is the network layer. Web services that are publicly available on the Internet are generally described in XML and are communicated over the existing HTTP infrastructure. Because of its omnipresence, HTTP can be seen as the quasi standard network protocol but also supported are the

[12] Depending on the Web services lifecycle when binding takes place - before runtime (static) or during runtime (dynamic).

Simple Mail Transfer Protocol (SMTP), File Transfer Protocol (FTP), e-mail, and so on.

8.3 XML-Based Messaging Layer - SOAP

SOAP once stood for 'Simple Object Access Protocol' but was considered to be misleading and therefore this acronym was dropped with Version 1.2. SOAP is a simple XML-based protocol to exchange structured data between network applications, normally using HTTP. SOAP therefore is the standard enveloping and exchanging transport mechanism embedding document-centric messages and remote procedure calls (RPC's) using XML [23].

This protocol is chosen as the XML messaging protocol for several reasons:

- SOAP is simple and extensible - an HTTP message within an XML envelope.
- SOAP messages support publishing, finding and binding Web services operations.
- SOAP embraces message extensions like headers and standard coding mechanism of operations or functions, to satisfy compliance with standards at every level.
- SOAP can be used in combination with a variety of network protocols such as HTTP, SMTP, FTP (see above).

8.3.1 SOAP message structure

The structure of a SOAP message with/out attachment can be seen in Figure 8.2.

SOAP consists of three parts:

- The envelope that defines a framework, describing what is in a message.
- A set of coding rules to express the instances of application-specific data types.
- Conventions to represent remote procedure calls and responses.

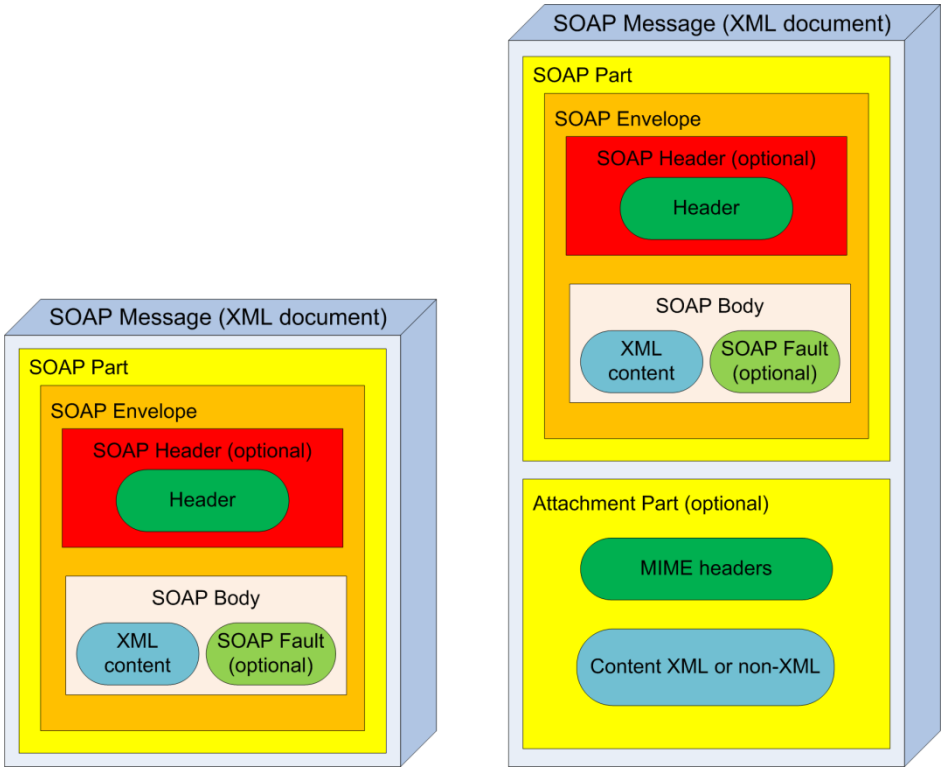

Figure 8.2: SOAP Message Structure with/out Attachment

8.3.2 SOAP message example

In this example, a `GetStockPrice` request is sent to a server. The request has a `StockName` parameter, and a `Price` parameter will be returned in the response. The namespace for the function is defined in 'http://www.example.org/stock' address [24].

SOAP request message:

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <soap:Body xmlns:m="http://www.example.org/stock">
    <m:GetStockPrice>
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>

</soap:Envelope>
```

SOAP response message:

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
```

```
  <soap:Body xmlns:m="http://www.example.org/stock">
    <m:GetStockPriceResponse>
      <m:Price>34.5</m:Price>
    </m:GetStockPriceResponse>
  </soap:Body>

</soap:Envelope>
```

8.3.3 XML Based Messaging using SOAP

The main requirements for a network node to operate as service requestor or provider are the capability to communicate over an accessible network and the capability to build and/or parse SOAP messages. Usually, a Web application server running SOAP performs these functions. As an alternative, a programming language-specific runtime library can be used that encapsulates these functions within an API. Figure 8.3 shows how SOAP (XML-based messaging) and network protocols (HTTP, FTP,...) builds the base of the Web services architecture [25].

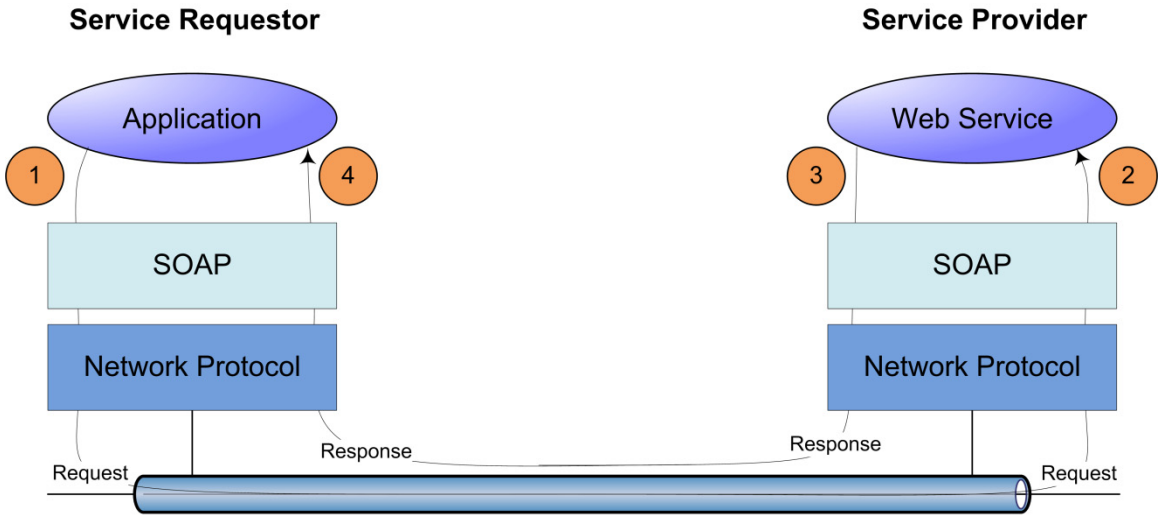

Figure 8.3: XML Based Messaging using SOAP

Typical scenario – Application integration using SOAP:

1. The service requestor's application creates a SOAP message. Together with the network address of the service provider, the service requestor passes this message to the local SOAP infrastructure (e.g., the SOAP client). The SOAP client runtime uses an underlying network protocol (e.g. HTTP) to transmit the SOAP message over the network.
2. The SOAP message is delivered to the service provider's SOAP runtime (e.g. SOAP server). The SOAP server converts the XML message into programming language-specific objects if required by the application and routes the request to the service provider's Web service.
3. The Web service processes the request message and formulates a response, of course also a SOAP message. The response is passed to the local SOAP

runtime specifying the service requestor as its destination, where the SOAP message response is sent to the service requestor.

4. Finally the response message is received by the networking infrastructure on the service requestor's node, where it is routed through the SOAP infrastructure. Optionally the XML message is converted into objects of the target programming language. The response message is then passed to the application.

Neither the requestor nor the provider must be aware of the other's underlying platform, programming language, or distributed object model (if any). The service description combined with the underlying SOAP infrastructure hides these details apart from the service requestor's application and the service provider's Web service.

8.4 Service Description Layer

8.4.1 From XML Messaging to Web Services

A stack of description documents defines the service description layer. This stack is the minimum standard service description necessary to support interoperable Web services. The Web services architecture uses the Web Services Description Language (WSDL) standard for XML-based service description.

WSDL defines the interface and mechanics of service interaction. WSDL is an XML document for describing Web services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented (RPC) content. The operations and messages are described abstractly, and then bound both to a concrete network protocol and message format in order to define an endpoint. WSDL is extensible to allow the description of endpoints and messages, regardless of message formats and network protocols used in the communication.

Additional description is necessary to specify high level aspects of the Web service. The WSDL document can be complemented in order to describe the business context, quality of service (QOS) and service-to-service relationships. For example, the business context is described using UDDI data structures in addition to the WSDL document [22].

Because a Web service is defined as being network-accessible via the Web service stack and represented by a service description, the lower three layers are required to provide or use any Web service (see Figure 8.1).

The simplest stack consists of:

1. HTTP for the network layer,
2. SOAP for the XML messaging layer and
3. WSDL for the service description layer.

This is the interoperable base stack that all inter-enterprise, or public, Web services should support.

8.4.2 Basic Web Service Description

Using WSDL in the Web services architecture divides the basic service description into two parts - the service interface and the service implementation. Its advantage is that each part can be defined separately and independently, and reused by other parts.

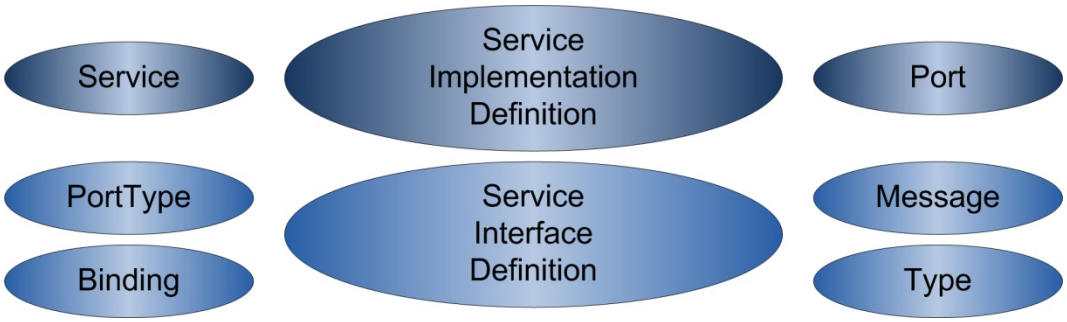

Figure 8.4: Basic Web Service Description

A *service interface definition* is a reusable service definition that can be referenced and instantiated by multiple service implementation definitions. The service interface contains WSDL elements that embed the reusable fragment of the service description (Figure 8.4):

- The *WSDL:types* elements describes the use of complex data types within the message.
- The *WSDL:message* element is used to define the input and output parameters of an operation. It specifies which XML data types determine diverse parts of a message.
- The *WSDL:portType* element defines the operations of the Web service. Like an operation as a method signature in a programming language it defines the XML messages that can appear in the input and output data flows.
- The *WSDL:binding* element describes protocol, data format, security and other attributes of a specific service interface.

The WSDL document that describes how a particular service interface is implemented by a service provider is the *service implementation definition*. A Web service is modeled as a *WSDL:service* element, which contains a collection of *WSDL:port* elements (usually one). A port associates an endpoint, for instance a network address location or URL, with a *WSDL:binding* element from a service interface definition (Figure 8.4).

Together this pair makes up a basic WSDL definition of the service containing sufficient information to describe to the service requestor how to invoke and interact with the Web service. Other information about the service provider's endpoint is provided by the complete Web service description of the service [26].

8.4.3 Full WSDL Syntax

The full WSDL 1.2 syntax as described in the W3C working draft is listed below [27].

```
<wsdl:definitions name="nmtoken"? targetNamespace="uri">
    <import namespace="uri" location="uri"/> *
    <wsdl:documentation .... /> ?
    <wsdl:types> ?
        <wsdl:documentation .... /> ?
        <xsd:schema .... /> *
    </wsdl:types>
    <wsdl:message name="ncname"> *
        <wsdl:documentation .... /> ?
        <part name="ncname" element="qname"? type="qname"?/> *
    </wsdl:message>
    <wsdl:portType name="ncname"> *
        <wsdl:documentation .... /> ?
        <wsdl:operation name="ncname"> *
            <wsdl:documentation .... /> ?
            <wsdl:input message="qname"> ?
                <wsdl:documentation .... /> ?
            </wsdl:input>
            <wsdl:output message="qname"> ?
                <wsdl:documentation .... /> ?
            </wsdl:output>
            <wsdl:fault name="ncname" message="qname"> *
                <wsdl:documentation .... /> ?
            </wsdl:fault>
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:serviceType name="ncname"> *
        <wsdl:portType name="qname"/> +
    </wsdl:serviceType>
    <wsdl:binding name="ncname" type="qname"> *
        <wsdl:documentation .... /> ?
        <-- binding details --> *
        <wsdl:operation name="ncname"> *
            <wsdl:documentation .... /> ?
            <-- binding details --> *
            <wsdl:input> ?
                <wsdl:documentation .... /> ?
                <-- binding details -->
            </wsdl:input>
            <wsdl:output> ?
                <wsdl:documentation .... /> ?
                <-- binding details --> *
            </wsdl:output>
            <wsdl:fault name="ncname"> *
                <wsdl:documentation .... /> ?
                <-- binding details --> *
            </wsdl:fault>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="ncname" serviceType="qname"> *
        <wsdl:documentation .... /> ?
        <wsdl:port name="ncname" binding="qname"> *
            <wsdl:documentation .... /> ?
            <-- address details -->
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

A concrete example for a Web service description is given in chapter 12.3.

8.4.4 Complete Web Service Description

The complete Web service description answers questions about:

- The business and type of business hosting the service.
- The products associated with the service.
- The associated categories in various company and product systems.
- The provided keywords so that it is easier to find the service.
- Other aspects of the service such as Quality of Service or Security.

Finally, the top layer in the service description stack is the *agreement description* using UDDI. An agreement description represents a simple coordination of Web service invocations between two business partners to complete a multi-step business interaction.

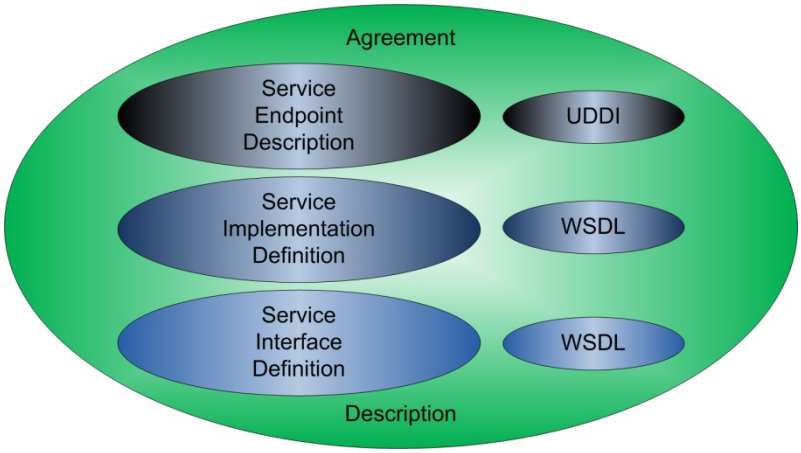

Figure 8.5: Complete Web Service Description

The *service endpoint description* adds further semantics to the service description that apply to a particular implementation of the service. Security attributes can define the access policy to the Web service. Quality of Service attributes will specify performance-oriented service capabilities, for example, to respond within a certain period of time. UDDI (Universal Description, Discovery, and Integration) therefore provides a mechanism for holding descriptions of Web services which is not covered more detailed.

As an example, the coordination of roles such as buyer and seller within a purchasing protocol which outlines the requirements that each role must fulfill. For example, the seller must have Web services that receive request for quote (RFQ) messages, purchase order (PO) messages and payment messages. The buyer role must have Web services that receive quotes (RFQ response messages), invoice messages and account summary messages. This simple coordination of Web services into business roles is essential for establishing multistep, service-oriented interactions between business partners.

8.5 Publication and Discovery of Service Descriptions

Any action that makes a WSDL document available to a service requestor qualifies as *service publication.*

A service description can be published using a variety of mechanisms. The simplest scenario is the service provider sending a WSDL document directly to a service requestor what is called direct publication. Direct publication, ideally via e-mail, is useful for statically bound applications. Alternatively, the service provider can publish the WSDL document describing the service to a private UDDI registry or UDDI operator node.

Any mechanism that allows the service requestor to gain access to the service description and make it available to the application at runtime qualifies as *service discovery*.

The simplest scenario of discovery is static discovery where the service requestor retrieves a WSDL document from a local file. This is usually the WSDL document obtained through a direct publish or a previous find operation.

9 Web Services Development Lifecycle

A typical end-to-end lifecycle scenario would start with the creation and publication of a service interface (*build*), proceed to the creation and deployment of the Web service (*deploy*), move on to the publication of the service implementation definition and end with the invocation of the Web service by the service requestor (*run*).

The development lifecycle includes the following phases:

- *build*
- *deploy*
- *run*
- *manage*

Each Web service operator – service registry, service provider and service requestor – has specific requirements for each element of the development lifecycle.

9.1 Build Phase

The build lifecycle phase involves development and testing of the Web services implementation. Further it includes the definition of the service interface description and the definition of the service implementation description. Locating an existing service interface definition is also a build-time task.

The Web services implementations can be provided by creating new Web services, transforming existing applications into Web services, or composing new Web services from other Web services and applications.

There are some similarities between a Web service development approach and object-oriented programming. Both use concepts such as encapsulation, interface inheritance and dynamic binding. This means that object-oriented design methodologies can be applied to Web services design, but it is not required to design a Web service.

9.2 Deploy Phase

The tasks of the deploy phase of the development lifecycle include the publication of the service interface and service implementation definition, deployment of the runtime code for the Web service as well as integration with back-end legacy systems.

9.3 Run Phase

During the run lifecycle phase, the Web service is fully deployed and operational. In this state, a service requestor can find the service definition and invoke all defined service operations. The runtime functions include static and dynamic binding, service interactions as a function of Simple Object Access Protocol (SOAP) messaging and interactions with legacy systems.

9.4 Manage Phase

The manage lifecycle phase covers continual management and administration of the Web service application. Security, availability, performance, quality of service and

business processes must all be addressed. This document focuses on the development of Web services and does not cover this phase of the lifecycle.

10 Developing Web Services

This section describes the Web service lifecycle for each operator: service registry, service provider and service requestor.

10.1 Service Registry

Development and deployment of a service registry is not covered, because it is a passive participant. It is assumed that the registry has been built and deployed before it is selected for use by the service provider or service requestor.

10.2 Service Provider

The service provider in this context is software. To develop a Web service there exist four basic scenarios to implement a service provider. Which scenario is used for the implementation is based on the creation of a new service interface and application. Table 10.1 provides an overview of these development scenarios [28].

	New Service Interface	Existing Service Interface
New Web Service	**Green field**	**Top-down**
Existing Application	**Bottom-up**	**Meet-in-the-middle**

Table 10.1: Basic Scenarios for Service Provider Implementation

10.2.1 Green Field Scenario

The green field scenario for developing Web services describes how a new service interface will be created for a new Web service, as shown in Table 10.1. The Web service is created first and then the service interface definition is generated, so both are owned by the service provider.

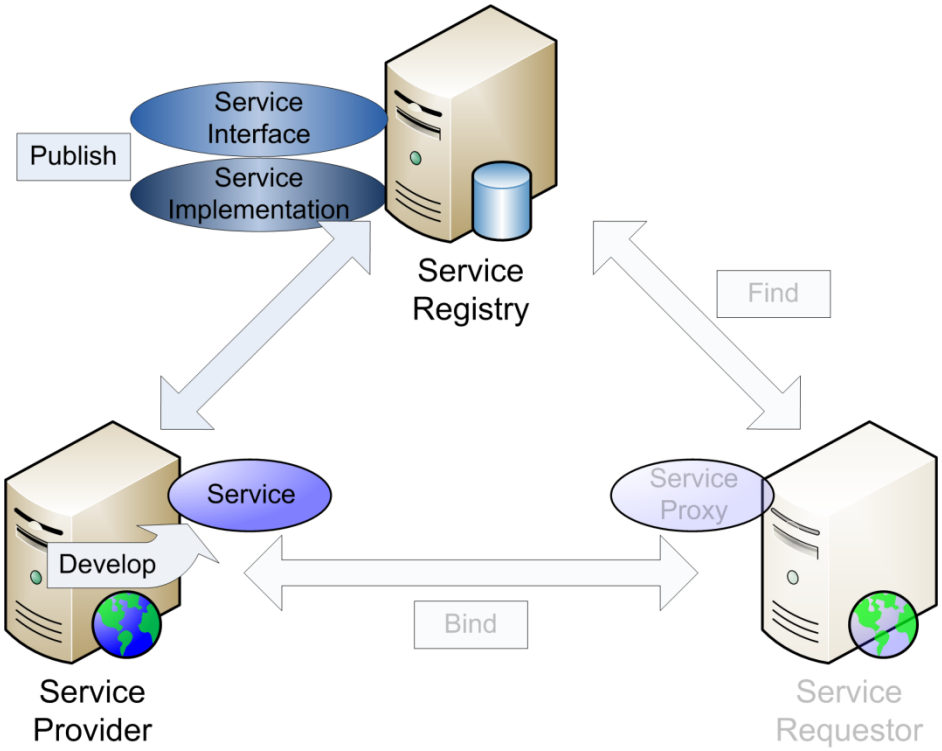

Figure 10.1: Green Field Scenario

10.2.1.1 Build Phase

1. Develop the new Web service.

   Design and implement the application that represents the Web service, and verify that all of its interfaces work correctly.

2. Define a new service interface.

   The next step is to generate the service interface definition from the implementation of the service. The service interface should not be generated until the Web service development is complete because the interface must match the exact implementation of the service.

10.2.1.2 Deploy Phase

1. Publish the service interface definition before the service is deployed.

   The service interface definition is used by a service requestor to determine how to bind to the service.

2. Deploy the Web service.

   Deploy the runtime code and any deployment meta data (e.g. the deployment descriptor to deploy a SOAP service) that is required to run the service. After a service has been deployed, it is ready to be used by a service requestor.

3. Create the service implementation definition.

   Depending on how and where the service was deployed the service implementation definition should be created, because it can contain references

to more than one version of the deployed Web service. This allows the service provider to implement different levels of service for service requestors.

4. Publish the service implementation definition.

   After the service implementation definition is published, a service requestor can find the service definition and use it to bind to the Web service. Therefore the service implementation definition contains the definition of the network-accessible endpoint or endpoints.

10.2.1.3 Run Phase

Run the Web service.

The runtime environment for the Web service consists of the platform on which it was deployed to run. If the Web service is a servlet, then it runs in the context of a Web application server. If the Web service is a SOAP service, then it runs in the context of a SOAP server.

10.2.2 Top-Down Scenario

The top-down scenario is where a new Web service can be developed matching to an existing service interface, see also Table 10.1. Figure 10.2 shows, that the service provider must find the service interface, implement the interface contained in this definition, and then deploy the new Web service. Only the service implementation is owned by the service provider.

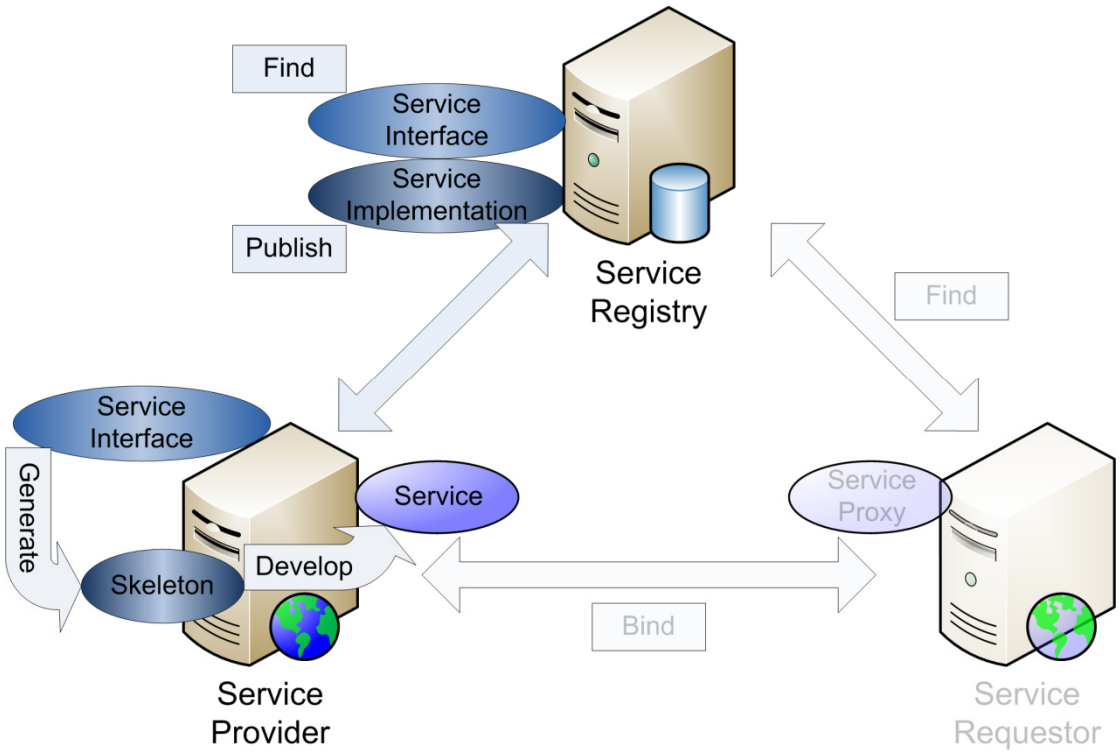

Figure 10.2: Top-Down Scenario

10.2.2.1 Build Phase

1. Find the service interface.

   Locate the service interface that will be implemented by the Web service by searching the service registry or an industry specification registry.

2. Generate the service implementation template.

   An implementation template of the Web service is generated by using the service interface definition. The template contains all of the methods and parameters that must be implemented by the Web service to comply with the service interface.

3. Develop the new Web service.

10.2.2.2 Deploy Phase

The only difference here, compared to the green field scenario, is that the service interface has already been published by another operator.

1. Deploy the Web service.
2. Create the service implementation definition.
3. Publish the service implementation definition.

10.2.2.3 Run Phase

Run the Web service.

The runtime environment for the Web service consists of the platform on which it was deployed to run.

10.2.3 Bottom-Up Scenario

As shown in Table 10.1 the bottom-up scenario creates a new service interface for an existing application. The application can be implemented as an Enterprise Java Bean™ (EJB), Java Bean, servlet, C++ or Java class file, or Component Object Model (COM) class. The service interface is derived from the application's API, as Figure 10.3 shows.

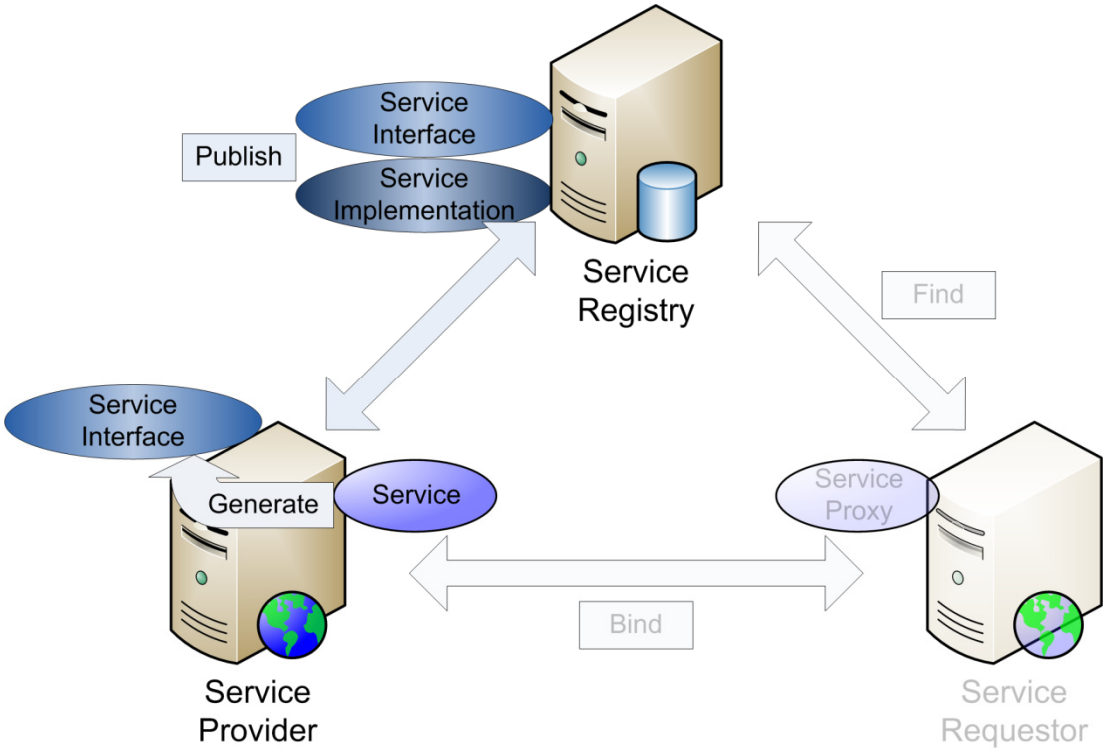

Figure 10.3: Bottom-Up Scenario

10.2.3.1 Build Phase

Generate the service interface.

The service interface is generated from the implementation of the application that represents the Web service.

10.2.3.2 Deploy Phase

1. Deploy the Web service.
2. Create the service implementation definition.
3. Publish the service interface definition.

   The service interface definition must be published before the service implementation definition can be published.
4. Publish the service implementation definition.

10.2.3.3 Run Phase

Run the Web service.

The runtime environment for the Web service consists of the platform on which it was deployed to run.

10.2.4 Meet-in-the-Middle Scenario

When a service interface and an application implementing the Web service already exist the meet-in-the-middle scenario is used, as shown in Table 10.1.

The main task here is to map the existing application interfaces to those defined in the service interface definition. This can be done by creating a wrapper for the application that uses the service interface definition. The wrapper contains an implementation that maps the service interface into the existing application interface. Figure 10.4 shows the mapping process.

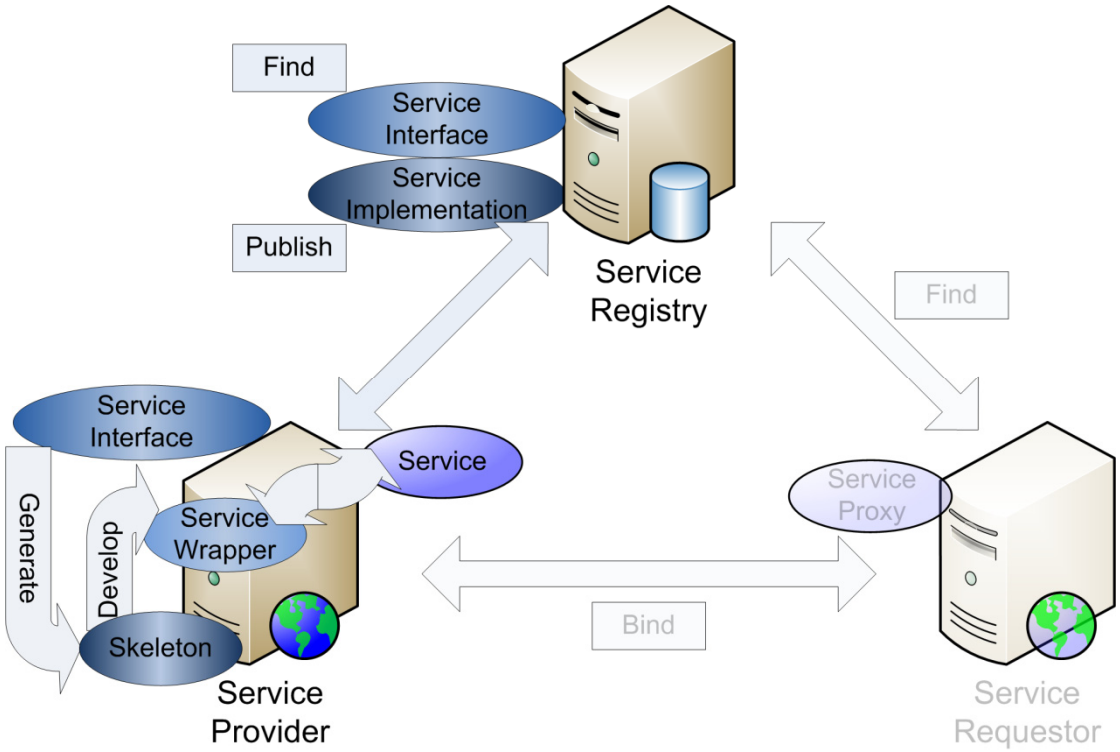

Figure 10.4: Meet-in-the-Middle Scenario

10.2.4.1 Build Phase

The first two build steps are similar as those for the top-down scenario.

1. Find the service interface.

   Locate the service interface that will be implemented by the Web service by searching the service registry or an industry specification registry.

2. Generate the service implementation template.

3. Develop the service wrapper.

   The service wrapper is designed and implemented by using the service implementation template created in the previous step.

10.2.4.2 Deploy Phase

The deployment steps for the meet-in-the-middle scenario are similar to those of the bottom-up scenario the only difference is that the service interface definition is already published.

1. Deploy the Web service.

2. Create the service implementation definition.

3. Publish the service implementation definition.

10.2.4.3 Run Phase

Run the Web service.

The runtime environment for the Web service consists of the platform on which it was deployed to run.

10.3 Service Requestor

The service requestor passes through the same lifecycle as the service provider, but the requestor performs different tasks during each phase. The build time tasks for the service requestor are dictated based on the method for binding to a Web service.

From the service interface a service proxy or stub is generated which contains all of the code that is required to access and invoke a Web service. For example, if the Web service is a SOAP service, the service proxy will contain all of the SOAP client code that is required to invoke a method on the SOAP service.

As Table 10.2 shows, there are three methods for binding to a specific service. S*tatic binding* is used only at build time, whereas *dynamic binding* can be used either at build time or runtime. Static binding cannot be used at runtime, because it requires all of the information needed to bind to a service at build time [28].

	Static Binding	Dynamic Binding
Build	**Static binding**	**Build-time dynamic binding**
Run	**[not applicable]**	**Runtime dynamic binding**

Table 10.2: Methods for Service Requestor Binding

10.3.1 Static Binding

When there is only one service implementation that will be used at runtime a service requestor will use static binding (see Figure 10.5). The static binding is done at build time by locating the service implementation definition for the single Web service. The service implementation definition contains a reference to the service interface, which will be used to generate the service proxy code. The service proxy contains a complete implementation of the client application that can be used by the service requestor to invoke Web service operations.

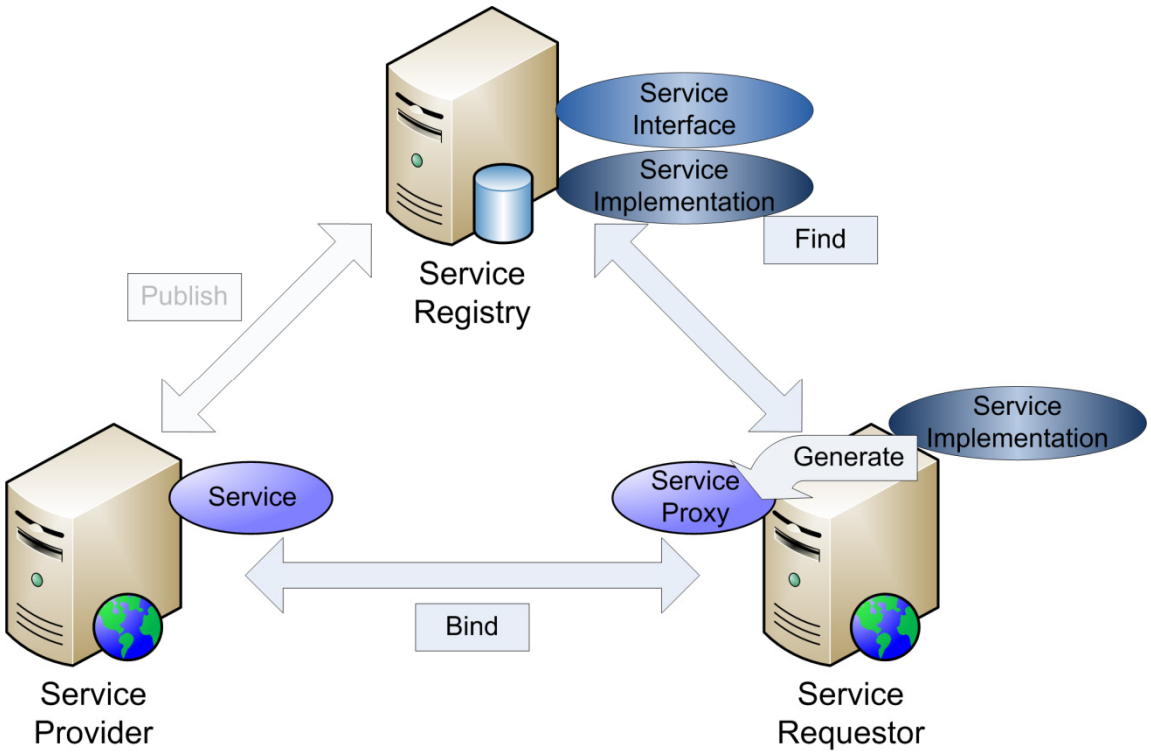

Figure 10.5: Static Binding

10.3.1.1 Build Phase

1. Find the service implementation definition.

   At build time, the service requestor must find the service implementation definition for the Web service which contains a reference to the service interface definition, and the location where the service can be accessed.

2. Generate the service proxy.

   Both, the service interface definition and the service location information are used to generate the service proxy implementation. The service proxy will try to access the Web service always at the same location and will match with the service interface.

3. Test the service proxy.

   To verify that the service proxy can interact correctly with the specified Web service, it should be tested.

10.3.1.2 Deploy Phase

Deploy the service proxy.

After testing, it is deployed with the client application in the client runtime environment.

10.3.1.3 Run Phase

Invoke the Web service.

Run the requestor application which will invoke the Web service via the service proxy.

10.3.2 Build-Time Dynamic Binding

This binding method is used when a service requestor wants to use a specific type of Web service, but the implementation is not known until runtime or it can change at runtime. The type of service is defined in a service interface definition.

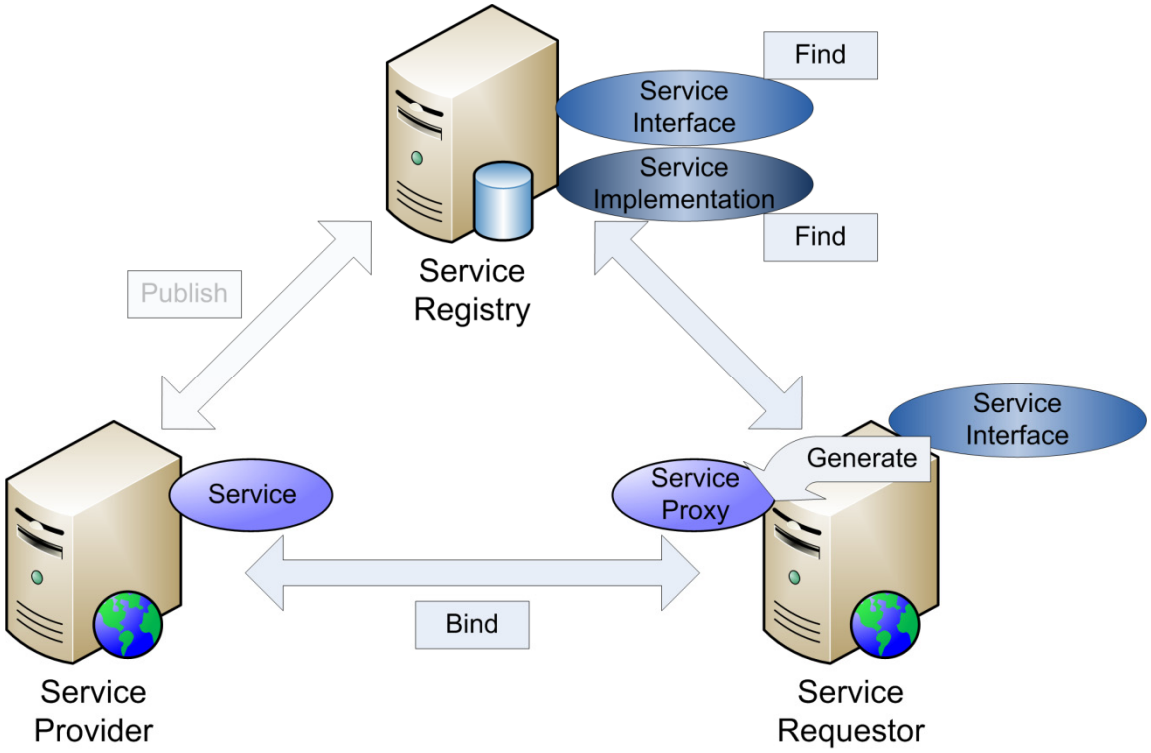

Figure 10.6: Build-Time Dynamic Binding

10.3.2.1 Build Phase

1. Find the service interface definition.

   First the service interface definition for the type of service that will be used by the service requestor must be found. The service interface contains only the abstract definition of the Web service operations.

2. Generate the generic service proxy.

   The service interface definition is used to generate a generic service proxy which can be used to access any implementation of the service interface. Unlike the service proxy generated for static binding, this proxy will not contain knowledge of a specific service implementation. So the generic service proxy will contain code to locate a service implementation by searching a service registry.

3. Test the service proxy.

Just find an implementation of the service interface for testing.

10.3.2.2 Deploy Phase

Deploy the service proxy.

If the service proxy passed testing and works correctly it should be deployed within the runtime environment. This process can also include the deployment of the requestor application that will use the service proxy. The application must have access to the service registry that will be searched for an implementation of the service interface.

10.3.2.3 Run Phase

1. Find the Service implementation definition.

   An implementation of the service must be located in the service registry before the service proxy can invoke a service.

2. Invoke the Web service.

   After a service implementation has been found, the service proxy can be used to invoke the Web service.

10.3.3 Runtime Dynamic Binding

Runtime dynamic binding is similar to build-time dynamic binding the only difference is that the service interface is found at runtime. A service interface is used to generate a general service proxy interface that can be used to invoke any implementation of the service interface. Generally this binding method would be used with a user interface, because machine-to-machine interactions cannot be absolutely dynamic.

10.3.3.1 Build Phase

Build the service requestor application.

The service requestor application is built using a dynamic binding runtime interface. This interface is used to find a service implementation, and then retrieve the service interface associated with the service implementation.

10.3.3.2 Deploy Phase

Deploy the service requestor application.

The service requestor application is deployed so that it will run and use the Web service runtime environment.

10.3.3.3 Run Phase

1. Find the service implementation definition.

   To find a service implementation definition the service requestor application uses runtime environment. Different methods can be used to locate a service implementation in a service registry. It can be found by first locating a business or type of business, and then determining the services offered by those businesses. The service implementation could also be located by

searching for a classification of service, or by first locating a type of service (or service interface). If the service interface is target of a search operation, then it is used to locate the implementations of service interface.

2. Generate and deploy the service proxy.

   The service proxy code that will be used to invoke the service is generated using the service interface associated with the service implementation. After the code generation, it is compiled and made available in the runtime environment.

3. Invoke the Web service.

   The generated service proxy code is used to invoke the Web service.

11 Web Services and Java Technology

11.1 Web Service Tools - Java 2 Platform

A set of several developing platforms, tools, and kits can be used to help build these scenarios. Development tools automate various aspects of Web service development simplifying design, deployment and integration [17] and [29].

The Java 2 Platform, Enterprise Edition (J2EE) version 1.4 provides comprehensive support for Web services. Existing J2EE components can be easily exposed as Web services.

The following implementations use the tools provided with the J2EE environment for several reasons:

- Interoperability

  Web services are integrated through the JAX-RPC 1.1 API, which can be used to develop service endpoints based on SOAP. JAX-RPC 1.1 provides interoperability with Web services based on the Web Services Description Language (WSDL) and Simple Object Access Protocol (SOAP).

- Portability

  J2EE 1.4 supports the WS-I Basic Profile to ensure that Web services are portable not only across J2EE implementations, but are also interoperable with any Web service developed, using any platform that conforms to the WS-I standards.

- Scalability

  J2EE containers provide transaction support, database connections, life cycle management, and other services that are scalable and require no code from application developers.

- Reliability
- No single-vendor lock-in

11.2 J2EE 1.4 SDK

The J2EE 1.4 SDK gives access to several tools, including **wscompile**, which takes the service definition interface and generates the client-side stubs or server-side skeletons, or a WSDL description for the provided interface [30].

The J2EE 1.4 SDK includes the following tools:

- J2EE 1.4 Application Server
- Java 2 Platform, Standard Edition (J2SE) 1.4.2_01
- J2EE Samples
- Sun ONE Message Queue
- PointBase Database Server

11.3JSR 109 - Implementing Enterprise Web Services

The process of developing and deploying Web services is coupled with the runtime system. The JSR 109 specification promotes building portable and interoperable Web services in the J2EE 1.4 environment. JSR 109 leverages J2EE technologies to provide an industry standard for developing and deploying Web services on the J2EE platform, and it provides a service architecture that is familiar to J2EE developers [31]. This specification outlines the lifecycle of Web services to include:

- **Development**: Standardizes the Web services programming model as well as the deployment descriptors.
- **Deployment**: Describes the deployment actions expected of a J2EE 1.4 container.
- **Service publication**: Specifies how the WSDL is made available to clients.
- **Service consumption**: Standardizes the client deployment descriptors.

11.4J2EE Web Services

JAX-RPC is a Java API for XML-based Remote Procedure Calls (RPC's). An RPC is represented using an XML-based protocol such as SOAP, which defines an envelope structure, encoding rules, and convention for representing RPC calls and responses, which are transmitted as SOAP messages over HTTP [29] and [32].

See also Figure 8.2 chapter Web Services Architecture.

The advantage of JAX-RPC is that it hides the complexity of SOAP messages from the developer.

Here how it works:

The developer specifies the remote procedures (Web services) that can be invoked by remote clients in a Java programming language interface; the developer implements the interface. The client view of a Web service is a set of methods that perform business logic on behalf of the client. A client accesses a Web service using a Service Endpoint Interface as defined by JAX-RPC. Client developers create the client - a proxy or a local object that represents the remote service that is automatically generated - and then simply invoke the methods on the proxy. Generating or parsing SOAP messages is all taken care of by the JAX-RPC runtime system.

Note that J2EE Web services can be invoked by any Web service client, and any J2EE Web service client can invoke any Web service.

Figure 11.1 shows how a Java client communicates with a Java Web service in the J2EE 1.4 platform. J2EE applications can use Web services published by other providers, regardless of how they are implemented. In the case of non-Java technology-based clients and services, the Figure would change slightly.

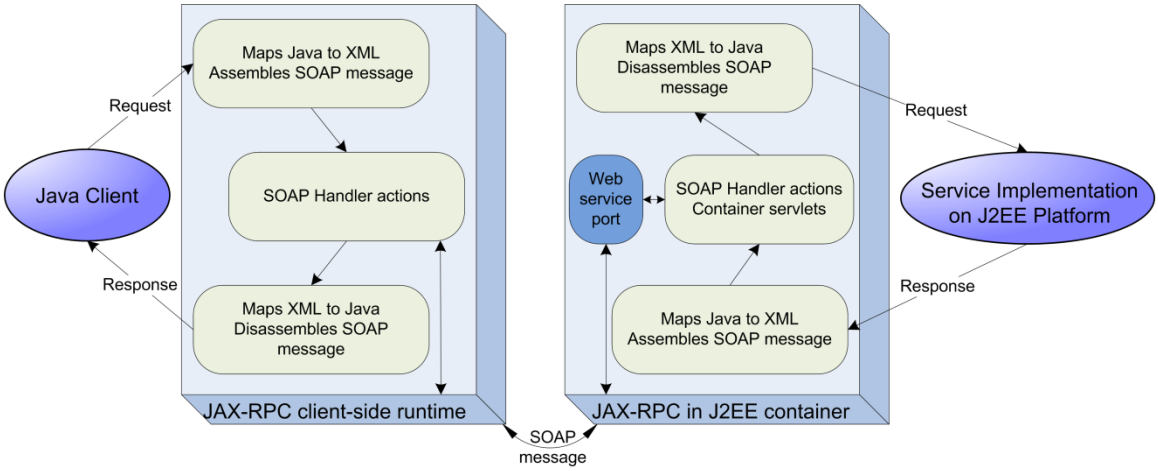

Figure 11.1: A Java Client Calling a J2EE Web Service

Note that a Web service client never accesses a service directly; it does so through the container. This is a good thing, since it allows a Web service to benefit from the added functionality that the container provides -- such as security, enhanced logging, and quality-of-service guarantees.

11.5Working with JAX-RPC

When working with JAX-RPC, remember that it maps Java types to XML/WSDL definitions. Knowing all the details of these mappings is not needed, but you should be aware that not all J2SE classes can be used as method parameters or return types in JAX-RPC [32].

JAX-RPC supports the following primitive data types:

- `boolean`
- `byte`
- `double`
- `float`
- `int`
- `long`
- `short`
- `string`
- `array`

In addition, it supports the following wrapper and utility classes:

- `java.lang.Boolean`
- `java.lang.Byte`
- `java.lang.Double`

- `java.lang.Float`
- `java.lang.Integer`
- `java.lang.Long`
- `java.lang.Short`
- `java.lang.String`
- `java.math.BigDecimal`
- `java.math.BigInteger`
- `java.net.URI`
- `java.util.Calendar`
- `java.util.Date`

JAX-RPC also supports something called a *value type*, which is a class that can be passed between a client and a service as a parameter or return value.

A value type must follow these rules:

- It must have a public default constructor.
- It must not implement `java.rmi.Remote`.
- Its fields must be JAX-RPC supported types.
- Also, a `public` field cannot be `final` or `transient`, and a non-public field must have corresponding `getter` and `setter` methods.

11.6 Creating a Web Service

Building an XML-RPC style Web service using the J2EE 1.4 platform involves the following five steps [29] and [31]:

1. Design and code the Web service endpoint interface.
2. Implement the service endpoint interface.
3. Write a configuration file.
4. Generate the necessary mapping files.
5. Packaging the service in a WAR file and deploy it using the deployment tool.

11.6.1 Design and Code the Service Endpoint Interface

The first step in creating a Web service is to design and code its endpoint interface, in which you declare the methods that a Web service remote client may invoke on the service.

Developing such an interface, must ensure that:

- It extends the `java.rmi.Remote` interface.

- It does not have constant declarations such as `public static final`.
- Its methods throw the `java.rmi.RemoteException` or one of its subclasses.
- Its method parameters and return data types are supported JAX-RPC types.

11.6.2 Implement the Service Endpoint Interface

The next step is to implement the interface and compile the `.java` files to generate the `.class` files.

Here the respective command:

```
prompt> javac -d build *.java
```

See chapter 12.1.2 for an example.

11.6.3 Write a Configuration File

The next step is to define a configuration file to be passed to the `wscompile` tool.

This file tells `wscompile` to create a WSDL file with the following information:

- The service name.
- The WSDL namespace.
- The package where the classes for the service are specified.
- The service endpoint interface.

See chapter 12.2for an example.

11.6.4 Generate the Necessary Mapping Files

Now, use the `wscompile` tool to generate the necessary files.

Consider the following command:

```
prompt> wscompile -define -mapping build/mapping.xml -d
build -nd build -classpath build config.xml
```

This command, which reads the `config.xml` file created earlier, creates the `*.wsdl` file and `mapping.xml`.

The command line options or flags are:

`-define`: instructs the tool to read the service endpoint interface and create a WSDL file.

`-mapping`: specifies the mapping file and where it should be written.

`-d` and `-nd`: specifies where to place generated output files and non-class output files.

Now a Web service that is ready to be packaged and deployed has been built.

The WSDL file, generated by the `wscompile` tool, provides an XML description (based on WSDL) of the service that clients can invoke. To understand the details of the file you need some knowledge of WSDL.

See chapter 12.3 for an example.

The mapping file, `mapping.xml`, generated by the `wscompile` tool follows the JSR 109 standard for Java ←→ WSDL mappings. The structure of the JAX-RPC mapping file matches closely with the structure of the WSDL file - note the relationship between Java packages and XML namespaces. Each service offered is represented as a `service-interface-mapping` element. This element contains the mapping for the fully qualified class name of the service interface, WSDL service names, and WSDL port names. In addition, the JAX-RPC mapping file provides mappings for WSDL bindings, WSDL port types, WSDL messages, and so on.

11.6.5 Packaging and Deploying the Service

A JAX-RPC Web service is a Web component, in J2EE terminology, and hence you can use `deploytool` to package and generate all the necessary configuration files, and to deploy the service.

Behind the scenes, a JAX-RPC Web service is implemented as a servlet. Because a servlet is a Web component, you run the New Web Component wizard of the deployment tool utility to package the service.

During this process the wizard performs the following tasks:

- Creating the Web application deployment descriptor.
- Creating a WAR file.
- Adding the deployment descriptor and service files to the WAR file.

See chapter 12.4 for an example.

11.7 Creating a Web Service Client

A client invokes a Web service in the same way as it invokes a method locally.

11.7.1 Types of Web Service Clients

There are the following three types of Web service clients:

- **Static Stub**: A Java class that is statically bound to a service endpoint interface. A stub, or a client proxy object, defines all the methods that the service endpoint interface defines. Therefore, the client can invoke methods of a Web service directly via the stub. The advantage of this is that it is simple and easy to code. The disadvantage is that the slightest change of Web service definition lead to the stub being useless and this means the stub must be regenerated. Use the static stub technique if you know that the Web service is stable and is not going to change its definition. A static stub is tied to the implementation which means, it is implementation-specific.
- **Dynamic Proxy**: Supports a service endpoint interface dynamically at runtime. Here, no stub code generation is required. A proxy is obtained at runtime and requires a service endpoint interface to be instantiated. As for

invocation, it is invoked in the same way as a stub. This is useful for testing Web services that may change their definitions. The dynamic proxy needs to be re-instantiated but not re-generated as is the case with stub.

- **Dynamic Invocation Interface (DII)**: Defines `javax.xml.rpc.Call` object instance for dynamic invocation. Unlike a stub or proxy, it must be configured before it can be used. A client needs to provide: operation name, parameter names, types, modes, and port type. As you can tell, much more coding is involved here. The major benefit is that since `Call` is not bound to anything, there is no impact of changes on the client side whenever the Web service definition changes.

11.7.2 Browser-Based Client

Finally, develop a Web client in which the Web service is invoked from a browser-based form (Java Server Page). For the implementation the static stub client method is used. The client calls the method through a stub, or a local object that acts as a client proxy to the remote service. It is called a static stub because the stub is generated before runtime by the `wscompile` tool.

Consider the following steps:

1. Before developing the Java client itself, you need to write a configuration file (in XML) that describes the location of the WSDL file.

   The URL in the configuration file identifies the location of the WSDL file. If you try this URL, you'd see the appropriate WSDL service file, assuming the Web service is deployed.

   See chapter 12.5.1 for an example.

2. Once you have written the configuration file, you are ready to generate client stubs, using the following command:

   ```
   prompt> wscompile -gen:client -d build -classpath build config-wsdl.xml
   ```

   This commands reads the `*.wsdl` (the location of which is specified in the `config-wsdl.xml`), then generates files based on the information in the WSDL file and on the command-line flags.

   The `-gen:client` instructs `wscompile` to generate the stubs, as well as other runtime files needed such as serializers and value types.

3. The next step is to write the Web client as a servlet or a Java Server Pages technology page (JSP).

   See chapter 12.5.2 for an example.

4. The last step is to package and deploy the Web client as a JSP Web component using the `deploytool`. The specified URL is to be used to access the service.

See chapter 12.5.3 for an example.

12 Pricing Web Service

The idea behind this Web service is to implement the presented models in the J2EE 1.4 platform and to use it for the valuation of options [29], [33] and [34].

12.1 Service Endpoint Interface

12.1.1 Designing

Due to the variety of models different services with partly dependencies among each other exist. Here the power of Web services is demonstrated by one service using or better supporting the other with its valuation results.

In Figure 12.1 the conceptual design of the Pricing Web service is given.

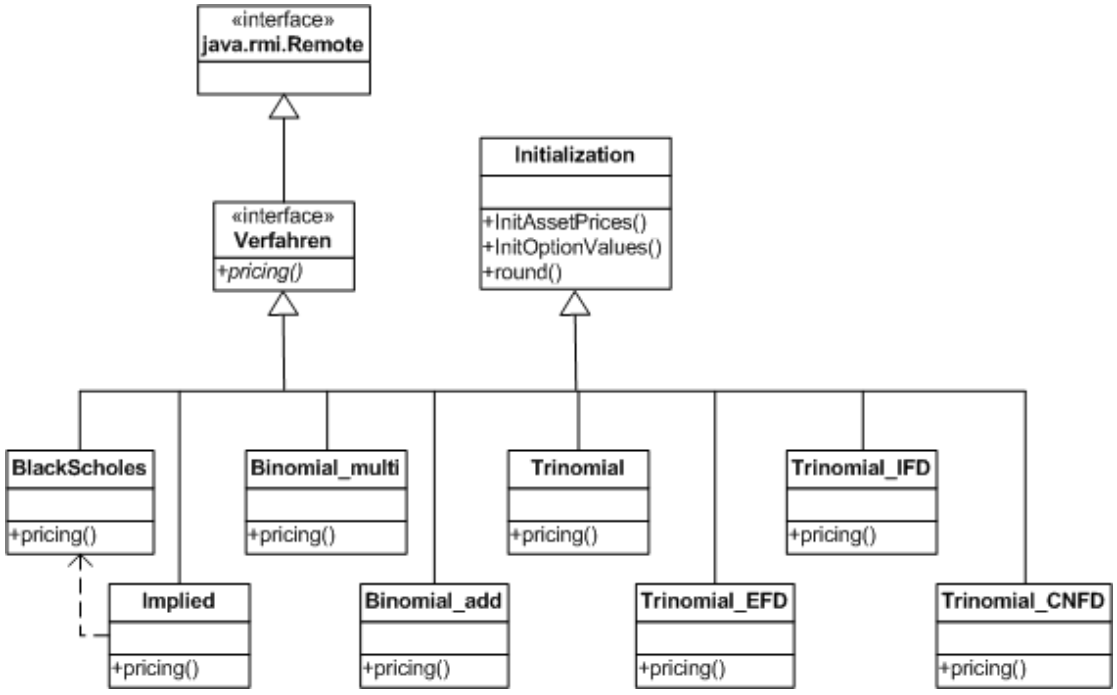

Figure 12.1 UML Diagram of the Pricing Web Service

12.1.2 Coding and Implementing

The interface file named `Verfahren.java` looks as follows:

```
package pricing;

import java.lang.*;
import java.rmi.Remote;
import java.rmi.RemoteException;


public interface Verfahren extends Remote
{
public abstract Lattice pricing(Instrument inst) throws
RemoteException;
}
```

12.2 Configuring

To describe the name of the service, its namespace, the package name and the name of the interface a configuration file is necessary.

This file named `config.xml` has the following look:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
  xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
  <service
      name="BS"
      targetNamespace="urn:Foo"
      typeNamespace="urn:Foo"
      packageName="pricing">
      <interface name="pricing.Verfahren"/>
  </service>
</configuration>
```

12.3 Mapping

The WSDL file, generated by the `wscompile` tool, provides an XML description of the service that clients can invoke and looks as follows:

```
<?xml version="1.0" encoding="UTF-8"?>

<definitions name="BS" targetNamespace="urn:Foo" xmlns:tns="urn:Foo"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <types>
    <schema targetNamespace="urn:Foo" xmlns:tns="urn:Foo"
xmlns:soap11-enc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns="http://www.w3.org/2001/XMLSchema">
      <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
      <complexType name="Instrument">
        <sequence>
          <element name="alpha" type="double"/>
          <element name="derivativ" type="tns:Derivativ"/>
          <element name="dx" type="double"/>
          <element name="underlying"
type="tns:Underlying"/></sequence></complexType>
      <complexType name="Derivativ">
        <sequence>
          <element name="b" type="double"/>
          <element name="barrier" type="boolean"/>
          <element name="barrierCondition"
type="tns:BarrierCondition"/>
          <element name="barrierDirection"
type="tns:BarrierDirection"/>
          <element name="k" type="double"/>
          <element name="n" type="double"/>
          <element name="optionType" type="tns:OptionType"/>
          <element name="optionn" type="tns:Optionn"/>
          <element name="reb" type="double"/>
          <element name="t" type="double"/></sequence></complexType>
      <complexType name="BarrierCondition">
        <sequence>
```

```
          <element name="value" type="int"/></sequence></complexType>
      <complexType name="BarrierDirection">
        <sequence>
          <element name="value" type="int"/></sequence></complexType>
      <complexType name="OptionType">
        <sequence>
          <element name="value" type="int"/></sequence></complexType>
      <complexType name="Optionn">
        <sequence>
          <element name="value" type="int"/></sequence></complexType>
      <complexType name="Underlying">
        <sequence>
          <element name="div" type="double"/>
          <element name="payDiv" type="boolean"/>
          <element name="r" type="double"/>
          <element name="s" type="double"/>
          <element name="sig"
type="double"/></sequence></complexType>
      <complexType name="Lattice">
        <sequence>
          <element name="m" type="int"/>
          <element name="n" type="int"/>
          <element name="results"
type="tns:ArrayOfArrayOfNode"/></sequence></complexType>
      <complexType name="ArrayOfArrayOfNode">
        <complexContent>
          <restriction base="soap11-enc:Array">
            <attribute ref="soap11-enc:arrayType"
wsdl:arrayType="tns:ArrayOfNode[]"/></restriction></complexContent></
complexType>
      <complexType name="ArrayOfNode">
        <complexContent>
          <restriction base="soap11-enc:Array">
            <attribute ref="soap11-enc:arrayType"
wsdl:arrayType="tns:Node[]"/></restriction></complexContent></complex
Type>
      <complexType name="Node">
        <sequence>
          <element name="empty" type="boolean"/>
          <element name="values"
type="tns:ArrayOfdouble"/></sequence></complexType>
      <complexType name="ArrayOfdouble">
        <complexContent>
          <restriction base="soap11-enc:Array">
            <attribute ref="soap11-enc:arrayType"
wsdl:arrayType="double[]"/></restriction></complexContent></complexTy
pe></schema></types>
  <message name="Verfahren_pricing">
    <part name="Instrument_1" type="tns:Instrument"/></message>
  <message name="Verfahren_pricingResponse">
    <part name="result" type="tns:Lattice"/></message>
  <portType name="Verfahren">
    <operation name="pricing" parameterOrder="Instrument_1">
      <input message="tns:Verfahren_pricing"/>
      <output
message="tns:Verfahren_pricingResponse"/></operation></portType>
  <binding name="VerfahrenBinding" type="tns:Verfahren">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="rpc"/>
    <operation name="pricing">
      <soap:operation soapAction=""/>
```

```
      <input>
        <soap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
use="encoded" namespace="urn:Foo"/></input>
      <output>
        <soap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
use="encoded" namespace="urn:Foo"/></output></operation></binding>
  <service name="BS">
    <port name="VerfahrenPort" binding="tns:VerfahrenBinding">
      <soap:address
location="REPLACE_WITH_ACTUAL_URL"/></port></service></definitions>
```

12.4 Packaging and Deploying

To package and generate all the necessary configuration files within a `*.war` Web application archive, `deploytool` is used.

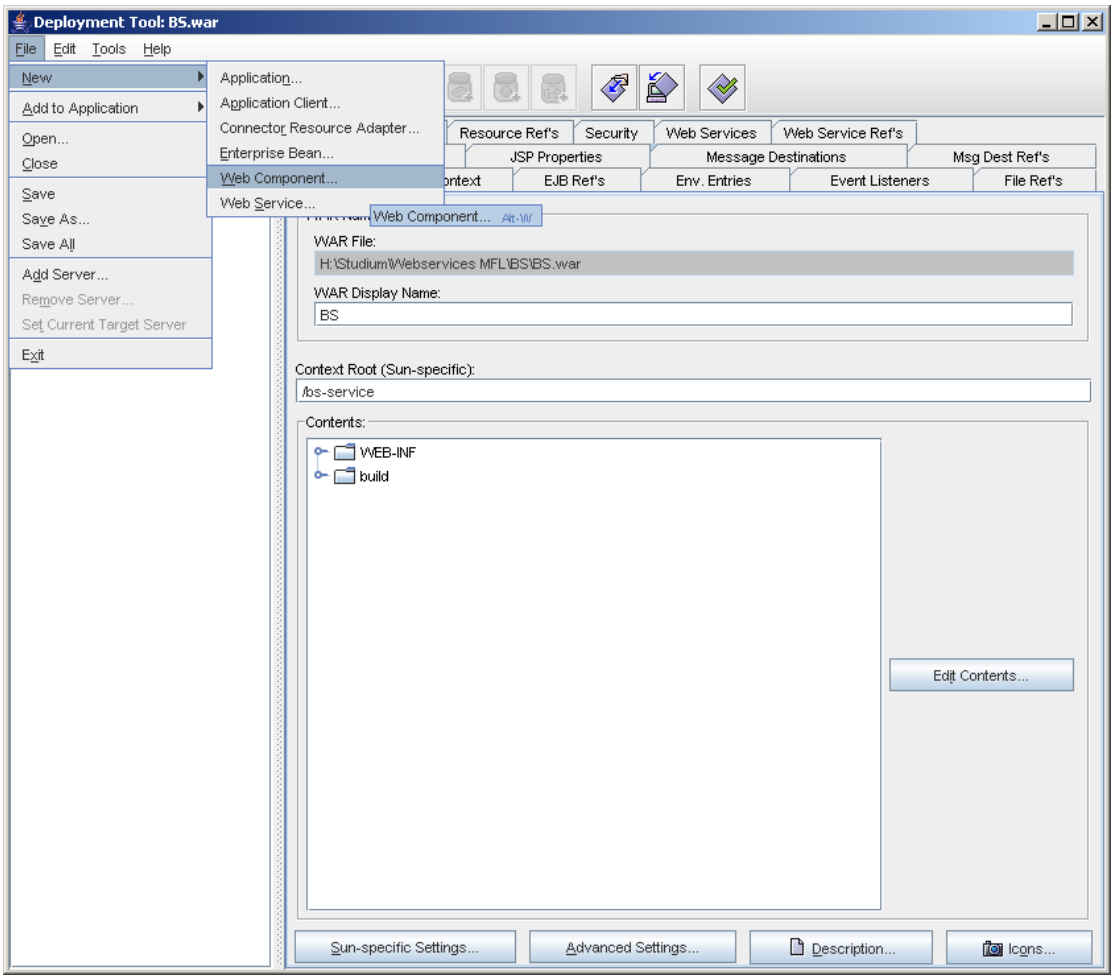

Figure 12.2 Deployment Tool - Packaging the Pricing Web Service

Finally `deploytool` is used to deploy the service.

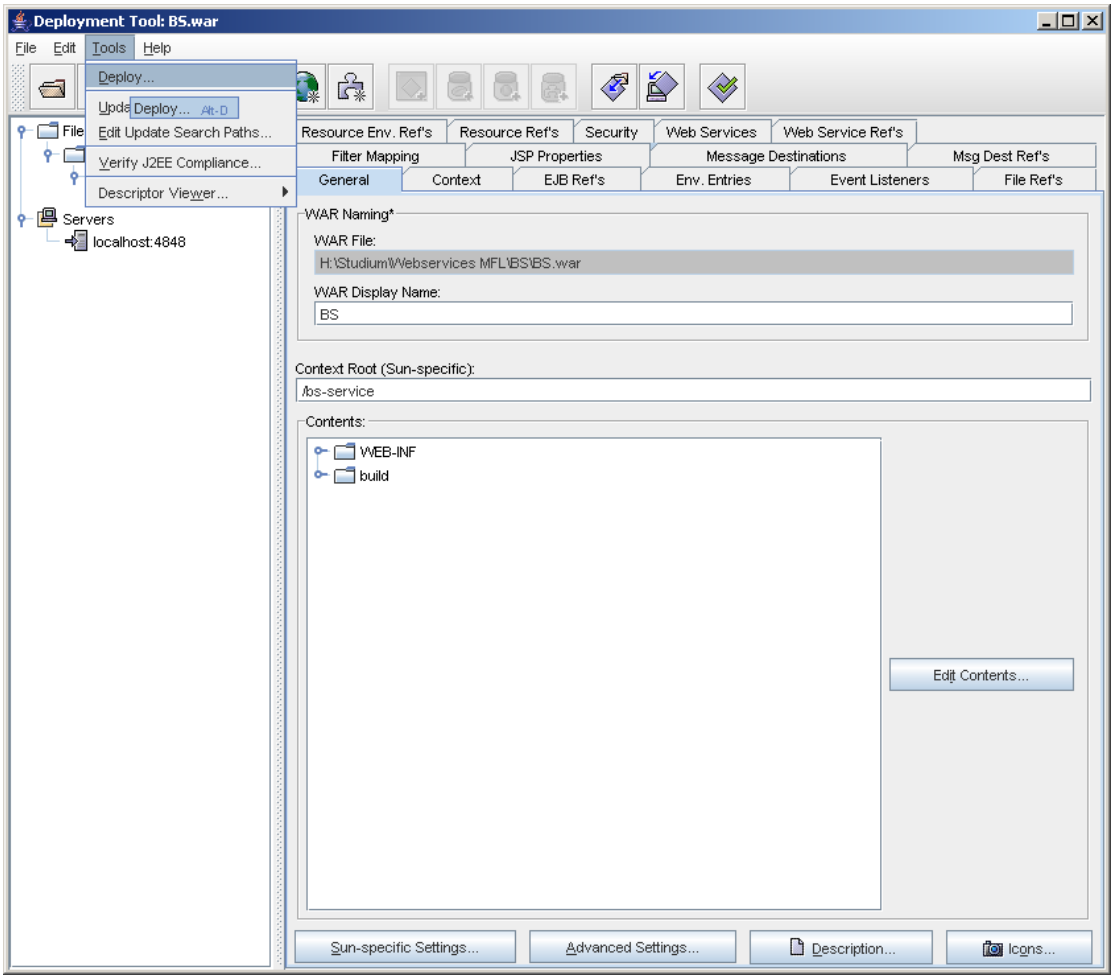

Figure 12.3: Deployment Tool – Deploying the Pricing Web Service

12.5 Web Client

12.5.1 Configuring and Generating Client Stubs

To describe the location of the service WSDL file and package name a configuration file is necessary.

This file named `config-wsdl.xml` has the following look:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
      xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
  <wsdl location="http://localhost:8080/bs-service/bs?WSDL"
       packageName="pricing"/>
</configuration>
```

12.5.2 Coding the Java Server Page

The next step is to write the Web client as a Java Server Pages technology page (JSP).

This page with the form to enter the input variables for the Pricing Web Service has the following look:

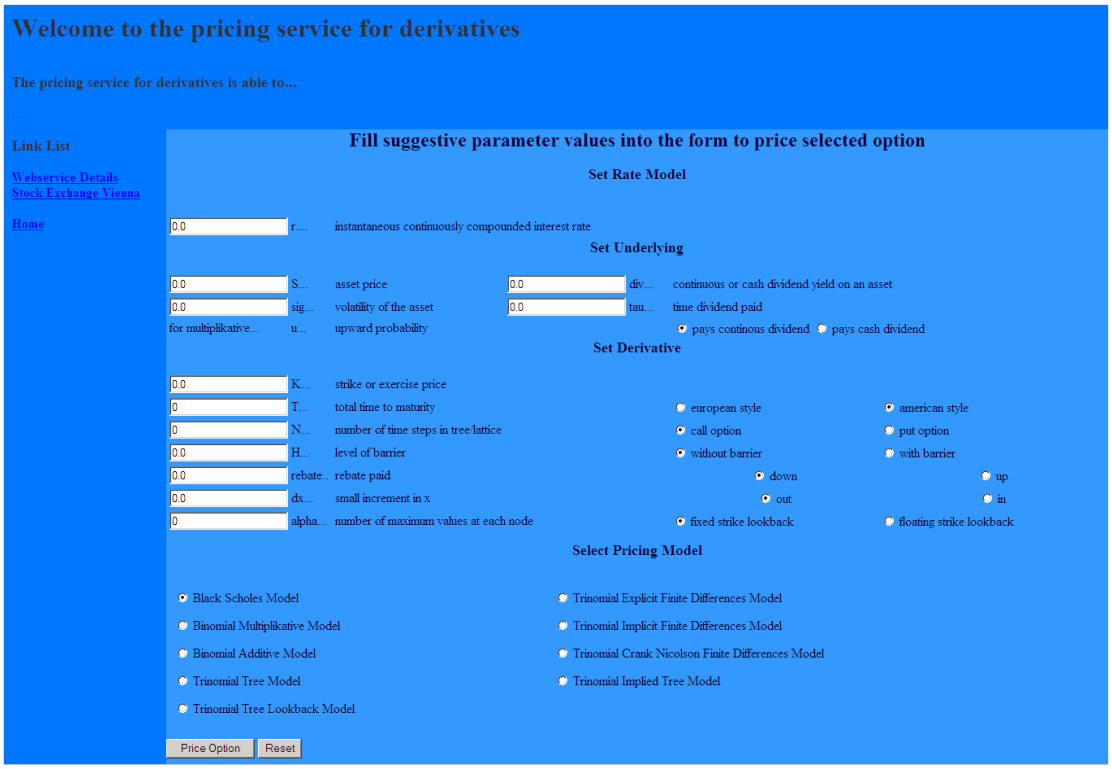

Figure 12.4: Web Client Java Server Page

12.5.3 Packaging and Deploying

To package the Web client as a JSP Web component and generate all the necessary configuration files within a `*.war` Web application archive, `deploytool` is used again. The specified URL is to be used to access the service.

12.6 Pricing Web Service Examples

12.6.1 Multiplicative Binomial Model

12.6.1.1 Pricing a European Call Option with Multiplicative Binomial Tree

Pricing a at-the-money European call option with one-year maturity and a current asset price of 100. The binomial tree has four time steps and up and down proportional jumps of 1.1 and 0.9091 respectively. The continuously compounded interest rate is assumed to be 4 per cent per annum -

i.e. $K = 100$, $T = 1$, $S = 100$, $r = 0.04$, $N = 4$, $u = 1.1$, and $d = 1/u = 0.9091$.

Figure 12.5 illustrates the numerical results, where nodes in the tree are represented by the boxes in which the upper value is the asset price and the lower value is the option price.

Pre-computing the constants:

$$\Delta t = \frac{T}{N} = \frac{1}{4} = 0.25$$
$$p = \frac{e^{r\times\Delta t} - d}{u - d} = \frac{e^{0.04\times 0.25} - 0.9091}{1.1 - 0.9091} = 0.5288$$
$$disc = e^{-r\times\Delta t} = e^{-0.04\times 0.3333} = 0.99$$

Computing the asset prices at maturity:

At node (4,0)

$$S_{4,0} = S\times d^{N} = 100\times 0.9091^{4} = 68.3013$$

At node (4,3)

$$S_{4,3} = S_{4,2}\times\frac{u}{d} = 100\times\frac{1.1}{0.9091} = 121.00$$

Computing the option values at maturity:

At node (4,3)

$$C_{4,3} = \max(0, S_{4,3} - K) = \max(0, 121.00 - 100.00) = 21.00$$

Performing discounted expectations back through the tree:

For node (3,3)

$$C_{3,3} = disc\times(p\times C_{4,4} + (1-p)\times C_{4,3}) = 0.99\times(0.5288\times 46.41 + (1-0.5288)\times 21.00 = 34.095$$

For node (0,0) - today -

$$C_{0,0} = disc\times(p\times C_{1,1} + (1-p)\times C_{1,0}) = 0.99\times(0.5288\times 14.7171 + (1-0.5288)\times 3.014 = 9.1115$$

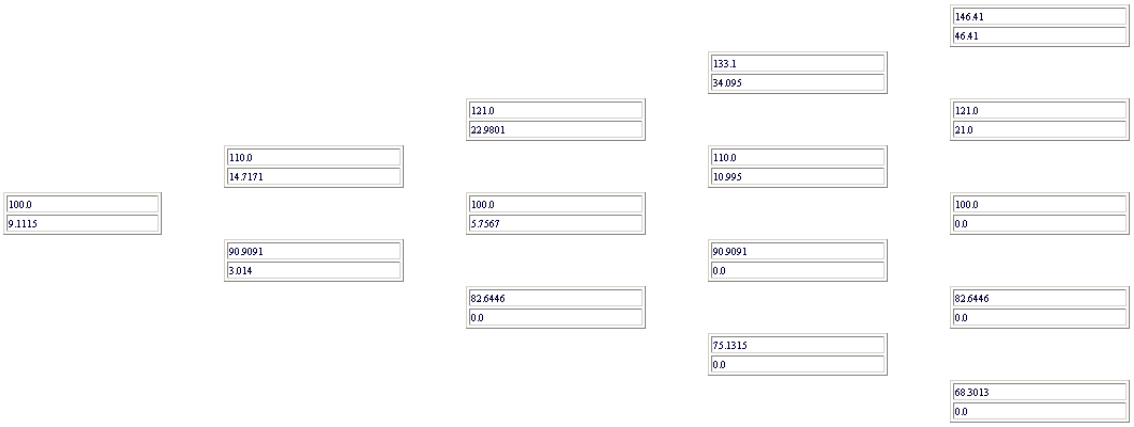

Figure 12.5: Pricing a European Call Option with Multiplicative Binomial Tree (JSP)

12.6.1.2 Pricing an American Put Option with Multiplicative Binomial Tree

Pricing an at-the-money American put option with one-year maturity and a current asset price of 100. The binomial tree has four time steps and up and down proportional jumps of 1.1 and 0.9091 respectively. The continuously compounded interest rate is assumed to be 4 per cent per annum -

i.e. $K = 100$, $T = 1$, $S = 100$, $r = 0.04$, $N = 4$, $u = 1.1$, and $d = 1/u = 0.9091$.

Figure 12.6 illustrates the numerical results, where nodes in the tree are represented by the boxes in which the upper value is the asset price and the lower value is the option price.

Pre-computing the constants:

$$\Delta t = \frac{T}{N} = \frac{1}{4} = 0.25$$

$$p = \frac{e^{r\times\Delta t} - d}{u - d} = \frac{e^{0.04\times 0.25} - 0.9091}{1.1 - 0.9091} = 0.5288$$

$$disc = e^{-r\times\Delta t} = e^{-0.04\times 0.3333} = 0.99$$

Computing the asset prices at maturity:

At node (4,0)

$$S_{4,0} = S\times d^{N} = 100\times 0.9091^{4} = 68.3013$$

at node (4,3)

$$S_{4,3} = S_{4,2} \times \frac{u}{d} = 100 \times \frac{1.1}{0.9091} = 121.00$$

Computing the option values at maturity:

At node (4,1)

$$C_{4,1} = \max(0, K - S_{4,1}) = \max(0, 100.00 - 82.6446) = 17.3554$$

Performing discounted expectations back through the tree:

For node (3,1)

$$C_{3,1} = \max((disc \times (p \times C_{4,2} + (1-p) \times C_{4,1})), K - S_{3,1}) = \max(8.0959, 100.00 - 90.9091) = 9.0909$$

For node (0,0) - today -

$$C_{0,0} = \max((disc \times (p \times C_{1,1} + (1-p) \times C_{1,0})), K - S_{0,0}) = \max(5.848, 100.00 - 100.00) = 5.848$$

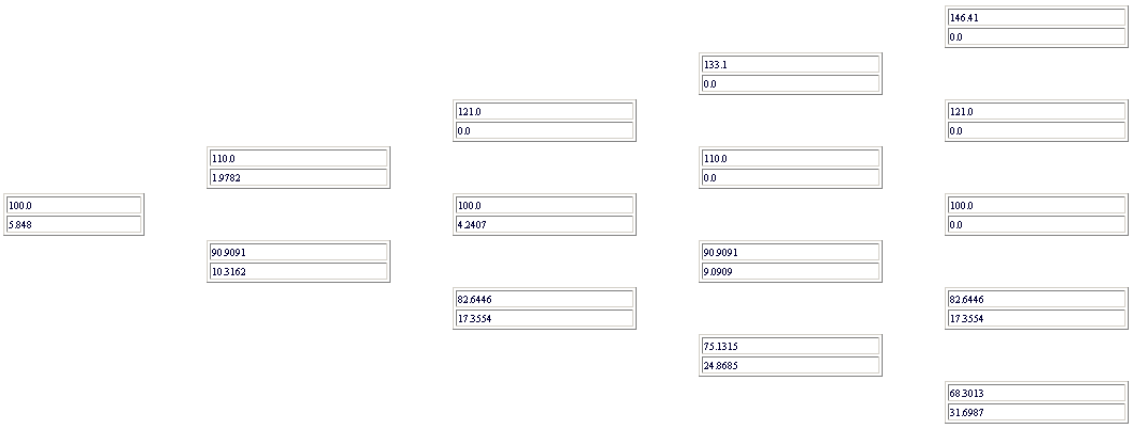

Figure 12.6: Pricing an American Put Option with Multiplicative Binomial Tree (JSP)

12.6.2 Additive Binomial Model

12.6.2.1 Pricing a European Call Option with Additive Binomial Tree

Pricing a at-the-money European call option with one-year maturity and a current asset price of 100 and volatility of 20 per cent. The continuously compounded interest rate is assumed to be 4 per cent per annum and the binomial tree has four time steps -

i.e. $K = 100$, $T = 1$, $S = 100$, $r = 0.04$, $N = 4$, $sig = 0.20$.

Figure 12.7 illustrates the numerical results, where nodes in the tree are represented by the boxes in which the upper value is the asset price and the lower value is the option price.

Pre-computing the constants:

$$\Delta t = \frac{T}{N} = \frac{1}{4} = 0.25$$
$$\nu = r - \frac{1}{2}\sigma^2 = 0.04 - \frac{1}{2}0.20^2 = 0.02$$
$$\Delta x_u = \sqrt{\sigma^2 \times \Delta t + (\nu \times \Delta t)^2} = \sqrt{0.20^2 \times 0.25 + (0.02 \times 0.25)^2} = 0.1001$$
$$\Delta x_d = -\Delta x_u = -0.1001$$
$$p_u = \frac{1}{2} + \frac{1}{2}\left(\frac{\nu \times \Delta t}{\Delta x_u}\right) = \frac{1}{2} + \frac{1}{2}\left(\frac{0.02 \times 0.25}{0.1001}\right) = 0.525$$
$$p_d = 1 - p_u = 1 - 0.525 = 0.475$$
$$disc = e^{-r \times \Delta t} = 0.99$$

Computing the asset prices at maturity:

At node (4,0)

$$S_{4,0} = S \times e^{N^* \Delta x_d} = 100 \times e^{4 \times (-0.1001)} = 66.9985$$

At node (4,1)

$$S_{4,1} = S_{4,0} \times e^{(\Delta x_u - \Delta x_d)} = 66.9985 \times e^{(0.1001 - (-0.1001))} = 81.8526$$

Computing the option values at maturity:

At node (4,3)

$$C_{4,3} = \max(0, S_{4,3} - K) = \max(0, 122.1708 - 100) = 22.1708$$

Performing discounted expectations back through the tree:

For node (3,2)

$$C_{3,2} = disc \times (p_u \times C_{4,3} + p_d \times C_{4,2}) = 0.99 \times (0.525 \times 22.1708 + 0.475 \times 0.0) = 11.5232$$

For node (0,0) - today -

$$C_{0,0} = disc \times (p_u \times C_{1,1} + p_d \times C_{1,0}) = 0.99 \times (0.525 \times 15.3659 + 0.475 \times 3.1128) = 9.4503$$

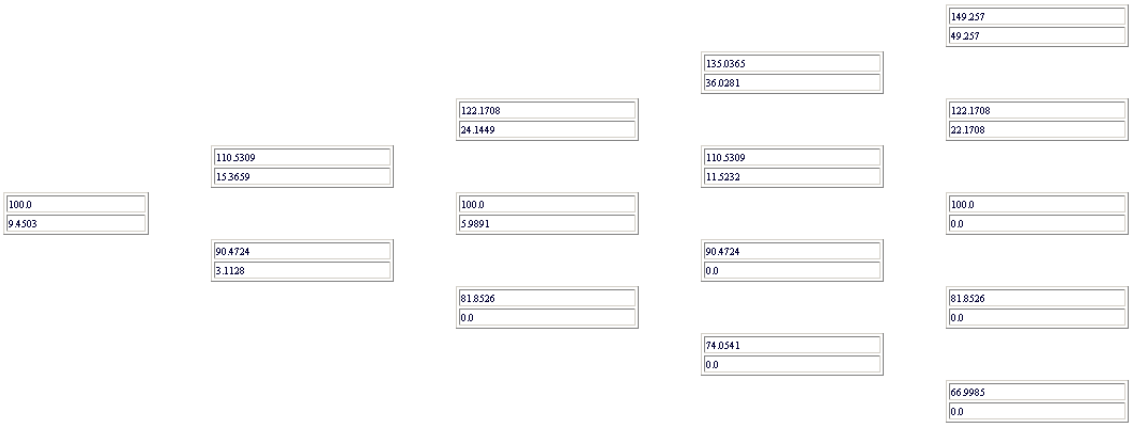

Figure 12.7: Pricing a European Call Option with Additive Binomial Tree (JSP)

12.6.2.2 Pricing an American Put Option with Additive Binomial Tree

Pricing an at-the-money American put option with one-year maturity and a current asset price of 100 and volatility of 20 per cent. The continuously compounded interest rate is assumed to be 4 per cent per annum and the binomial tree has four time steps -

i.e. $K = 100$, $T = 1$, $S = 100$, $r = 0.04$, $N = 4$, $sig = 0.20$.

Figure 12.8 illustrates the numerical results, where nodes in the tree are represented by the boxes in which the upper value is the asset price and the lower value is the option price.

Pre-computing the constants:

$$\Delta t = \frac{T}{N} = \frac{1}{4} = 0.25$$

$$\nu = r - \frac{1}{2}\sigma^2 = 0.04 - \frac{1}{2}0.20^2 = 0.02$$

$$\Delta x_u = \sqrt{\sigma^2 \times \Delta t + (\nu \times \Delta t)^2} = \sqrt{0.20^2 \times 0.25 + (0.02 \times 0.25)^2} = 0.1001$$

$$\Delta x_d = -\Delta x_u = -0.1001$$

$$p_u = \frac{1}{2} + \frac{1}{2}\left(\frac{\nu \times \Delta t}{\Delta x_u}\right) = \frac{1}{2} + \frac{1}{2}\left(\frac{0.02 \times 0.25}{0.1001}\right) = 0.525$$

$$p_d = 1 - p_u = 1 - 0.525 = 0.475$$

$$disc = e^{-r \times \Delta t} = 0.99$$

$$dpu = disc * p_u = 0.99 * 0.525 = 0.51975$$

$$dpd = disc * p_d = 0.99 * 0.475 = 0.47025$$

$$edxud = e^{(\Delta x_u - \Delta x_d)} = e^{(0.1001 - (-0.1001))} = 1.2216$$

$$edxd = e^{\Delta x_d} = e^{-0.1001} = 0.9047$$

Computing the asset prices at maturity:

At node (4,0)

$$S_{4,0} = S \times e^{N * \Delta x_d} = 100 \times e^{4 \times (-0.1001)} = 66.9985$$

At node (4,1)

$$S_{4,1} = S_{4,0} \times e^{(\Delta x_u - \Delta x_d)} = 66.9985 \times e^{(0.1001 - (-0.1001))} = 81.8526$$

Computing the option values at maturity:

At node (4,1)

$$C_{4,1} = \max(0, K - S_{4,1}) = \max(0, 100 - 81.8526) = 18.1474$$

Performing discounted expectations back through the tree:

For node (3,1)

$$C_{3,1} = dpu \times C_{4,2} + dpd \times C_{4,1} = 0.51975 \times 0.0 + 0.47025 \times 18.1474 = 8.5338$$

Computing the asset price

$$S_{3,1} = \frac{S_{4,1}}{edxd} = \frac{81.8526}{0.9047} = 90.4724$$

Applying the early exercise test

$$C_{3,1} = \max(C_{3,1}, K - S_{3,1}) = \max(8.5338, 100.00 - 90.4724) = 9.5276$$

For node (0,0) - today -

$$C_{0,0} = \max(C_{0,0}, K - S_{0,0}) = \max((0.51975 \times 2.1074 + 0.47025 \times 10.8637), 100 - 100) = 6.2045$$

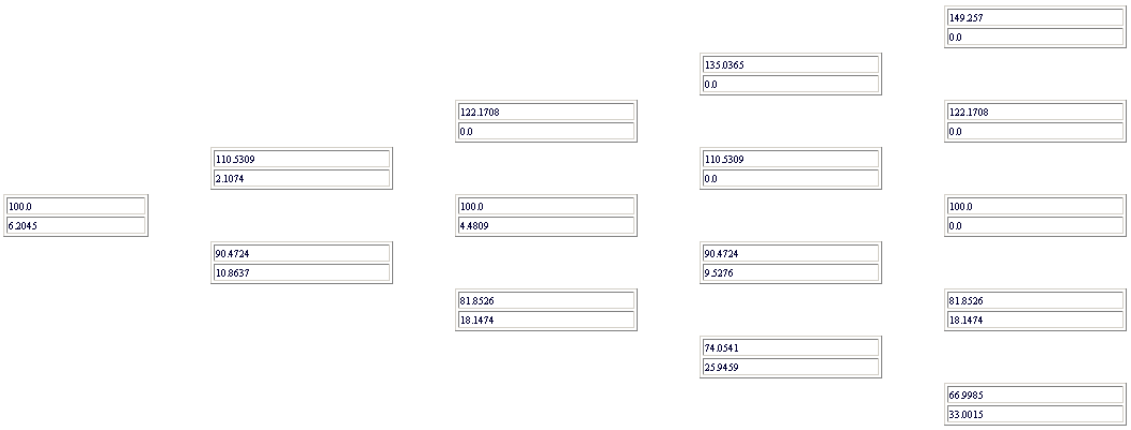

Figure 12.8: Pricing an American Put Option with Additive Binomial Tree (JSP)

12.6.2.3 Pricing an American Put Option with a Known Discrete Cash Dividend

Pricing an at-the-money American put option with one-year maturity and a current asset price of 100 and volatility of 20 per cent. The continuously compounded interest rate is assumed to be 4 per cent per annum and the binomial tree has four time steps. The asset pays a discrete cash dividend of 3 after six months -

i.e. $K = 100$, $T = 1$, $S = 100$, $r = 0.04$, $N = 4$, $\sigma = 0.20$, $D = 3$, $\tau = 0.5$.

Figure 12.9 illustrates the numerical results, where nodes in the tree are represented by the boxes in which the upper value is the asset price and the lower value is the option price.

Pre-computing the constants:

$$\Delta t = \frac{T}{N} = \frac{1}{4} = 0.25$$

$$\nu = r - \frac{1}{2}\sigma^2 = 0.04 - \frac{1}{2}0.20^2 = 0.02$$

$$\Delta x_u = \sqrt{\sigma^2 \times \Delta t + (\nu \times \Delta t)^2} = \sqrt{0.20^2 \times 0.25 + (0.02 \times 0.25)^2} = 0.1001$$

$$\Delta x_d = -\Delta x_u = -0.1001$$

$$p_u = \frac{1}{2} + \frac{1}{2}\left(\frac{\nu \times \Delta t}{\Delta x_u}\right) = \frac{1}{2} + \frac{1}{2}\left(\frac{0.02 \times 0.25}{0.1001}\right) = 0.525$$

$$p_d = 1 - p_u = 1 - 0.525 = 0.475$$

$$disc = e^{-r \times \Delta t} = 0.99$$

$$dpu = disc * p_u = 0.99 * 0.525 = 0.51975$$

$$dpd = disc * p_d = 0.99 * 0.475 = 0.47025$$

$$edxud = e^{(\Delta x_u - \Delta x_d)} = e^{(0.1001 - (-0.1001))} = 1.2216$$

$$edxd = e^{\Delta x_d} = e^{-0.1001} = 0.9047$$

Computing the asset prices at maturity:

At node (4,0)

$$S_{4,0} = (S - D \times e^{-r \times \tau})e^{N \times \Delta x_d} = (100 - 3.0 \times e^{-0.04 \times 0.5})e^{4 \times (-0.1001)} = 65.0248$$

At node (4,2)

$$S_{4,2} = S_{4,1} \times edxud = 79.4457 \times 1.2216 = 97.0594$$

Computing the option values at maturity:

At node (4,1)

$$C_{4,1} = \max(0, K - S_{4,1}) = \max(0, 100 - 79.4457) = 20.5543$$

Performing discounted expectations back through the tree:

For node (3,0)

$$C_{3,0} = dpu \times C_{4,1} + dpd \times C_{4,0} = 0.51975 \times 20.5543 + 0.47025 \times 34.9716 = 27.1285$$

Computing the asset price

$$S_{3,0} = \frac{S_{4,0}}{edxd} = \frac{65.0248}{0.9047} = 71.8764$$

Applying the early exercise test

$$C_{3,0} = \max(C_{3,0}, K - S_{3,0}) = \max(27.1285, 100 - 71.8764) = 28.1236$$

For node (0,0) - today -

$$C_{0,0} = dpu \times C_{1,1} + dpd \times C_{1,0} = 0.51975 \times 3.3719 + 0.47025 \times 13.0196 = 7.8757$$

Computing the asset price

$$S_{0,0} = \frac{S_{1,0}}{edxd} + D \times e^{-r(\tau - t)} = \frac{87.812}{0.9047} + 3.00 \times e^{-0.04(0.5 - 0.25)} = 100.0322$$

Applying the early exercise test

$$C_{0,0} = \max(C_{0,0}, K - S_{0,0}) = \max(7.8757, 100 - 100.0322) = 7.8757$$

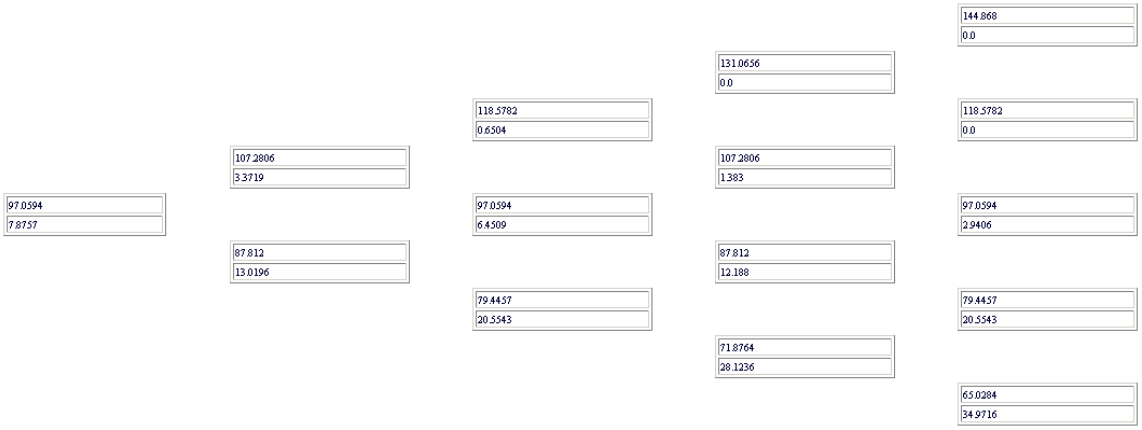

Figure 12.9: Pricing an American Put Option with a Known Discrete Cash Dividend (JSP)

12.6.2.4 Pricing an American Down-and-Out Call Option with Additive Binomial Tree

Pricing an at-the-money American down-and-out call option with one-year maturity and a current asset price of 100 and volatility of 20 per cent. The barrier is set at 110, the continuously compounded interest rate is assumed to be 4 per cent per annum and the binomial tree has four time steps -

i.e. $K = 100$, $T = 1$, $S = 100$, $r = 0.04$, $N = 4$, $\sigma = 0.20$, $H = 110$.

Figure 12.10 illustrates the numerical results, where nodes in the tree are represented by the boxes in which the upper value is the asset price and the lower value is the option price.

Pre-computing the constants:

$$\Delta t = \frac{T}{N} = \frac{1}{4} = 0.25$$

$$\nu = r - \frac{1}{2}\sigma^2 = 0.04 - \frac{1}{2}0.20^2 = 0.02$$

$$\Delta x_u = \sqrt{\sigma^2 \times \Delta t + (\nu \times \Delta t)^2} = \sqrt{0.20^2 \times 0.25 + (0.02 \times 0.25)^2} = 0.1001$$

$$\Delta x_d = -\Delta x_u = -0.1001$$

$$p_u = \frac{1}{2} + \frac{1}{2}\left(\frac{\nu \times \Delta t}{\Delta x_u}\right) = \frac{1}{2} + \frac{1}{2}\left(\frac{0.02 \times 0.25}{0.1001}\right) = 0.525$$

$$p_d = 1 - p_u = 1 - 0.525 = 0.475$$

$$disc = e^{-r \times \Delta t} = 0.99$$

$$dpu = disc * p_u = 0.99 * 0.525 = 0.51975$$

$$dpd = disc * p_d = 0.99 * 0.475 = 0.47025$$

$$edxud = e^{(\Delta x_u - \Delta x_d)} = e^{(0.1001 - (-0.1001))} = 1.2216$$

$$edxd = e^{\Delta x_d} = e^{-0.1001} = 0.9047$$

Computing the asset prices at maturity:

At node (4,0)

$$S_{4,0} = S \times e^{N \times \Delta x_d} = 100 \times e^{4 \times (-0.1001)} = 66.9985$$

At node (4,2)

$$S_{4,2} = S_{4,1} \times e^{(\Delta x_u - \Delta x_d)} = 81.8526 \times e^{(0.1001 - (-0.1001))} = 100.00$$

Computing the option values at maturity:

At node (4,3) $S_{4,3} > H$ and therefore

$$C_{4,3} = \max(0, S_{4,3} - K) = \max(0,122.1708 - 100) = 22.1708$$

Performing discounted expectations back through the tree and applying the barrier condition:

At node (0,0) $S_{0,0} < H$ and therefore

$$C_{0,0} = 0.0$$

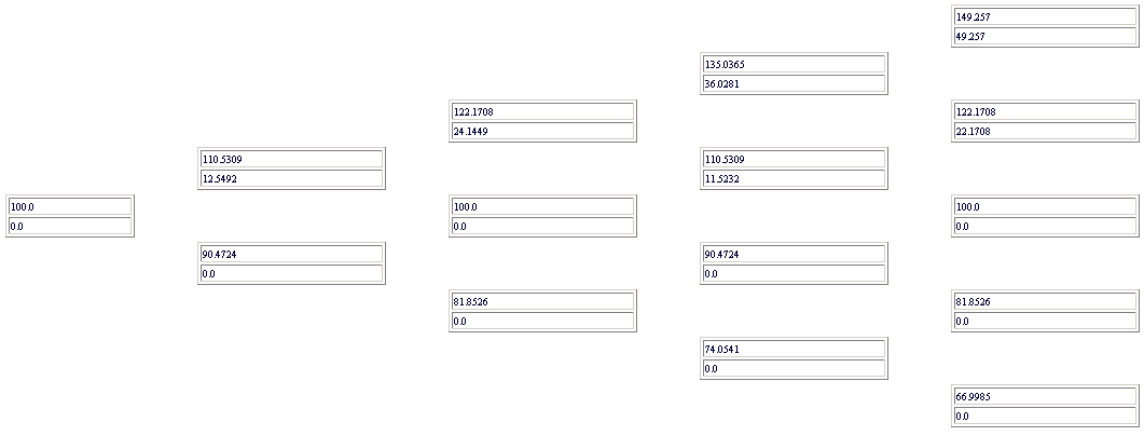

Figure 12.10: Pricing an American Down-and-Out Call Option with Additive Binomial Tree (JSP)

12.6.3 Trinomial and Finite Difference Models

12.6.3.1 Pricing a European Call Option in a Trinomial Tree

Pricing a at-the-money European call option with one-year maturity and a current asset price of 100 and volatility of 20 per cent. The continuously compounded interest rate is assumed to be 4 per cent per annum, the asset pays a continuous dividend yield of 3 per cent per annum, the trinomial tree has four time steps and the space step is 0.2 -

i.e. $K = 100$, $T = 1$, $S = 100$, $r = 0.04$, $N = 4$, $\sigma = 0.20$, $\delta = 0.03$, $\Delta x = 0.2$.

Figure 12.11 illustrates the numerical results, where nodes in the tree are represented by the boxes in which the upper value is the asset price and the lower value is the option price.

Pre-computing the constants:

$$\Delta t = \frac{T}{N} = \frac{1}{4} = 0.25$$

$$\nu = r - \delta - \frac{1}{2}\sigma^2 = 0.04 - 0.03 - 0.5 \times 0.02^2 = 0.0098$$

$$edx = e^{\Delta x} = e^{0.2} = 1.2214$$

$$p_u = \frac{1}{2}\left(\frac{\sigma^2 \Delta t + \nu^2 \Delta t^2}{\Delta x^2} + \frac{\nu \Delta t}{\Delta x}\right) = \frac{1}{2}\left(\frac{0.2^2 \times 0.25 + 0.0098^2 \times 0.25^2}{0.2^2} + \frac{0.0098 \times 0.25}{0.2}\right) = 0.1312$$

$$p_m = 1 - \left(\frac{\sigma^2 \Delta t + \nu^2 \Delta t^2}{\Delta x^2}\right) = 1 - \left(\frac{0.2^2 \times 0.25 + 0.0098^2 \times 0.25^2}{0.2^2}\right) = 0.7498$$

$$p_d = \frac{1}{2}\left(\frac{\sigma^2 \Delta t + \nu^2 \Delta t^2}{\Delta x^2} - \frac{\nu \Delta t}{\Delta x}\right) = \frac{1}{2}\left(\frac{0.2^2 \times 0.25 + 0.0098^2 \times 0.25^2}{0.2^2} - \frac{0.0098 \times 0.25}{0.2}\right) = 0.119$$

$$disc = e^{-r \times \Delta t} = 0.99$$

Computing the asset prices at maturity:

At node (4, 4)

$$S_{4,-4} = S \times e^{-N \times \Delta x} = 100 \times e^{-4 \times 0.2} = 44.9329$$

At node (4,3)

$$S_{4,-3} = S_{4,-4} \times edx = 44.9329 \times 1.2214 = 54.8812$$

Computing the option values at maturity:

At node (4,2)

$$C_{4,2} = \max(0, S_{4,2} - K) = \max(0, 149.1825 - 100.00) = 49.1825$$

Performing discounted expectations back through the tree:

$$C_{3,1} = disc \times (p_u \times C_{4,2} + p_m \times C_{4,1} + p_d \times C_{4,0}) =$$
$$0.99 \times (0.1312 \times 49.1825 + 0.7498 \times 22.1403 + 0.119 \times 0.0) = 22.2227$$

At node (0,0) - today -

$$C_{0,0} = disc \times (p_u \times C_{1,2} + p_m \times C_{1,1} + p_d \times C_{1,0}) =$$
$$0.99 \times (0.1312 \times 23.1332 + 0.7498 \times 6.0761 + 0.119 \times 0.7626) = 7.3314$$

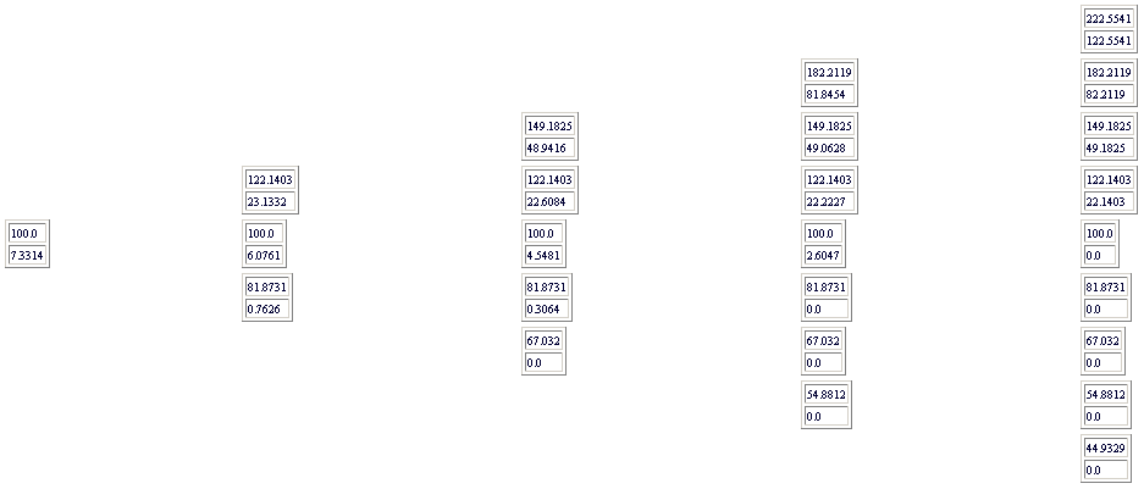

Figure 12.11: Pricing a European Call Option in a Trinomial Tree (JSP)

12.6.3.2 Pricing a European Call Option by Explicit Finite Difference Model

Pricing a at-the-money European call option with one-year maturity and a current asset price of 100 and volatility of 20 per cent. The continuously compounded interest rate is assumed to be 4 per cent per annum, the asset pays a continuous dividend yield of 3 per cent per annum, the trinomial tree has four time steps and the space step is 0.2 -

i.e. $K = 100$, $T = 1$, $S = 100$, $r = 0.04$, $N = 4$, $\sigma = 0.20$, $\delta = 0.03$, $\Delta x = 0.2$.

Figure 12.12 illustrates the numerical results, where nodes in the lattice are represented by the boxes in which the upper value is the asset price and the lower value is the option price.

Pre-computing the constants:

$$\Delta t = \frac{T}{N} = \frac{1}{4} = 0.25$$

$$\nu = r - \delta - \frac{1}{2}\sigma^2 = 0.04 - 0.03 - 0.5 \times 0.02^2 = 0.0098$$

$$edx = e^{\Delta x} = e^{0.2} = 1.2214$$

$$p_u = \frac{1}{2}\Delta t\left(\left(\frac{\sigma}{\Delta x}\right)^2 + \frac{\nu}{\Delta x}\right) = \frac{1}{2} \times 0.25 \times \left(\left(\frac{0.2}{0.2}\right)^2 + \frac{0.0098}{0.2}\right) = 0.1311$$

$$p_m = 1 - \Delta t\left(\frac{\sigma}{\Delta x}\right)^2 - r\Delta t = 1 - 0.25 \times \left(\frac{0.2}{0.2}\right)^2 - 0.04 \times 0.25 = 0.75$$

$$p_d = \frac{1}{2}\Delta t\left(\left(\frac{\sigma}{\Delta x}\right)^2 - \frac{\nu}{\Delta x}\right) = \frac{1}{2} \times 0.25 \times \left(\left(\frac{0.2}{0.2}\right)^2 - \frac{0.0098}{0.2}\right) = 0.1189$$

Computing the asset prices at maturity:

At node (4,-4)

$$S_{4,-4} = S \times e^{-N \times \Delta x} = 100 \times e^{-4 \times 0.2} = 44.9329$$

At node (4,-2)

$$S_{4,-2} = S_{4,-3} \times edx = 54.8812 \times 1.2214 = 67.032$$

Computing the option values at maturity:

At node (4,2)

$$C_{4,2} = \max(0, S_{4,2} - K) = \max(0,149.1825 - 100.00) = 49.1825$$

Performing discounted expectations back through the tree:

At node (3,1)

$$C_{3,1} = (p_u \times C_{4,2} + p_m \times C_{4,1} + p_d \times C_{4,0}) = 0.1311 \times 49.1825 + 0.75 \times 22.1403 + 0.1189 \times 0.0) = 22.2242$$

At node (0,0) - today -

$$C_{0,0} = (p_u \times C_{1,1} + p_m \times C_{1,0} + p_d \times C_{1,-1}) = 0.1311 \times 23.1505 + 0.75 \times 6.1195 + 0.1189 \times 0.7755) = 7.3793$$

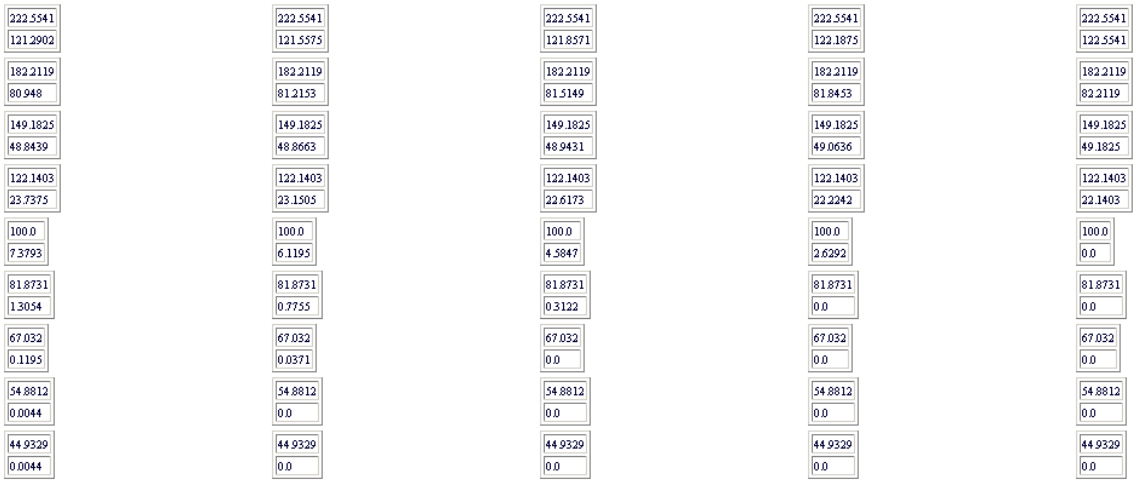

Figure 12.12: Pricing a European Call Option by Explicit Finite Difference Model (JSP)

12.6.3.3 Pricing an American Put Option by Explicit Finite Difference Model

Pricing an at-the-money American put option with one-year maturity and a current asset price of 100 and volatility of 20 per cent. The continuously compounded interest rate is assumed to be 4 per cent per annum, the asset pays a continuous dividend yield of 3 per cent per annum, the trinomial tree has four time steps and the space step is 0.2 -

i.e. $K = 100$, $T = 1$, $S = 100$, $r = 0.04$, $N = 4$, $\sigma = 0.20$, $\delta = 0.03$, $\Delta x = 0.2$.

Figure 12.13 illustrates the numerical results, where nodes in the lattice are represented by the boxes in which the upper value is the asset price and the lower value is the option price.

Pre-computing the constants:

$$\Delta t = \frac{T}{N} = \frac{1}{4} = 0.25$$

$$\nu = r - \delta - \frac{1}{2}\sigma^2 = 0.04 - 0.03 - 0.5 \times 0.02^2 = 0.0098$$

$$edx = e^{\Delta x} = e^{0.2} = 1.2214$$

$$p_u = \frac{1}{2}\Delta t\left(\left(\frac{\sigma}{\Delta x}\right)^2 + \frac{\nu}{\Delta x}\right) = \frac{1}{2} \times 0.25 \times \left(\left(\frac{0.2}{0.2}\right)^2 + \frac{0.0098}{0.2}\right) = 0.1311$$

$$p_m = 1 - \Delta t\left(\frac{\sigma}{\Delta x}\right)^2 - r\Delta t = 1 - 0.25 \times \left(\frac{0.2}{0.2}\right)^2 - 0.04 \times 0.25 = 0.75$$

$$p_d = \frac{1}{2}\Delta t\left(\left(\frac{\sigma}{\Delta x}\right)^2 - \frac{\nu}{\Delta x}\right) = \frac{1}{2} \times 0.25 \times \left(\left(\frac{0.2}{0.2}\right)^2 - \frac{0.0098}{0.2}\right) = 0.1189$$

Computing the asset prices at maturity:

$$S_{4,-3} = S \times e^{-N \times \Delta x} = 100 \times e^{-4 \times 0.2} = 54.8812$$

At node (4,-2)

$$S_{4,-2} = S_{4,-3} \times edx = 54.8812 \times 1.2214 = 67.032$$

Computing the option values at maturity:

At node (4,-2)

$$C_{4,-2} = \max(0, K - S_{4,-2}) = \max(0,100.00 - 67.032) = 32.968$$

Performing discounted expectations back through the tree:

At node (3,-1)

$$C_{3,-1} = p_u \times C_{4,0} + p_m \times C_{4,-1} + p_d \times C_{4,-2} = 0.1311 \times 0.0 + 0.75 \times 18.1269 + 0.1189 \times 32.968) = 18.1269$$

Applying the early exercise test:

$$C_{2,-1} = \max(C_{2,-1}, K - S_{2,-1}) = \max(16.942,100 - 81.8731) = 18.1269$$

At node (0,0) - today -

$$C_{0,0} = p_u \times C_{1,1} + p_m \times C_{1,0} + p_d \times C_{1,-1} = 0.1311 \times 0.7744 + 0.75 \times 5.4796 + 0.1189 \times 18.2326) = 6.5399$$

Applying the early exercise test:

$$C_{0,0} = \max(C_{0,0}, K - S_{0,0}) = \max(6.5399,100 - 100) = 6.5399$$

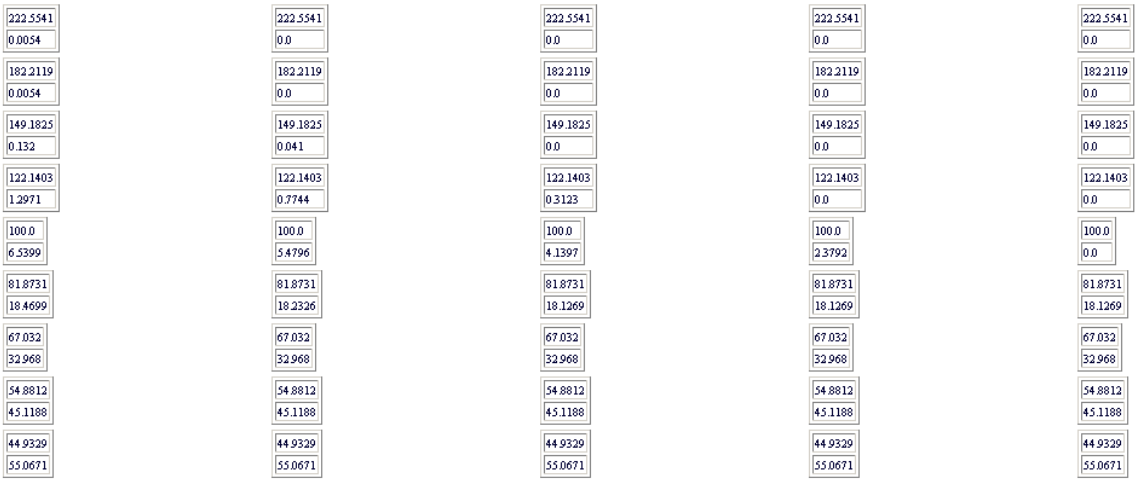

Figure 12.13: Pricing an American Put Option by Explicit Finite Difference Model (JSP)

12.6.3.4 Pricing an American Put Option by Implicit Finite Difference Model

Pricing an at-the-money American put option with one-year maturity and a current asset price of 100 and volatility of 20 per cent. The continuously compounded interest rate is assumed to be 4 per cent per annum, the asset pays a continuous dividend yield of 3 per cent per annum, the trinomial tree has four time steps and the space step is 0.2 -

i.e. $K = 100$, $T = 1$, $S = 100$, $r = 0.04$, $N = 4$, $\sigma = 0.20$, $\delta = 0.03$, $\Delta x = 0.2$.

Figure 12.14 illustrates the numerical results, where nodes in the lattice are represented by the boxes in which the upper value is the asset price and the lower value is the option price.

Pre-computing the constants:

$$\Delta t = \frac{T}{N} = \frac{1}{4} = 0.25$$

$$\nu = r - \delta - \frac{1}{2}\sigma^2 = 0.04 - 0.03 - 0.5 \times 0.02^2 = 0.0098$$

$$edx = e^{\Delta x} = e^{0.2} = 1.2214$$

$$p_u = -\frac{1}{2}\Delta t\left(\left(\frac{\sigma}{\Delta x}\right)^2 + \frac{\nu}{\Delta x}\right) = -\frac{1}{2} \times 0.25 \times \left(\left(\frac{0.2}{0.2}\right)^2 + \frac{0.0098}{0.2}\right) = -0.1311$$

$$p_m = 1 + \Delta t\left(\frac{\sigma}{\Delta x}\right)^2 + r\Delta t = 1 + 0.25 \times \left(\frac{0.2}{0.2}\right)^2 + 0.04 \times 0.25 = 1.26$$

$$p_d = -\frac{1}{2}\Delta t\left(\left(\frac{\sigma}{\Delta x}\right)^2 - \frac{\nu}{\Delta x}\right) = -\frac{1}{2} \times 0.25 \times \left(\left(\frac{0.2}{0.2}\right)^2 - \frac{0.0098}{0.2}\right) = -0.1189$$

Computing the asset prices at maturity:

At node (4,-3)

$$S_{4,-3} = S \times e^{-N \times \Delta x} = 100 \times e^{-4 \times 0.2} = 54.8812$$

At node (4,-2)

$$S_{4,-2} = S_{4,-3} \times edx = 54.8812 \times 1.2214 = 67.032$$

Computing the option values at maturity:

At node (4,-2)

$$C_{4,-2} = \max(0, K - S_{4,-2}) = \max(0,100.00 - 67.032) = 32.968$$

Performing discounted expectations back through the tree and solving the tri-diagonal system of equations:

At node (2,-2) the upper diagonal is eliminated:

$$p'_{m,-2} = p_m + p_d = 1.26 + (-0.1189) = 1.1411$$
$$p'_{-2} = C_{4,-2} + p_d \times \lambda_L = 32.9680 + (-0.1189) \times ((-1) \times (67.032 - 54.8812)) = 34.4127$$

At node (2,-1)

$$p'_{m,-1} = p_m - \frac{p_u}{p_{m,-2}} p_d = 1.26 - (-0.1311) \times (-0.1189) / 1.1411 = 1.2463$$
$$p'_{-1} = C_{4,-1} - \frac{p'_{-2}}{p_{m,-2}} p_d = 18.1269 - 34.4127 \times (-0.1189) / 1.1411 = 21.7126$$

At node (3,-1) back substituting:

$$C_{3,-1} = \frac{p'_{-1} - p_u C_{3,0}}{p'_{m,-1}} = (21.7126 - (-0.1311) \times 1.8885) / 1.2463 = 17.6203$$

Applying the early exercise test:

$$C_{3,-1} = \max(C_{3,-1}, K - S_{3,-1}) = \max(17,6203,100 - 81.8731) = 18.1269$$

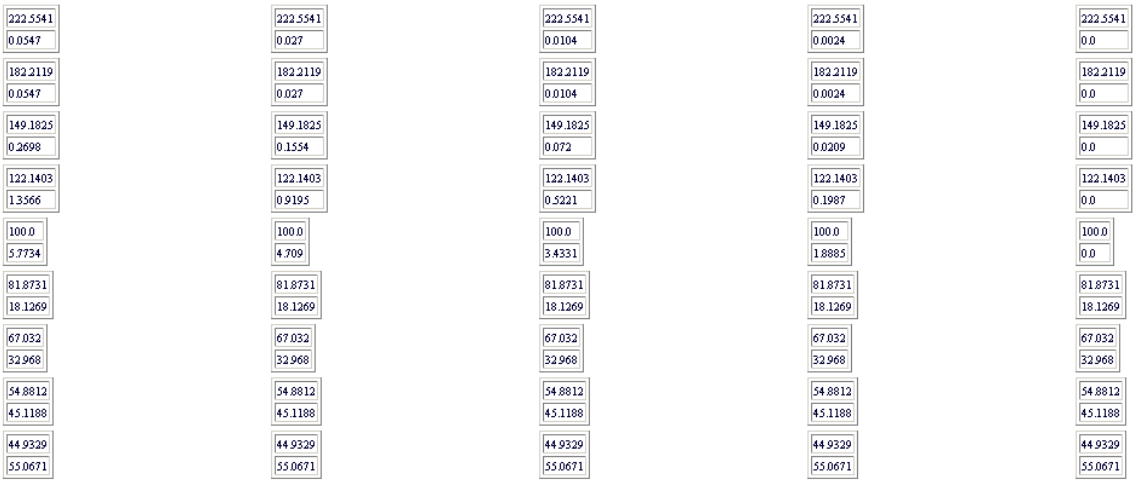

Figure 12.14: Pricing an American Put Option by Implicit Finite Difference Model (JSP)

12.6.3.5 Pricing an American Put Option by Crank-Nicolson Finite Difference Model

Figure 12.15 gives a numerical example similar to the implicit finite difference model. The calculations are virtually identical to those for the implicit finite difference method.

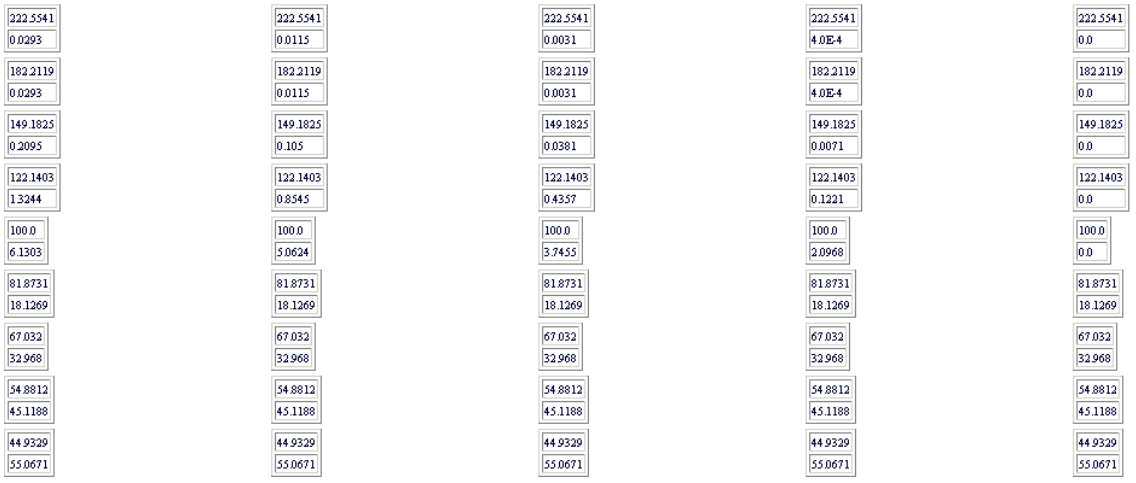

Figure 12.15: Pricing an American Put Option by Crank-Nicolson Finite Difference Model (JSP)

12.6.4 Implied Trinomial Tree Model

12.6.4.1 Pricing Implied Trinomial Tree State Prices and Transition Probabilities

Pricing state prices with one-year maturity and a current asset price of 100 and volatility of 20 per cent. The space step is chosen to be 0.2524 so that the explicit finite difference method stability condition is satisfied. The continuously compounded interest rate is assumed to be 4 per cent per annum and the trinomial tree has four time steps -

i.e. $T = 1$, $S = 100$, $r = 0.04$, $N = 4$, $\Delta x = 0.2524$.

Figure 12.16 illustrates the numerical results, where nodes in the tree are represented by the boxes in which the upper left value is the state price and the lower left value is the asset price. The right hand side values represent the transition probabilities.

Pre-computing the constants:

$$\Delta t = \frac{T}{N} = \frac{1}{4} = 0.25$$
$$edx = e^{\Delta x} = e^{0.2524} = 1.2871$$
$$infl = e^{r \times \Delta t} = e^{0.04 \times 0.25} = 1.0101$$

Computing the asset prices at maturity:

At node (4,-4)

$$S_{4,-4} = S \times e^{-N \times \Delta x} = 100 \times e^{-4 \times 0.2524} = 36.4365$$

At node (4,-3)

$$S_{4,-3} = S_{4,-4} \times edx = 36.4365 \times 1.2871 = 46.8978$$

Computing the state prices for each time step beginning with the upper half of the tree and then for the lower half of the tree:

At node (3,3)

$$p_{u,3,3} = infl \times \frac{Q_{4,4}}{Q_{3,3}} = 1.0101 \times \frac{0.000018}{0.0004} = 0.0456$$
$$p_{m,3,3} = \left(infl \times S_{3,3} - S_{3,2} - p_{u,3,3}\left(S_{3,4} - S_{3,2}\right)\right) / \left(S_{3,3} - S_{3,2}\right) =$$
$$\left(1.0101 \times 213.2297 - 165.6654 - 0.0456\left(274.4503 - 213.2297\right)\right) / \left(213.2297 - 165.6654\right) = 0.9407$$
$$p_{d,3,3} = 1 - p_{m,3,3} - p_{u,3,3} = 1 - 0.9407 - 0.0456 = 0.0137$$

At node (3,-1)

$$p_{d,3,-1} = \left(infl \times Q_{4,-2} - p_{u,3,-3} \times Q_{3,-3} - p_{m,3,-2} \times Q_{3,-2}\right) / Q_{3,-1} =$$
$$\left(1.0101 \times 0.0298 - 0.0627 \times 0.0003 - 0.8821 \times 0.0182\right) / 0.2101 = 0.0666$$
$$p_{m,3,-1} = \left(infl \times S_{3,-1} - S_{3,0} - p_{d,3,-1} \times \left(S_{3,-2} - S_{3,0}\right)\right) / \left(S_{3,-1} - S_{3,0}\right) =$$
$$\left(1.0101 \times 77.6934\text{-}100.00\text{-}0.0666\left(60.3626\text{-}100.00\right)\right) / \left(77.6934\text{-}100.00\right) = 0.8467$$
$$p_{u,3,-1} = 1 - p_{m,3,-1} - p_{d,3,-1} = 1 - 0.8467 - 0.0666 = 0.0867$$

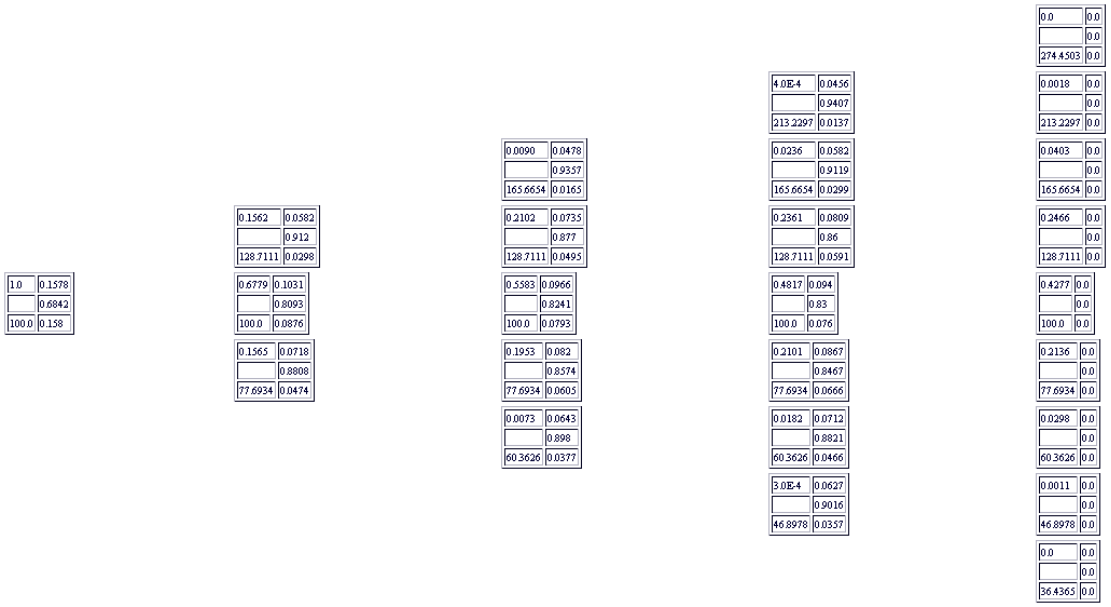

Figure 12.16: Pricing Implied Trinomial Tree State Prices and Transition Probabilities (JSP)

12.6.4.2 Pricing an American Down-and-Out Call Option by Implied Trinomial Tree Model

Pricing an at-the-money American down-and-out call option with one-year maturity and a current asset price of 100 and volatility of 20 per cent. The barrier is set at 110, the continuously compounded interest rate is assumed to be 4 per cent per annum. The implied trinomial tree is similar to the example above with four time steps and a space step of 0.2524. -

i.e. $K = 100$, $T = 1$, $S = 100$, $r = 0.04$, $N = 4$, $\sigma = 0.20$, $H = 110$, $X_{rebate} = 1$, $\Delta x = 0.2524$.

Figure 12.17 illustrates the numerical results, where nodes in the tree are represented by the boxes in which the upper left value is the state price and the lower left value is the option price. The right hand side values represent the transition probabilities.

Pre-computing the constants:

$$\Delta t = \frac{T}{N} = \frac{1}{4} = 0.25$$
$$disc = e^{-r \times \Delta t} = 0.99$$

Computing the option values at maturity:

At node (4,-1) $S_{4,-1} < H$, 77.6934 < 110 and therefore

$$C_{4,-1} = X_{rebate} = 1$$

At node (4,1) $S_{4,1} > H$, 128.7111 > 110 and therefore

$$C_{4,1} = \max(0, S_{4,1} - K) = \max(0, 128.7111 - 100) = 28.7111$$

Performing discounted expectations back through the tree and applying the barrier condition:

At node (1,1) $S_{1,1} > H$, 128.7111 > 100 and therefore

$$C_{1,1} = \max(disc \times (p_{u,1,1} \times C_{2,2} + p_{m,1,1} \times C_{2,1} + p_{d,1,1} \times C_{2,0}), S_{1,1} - K) =$$
$$\max(0.99(0.0582 \times 67.6465 + 0.912 \times 30.7422 + 0.0298 \times 1), 128.7111 - 100) = 31.6837$$

At node (0,0) - today - $S_{0,0} < H$ and therefore

$$C_{0,0} = X_{rebate} = 1$$

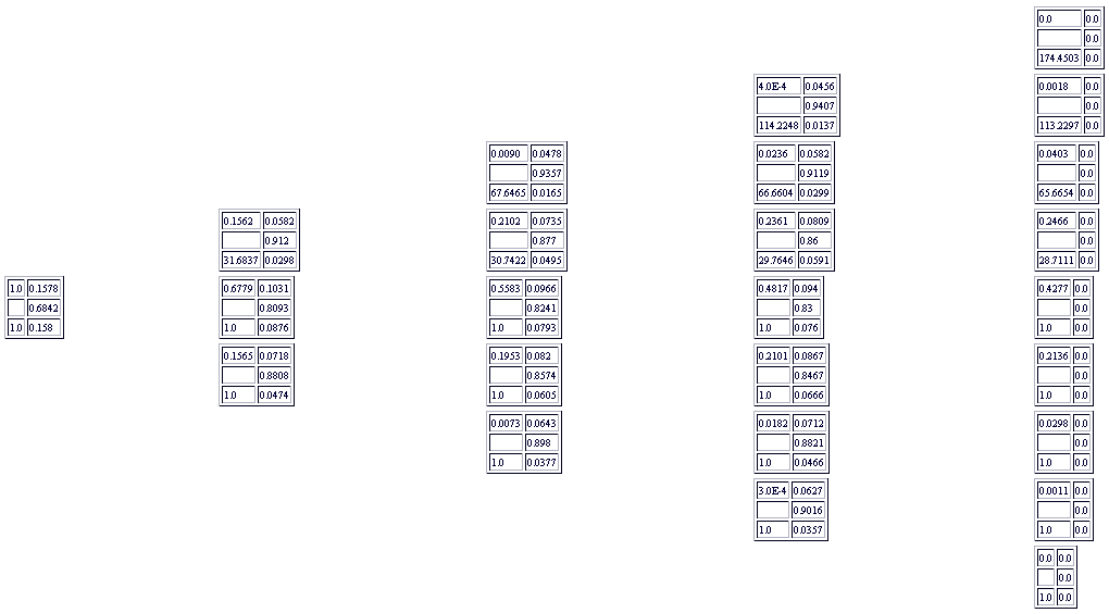

Figure 12.17: Pricing an American Down-and-Out Call Option by Implied Trinomial Tree Model (JSP)

REFERENCES

[1] Burton Gordon Malkiel and Richard E. Quandt, *Strategies and Rational Decisions in the Securities Options Market*, MIT Press, 1969.

[2] Herbert Filer, *Understanding Put and Call Options*, John Magee, Springfield, 1959.

[3] Joseph S. Davis, *Essays in the Earlier History of American Corporations,* Harvard University Press, Cambridge, 1917.

[4] Wikipedia, the free encyclopedia, *Underlying – Wikipedia, the free encyclopedia*, http://en.wikipedia.org/wiki/Underlying, 2008.

[5] Wikipedia, the free encyclopedia, *Asset – Wikipedia, the free encyclopedia*, http://en.wikipedia.org/wiki/Asset, 2008.

[6] Wikipedia, the free encyclopedia, *Option (finance) – Wikipedia, the free encyclopedia,* http://en.wikipedia.org/wiki/Option_(finance), 2008.

[7] Paul Wilmott, Sam Howison and Jeff Dewynne, *The Mathematics of Financial Derivatives: A Student Introduction,* Cambridge University Press, 1995.

[8] John Hull, *Options, Futures, and Other Derivatives*, Prentice Hall, 2006.

[9] Les Clewlow and Chris Strickland, *Implementing Derivatives Models*, Wiley, 1998.

[10] J. Cox, S. Ross and M. Rubinstein, *Option Pricing: A Simplified Approach,* Journal of Financial Economics 7, 1979, 229-264.

[11] R. Rendleman and B. Bartter, *Two State Option Pricing,* Journal of Finance 34, 1979, 1092-1110.

[12] John Hull and A. White, *Valuing Derivative Securities Using the Explicit Finite Difference Approach,* Journal of Financial and Quantitative Analysis 25, 1990, 87-100.

[13] M. J. Brennan and E. S. Schwartz, *Finite Difference Methods and Jump Processes Arising in the Pricing of Contingent Claims: A Synthesis,* Journal of Financial and Quantitative Analysis 13, 1978, 462-474.

[14] Masters 'O' Equity Asset Management, *Barrier Options by OptionTradingpedia.com*, http://www.optiontradingpedia.com/barrier_options.htm, 2008.

[15] Masters 'O' Equity Asset Management, *LookBack Options by OptionTradingpedia.com,* http://www.optiontradingpedia.com/lookback_options.htm, 2008.

[16] L. Ananthamurthy, *Introduction to Web Services,* http://www.webservices.org/index.php/article/articlestatic/75, 2003.

[17] David A. Chappell and Tyler Jewell, *Java Web Services: Using Java in Service-Oriented Architectures,* O'Reilly, 2002.

[18] World Wide Web Consortium, *Web Services Glossary*, http://www.w3.org/TR/ws-gloss/, 2004.

[19] K. Gottschalk, S. Graham, H. Kreger and J. Snell, *Introduction to Web Services Architecture,* http://www.research.ibm.com/journal/sj/412/gottschalk.pdf, 2002.

[20] Diana Reichardt, *A Field Guide to Services on Demand*, http://www.sun.com/software/whitepapers/webservices/wp-fieldguide.pdf, 2001.

[21] IBM Software Group, *Web Services Architecture Overview,* http://www.ibm.com/developerworks/web/library/w-ovr/, 2000.

[22] Gustavo Alonso, Fabio Casati, Harumi Kuno and Vijay Machiraju, *Web Services: Concepts, Architectures and Applications*, Springer, 2004.

[23] World Wide Web Consortium, *SOAP Specification,* http://www.w3.org/TR/soap, 2000.

[24] W3Schools by Refsnes Data, *SOAP Example,* http://www.w3schools.com/soap/soap_example.asp, 2008.

[25] XML Org, *Focus Areas XML.org focus area community,* http://www.xml.org/, 2008.

[26] World Wide Web Consortium, *Web Service Definition Language (WSDL)*, http://www.w3.org/TR/wsdl, 2001.

[27] W3Schools by Refsnes Data, *WSDL Syntax*, http://www.w3schools.com/wsdl/wsdl_syntax.asp, 2008.

[28] Peter Brittenham, *Web Services Development Concepts*, IBM Software Group, 2001.

[29] Kim Topley, *Java Web Services in a Nutshell,* O'Reilly, 2003.

[30] Sun Microsystems, *Java 2 Platform, Enterprise Edition (J2EE) 1.4*, http://java.sun.com/j2ee/1.4/, 2006.

[31] Sun Microsystems, *Java Enterprise Edition Web Services Technologies*, http://java.sun.com/javaee/technologies/webservices/index.jsp, 2006.

[32] Sun Microsystems, *Building Web Services with JAX-RPC*, http://developers.sun.com/appserver/reference/techart/jaxrpc/index.html, 2006.

[33] Sun Microsystems, *Java Web Services Developer Pack 1.5*, http://java.sun.com/webservices/docs/1.5/index.html, 2004.

[34] Sun Microsystems, *Using Web Services Effectively*, http://java.sun.com/blueprints/webservices/using/webservbp.html, 2002.

CURRICULUM VITAE

Alexander HOPFGARTNER
Wienergasse 24/7B
2380 Perchtoldsdorf
Austria

Personal Data

Date of Birth:	6th of July 1977
Nationality:	Austria
Family Status:	Single
Children:	None

School Education

1983-1987	Elementary School of Perchtoldsdorf
1987-1991	Secondary High School of Perchtoldsdorf
1991-1996	Technical Academy of Mödling (Management Engineering)

Academic Education

1996-2008	Study of Business Informatics University of Vienna - Faculty of Computer Science