ABSTRACT

As the Internet permeates through the boundaries of personal information management and redefines the way we maintain our day-to-day schedules, the need surfaces for an evolution of the age-old calendaring systems in wide use nowadays. Not only do such systems fail to place the end-user at the center of his/her experience, but are also designed with no inherent capability for aggregating events from different sources of interest to the user nor automating the management and tracking of such events. This thesis offers an innovative approach to calendaring and event management aimed at enhancing the end-user’s centralized scheduling experience while furnishing systems with the automated means to effectively publishing, managing, and retrieving events in a distributed environment like the World Wide Web.

The full-fledged calendaring solution specified and developed herein is implemented on a service-oriented 4-tier architecture enabling the seamless integration of a number of front-end clients and devices targeting users of different computing environments and calendaring needs. All the more, the advent of Web Services has made this interoperability highly achievable extending the system’s reach and range to any web-enabled device or medium.

The thesis concludes with an in-depth study of the market opportunity and business potential for what will be referred to hereinafter as calendaring and event management outsourcing. The tangible benefits of delegating all scheduling-related tasks to a specialized commodity Web Services provider are also spelled out in the light of the emerging trend to view software as a service rather than a product.

Thesis Supervisor: John R. Williams
Title: Associate Professor, Department of Civil & Environmental Engineering
# Table of Contents

`Table of Contents` .................................................................................................................. 3

`Table of Figures, Tables, and Listings` ...................................................................................... 8

`List of Acronyms` ....................................................................................................................... 11

`Acknowledgements` .................................................................................................................... 13

## Chapter 1  Introduction ........................................................................................................... 14

1.1 Prelude .................................................................................................................................. 14

1.2 Motivation ............................................................................................................................... 15

1.3 Thesis Organization and Guidelines ..................................................................................... 16

## Chapter 2  An Initiative Begins to Shape ............................................................................... 19

2.1 Justifying a Need ..................................................................................................................... 19

2.2 System-Specific Terminology ............................................................................................... 23

2.2.1 Codename and Logo ........................................................................................................ 23

2.2.2 Nomenclature .................................................................................................................. 24

2.3 End Objectives ....................................................................................................................... 25

2.3.1 Centralization .................................................................................................................. 25

2.3.2 Abstraction ...................................................................................................................... 25

2.3.3 Automation ..................................................................................................................... 25

2.3.4 Serviceability .................................................................................................................. 26

2.4 Architectural Overview ......................................................................................................... 26

2.4.1 System Components ..................................................................................................... 26

2.4.2 Alignment with Service-Oriented Architecture ............................................................ 29

2.4.3 Implementation Technologies ....................................................................................... 29

2.5 System Processes .................................................................................................................. 30

## Chapter 3  Primary Web Interface ......................................................................................... 31

3.1 Overview ............................................................................................................................... 31

3.2 Main Functionality ............................................................................................................... 31
### 3.2 Calendar Component

#### 3.2.1 Calendar Component

- Events Management
- Settings and Preferences

#### 3.2.2 Events Management

- Calendar Component

#### 3.2.3 Settings and Preferences

- Calendar Component

### 3.3 The Advent of ASP.NET

#### 3.3.1 Event-Driven Model

- Calendar Component

#### 3.3.2 Server-Side Processing

- Calendar Component

#### 3.3.3 Separation of Code from Content

- Calendar Component

#### 3.3.4 Server Controls

- Calendar Component

#### 3.3.5 User Controls

- Calendar Component

### 3.4 Design and Implementation

#### 3.4.1 Data Access

- Calendar Component

#### 3.4.2 Page Structure

- Calendar Component

#### 3.4.3 Pages and Controls

- Calendar Component

### Chapter 4 Client for Office Xp

#### 4.1 Overview

- Calendar Component

#### 4.2 COM and .NET Interoperability Difficulties

- Calendar Component

##### 4.2.1 Detecting Host Application

- Calendar Component

##### 4.2.2 Event Handling in Host Application

- Calendar Component

##### 4.2.3 Other COM-Related Issues

- Calendar Component

### Chapter 5 Tagging Schema and Message Handling

#### 5.1 Overview

- Calendar Component

#### 5.2 Tag Specification

- Calendar Component

##### 5.2.1 Flexibility

- Calendar Component

##### 5.2.2 Tag Description

- Calendar Component

##### 5.2.3 Example

- Calendar Component

### 5.3 Tag Handling

- Calendar Component

##### 5.3.1 Regular Expressions

- Calendar Component

##### 5.3.2 Process Flow

- Calendar Component

##### 5.3.3 Handler Logic

- Calendar Component
### 9.1 Evaluating miServices

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1.1 Tangible Benefits</td>
<td>113</td>
</tr>
<tr>
<td>9.1.2 Assessment Framework</td>
<td>114</td>
</tr>
<tr>
<td>9.1.3 Comparing Current Calendaring Technologies</td>
<td>115</td>
</tr>
</tbody>
</table>

### 9.2 What happened to .NET My Services?

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.2.1 Initiative Introduced</td>
<td>117</td>
</tr>
<tr>
<td>9.2.2 Initiative Remodeled</td>
<td>117</td>
</tr>
</tbody>
</table>

### 9.3 Viable e-Business Model

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.3.1 Model Semantics</td>
<td>118</td>
</tr>
<tr>
<td>9.3.2 Ownership of Customer Data and Relationship</td>
<td>120</td>
</tr>
<tr>
<td>9.3.3 Market Segmentation</td>
<td>120</td>
</tr>
</tbody>
</table>

### 9.4 Revenue Streams and Pricing

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.4.1 Web Services Implementation</td>
<td>122</td>
</tr>
<tr>
<td>9.4.2 Licensing the System</td>
<td>123</td>
</tr>
<tr>
<td>9.4.3 Consulting and Integration</td>
<td>124</td>
</tr>
<tr>
<td>9.4.4 Premium End-Usage</td>
<td>125</td>
</tr>
<tr>
<td>9.4.5 SOR Derived from Information Ownership</td>
<td>125</td>
</tr>
</tbody>
</table>

### References

Page 126

### Appendices

Page 128

#### Appendix I: Key System Process Diagrams

<table>
<thead>
<tr>
<th>Diagram</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP1: Event Dispatching</td>
<td>128</td>
</tr>
<tr>
<td>SP2: miLink Generation</td>
<td>128</td>
</tr>
<tr>
<td>SP3: Event Adding via miLink</td>
<td>128</td>
</tr>
<tr>
<td>SP4: miTags Handling</td>
<td>128</td>
</tr>
<tr>
<td>SP5: miEvents Management</td>
<td>128</td>
</tr>
<tr>
<td>SP6: New Event via miServices</td>
<td>128</td>
</tr>
</tbody>
</table>

#### Appendix II: miTerms (miSchedule Nomenclature)

Page 135

#### Appendix III: miFAQs (Frequently Asked Questions)

Page 138

#### Appendix IV: miPortal Screenshots

Page 141

#### Appendix V: Full miTags Description

Page 151

#### Appendix VI: `GetEventInstance` Method from miClasses

Page 153

#### Appendix VII: Listing of .NET My Services

Page 155
Appendix VIII: Executive Summary on ASP Industry ...........................................156

Defining Characteristics ..................................................................................156
Facts and Figures ..............................................................................................156
ASP Evaluation Metrics ...................................................................................156
Top-Line Benefits ..............................................................................................156
Bottom-Line Benefits .......................................................................................157
Market Estimations ...........................................................................................157
Revenue Forecast ..............................................................................................158
Table of Figures, Tables, and Listings

- Figure 1 Current Student-Learning Management System Interaction at MIT ........................................ 20
- Figure 2 Desired Student-Learning Management System Interaction at MIT ........................................ 23
- Figure 3 miSchedule Logo .................................................................................................................. 24
- Figure 4 miSchedule's 4-Tier Service-Oriented Architecture ............................................................. 28
- Figure 5 Master HTML Template with Modular User Controls .......................................................... 38
- Figure 6 Instant miLink Notifier in MS Outlook .................................................................................. 45
- Figure 7 miAddin Insertions into FrontPage's Context Menu .............................................................. 50
- Figure 8 miPassport Tab of miAddin .................................................................................................. 54
- Figure 9 miEvents (Create...) Tab of miAddin .................................................................................... 54
- Figure 10 miSettings Tab of miAddin .................................................................................................. 54
- Figure 11 Other... Tab of miAddin ...................................................................................................... 54
- Figure 12 frmConnecting While User Imports miCalendar ............................................................... 55
- Figure 13 frmDefaultCalendar Showing Detected Outlook Calendar Folders ................................... 55
- Figure 14 frmMiLink Displaying Retrieved miLink ........................................................................... 55
- Figure 15 frmSend Used to Select miEvent Recipients ...................................................................... 56
- Figure 16 miAddin Help File ............................................................................................................... 57
- Figure 17 Service Controller Showing miMail Service ........................................................................ 70
- Figure 18 'miTag General' Tab of the Mail Handler User Interface .................................................... 71
- Figure 19 'miTag Activity' Tab of the Mail Handler User Interface ..................................................... 73
- Figure 20 'Outgoing' Tab of the Mail Handler User Interface ................................................................ 73
- Figure 21 miSchedule's Back-End Repository and Logic .................................................................. 75
- Figure 22 Data Model Structure and Entity Relationships .................................................................. 77
- Figure 23 The 3 Basic Operations [22] ............................................................................................... 100
- Figure 24 Potential Web Service Providers [3] ................................................................................... 102
- Figure 25 miServices Software Development Kit (SDK) .................................................................. 107
- Figure 26 Distinctive Aspects of Different Calendaring Technologies .............................................. 116
- Figure 27 miSchedule's e-Business Model ......................................................................................... 119
- Figure 28 Basic Portal Taxonomy ........................................................................................................ 121
- Figure 29 SP1: Event Dispatching ....................................................................................................... 129
- Figure 30 SP2: Event miLink Generation ............................................................................................ 130
- Figure 31 SP3: Event Adding via miLink ............................................................................................. 131
- Figure 32 SP4: miTags Handling ......................................................................................................... 132
- Figure 33 SP5: miEvents Management ............................................................................................... 133
Figure 34 SP6: New Event via miServices ................................................................................ 134
Figure 35 Event View After Being Postponed............................................................................ 141
Figure 36 miSchedule Main Page .......................................................................................... 142
Figure 37 Event View (by event owner).................................................................................... 143
Figure 38 miCalendar Monthly View ...................................................................................... 144
Figure 39 miCalendar Weekly View ....................................................................................... 145
Figure 40 miCalendar Daily View .......................................................................................... 146
Figure 41 miCalendar List View .............................................................................................. 147
Figure 42 New Event View ........................................................................................................ 148
Figure 43 miSettings View ........................................................................................................ 149
Figure 44 miLists Management View ....................................................................................... 150
Figure 45 Worldwide ASP Revenue Forecast ......................................................................... 158

Table 1 Partial Listing of miPortal Pages with Descriptions.................................................... 40
Table 2 Partial Listing of miPortal Controls with Descriptions................................................... 41
Table 3 miTags: Aliases, Descriptions, Formats, and Qualifiers.............................................. 65
Table 4 miServices Public Instance Methods............................................................................. 109
Table 5 Web Service Opportunities Must Meet Three Criteria [12]............................................ 114
Table 6 Benefits of Using an Application Service Provider........................................................ 157
Table 7 Market Estimations for the ASP Industry...................................................................... 157

Listing 1 Registering a User Control on an ASP.NET Webpage................................................ 37
Listing 2 Embedding a User Control on an ASP.NET Webpage................................................ 37
Listing 3 Implementing the IDTExtensibility2 to Integrate with Host Application .................. 44
Listing 4 Handling the NewMail Event in Outlook.................................................................. 46
Listing 5 Libraries Referenced by miAddin’s Front-End (right)............................................... 47
Listing 6 Structure of miAddin’s Front-End Solution (left)....................................................... 47
Listing 7 Adapting miAddin Interface in FrontPage Host Application....................................... 49
Listing 8 Connect Class Members ........................................................................................... 51
Listing 9 Settings Class Fields ................................................................................................. 54
Listing 10 miAddin Server-Side Classes and Web Services...................................................... 58
Listing 11 miAddinEvent vs. miAddinImportEvent in miAddin Services.................................. 59
Listing 12 Stripped-Down Version of Core miUser Class......................................................... 59
Listing 13 Synchronous and Asynchronous miAddin Web Methods......................................... 60
Listing 14 Dependencies Included in miAddin Deployment Package......................................... 61
Listing 15 miTagged Email Example ....................................................................................... 66
Listing 16 Sample Regular Expression Used to Validate TimeZone........................................... 67
List of Acronyms

API  Application Programming Interface
ASP  Application Service Provider or Active Server Pages
B2B  Business To Business
B2C  Business To Customer
B2E  Business To Employee
CGI  Common Gateway Interface
CLR  Common Language Runtime
COM  Component Object Model
CORBA  Common Object Request Broker Architecture
CRM  Customer Relationship Management
CSS  Cascading Style Sheets
CSV  Comma Separated Value
DCOM  Distributed Component Object Model
DLL  Dynamic Link Library
DNS  Domain Name Registry
ERP  Enterprise Resource Planning
HR  Human Resources
HTML HyperText Markup Language
HTTP HyperText Transfer Protocol
IDC  International Data Corporation
ISV  Independent Software Vendor
ITAA  Information Technology Association of America
JIT  Just-In-Time
MIT  Massachusetts Institute of Technology
OLE  Object Linking and Embedding
PDA  Personal Digital Assistant
PIM  Personal Information Manager
POP  Post Office Protocol
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAD</td>
<td>Rapid Application Development</td>
</tr>
<tr>
<td>ROI</td>
<td>Return on Investment</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SLA</td>
<td>Service Level Agreement</td>
</tr>
<tr>
<td>SME</td>
<td>Small and Medium-Sized Enterprises</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architecture</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SOI</td>
<td>Service Oriented Interface</td>
</tr>
<tr>
<td>SOR</td>
<td>Sources of Revenue</td>
</tr>
<tr>
<td>SOS</td>
<td>Service Oriented Software</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>TCO</td>
<td>Total Cost of Ownership</td>
</tr>
<tr>
<td>UDDI</td>
<td>Universal Description, Discovery, and Identification</td>
</tr>
<tr>
<td>VSP</td>
<td>Vertical Service Provider</td>
</tr>
<tr>
<td>WAP</td>
<td>Wireless Access Protocol</td>
</tr>
<tr>
<td>WS</td>
<td>Web Services</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Services Description Language</td>
</tr>
<tr>
<td>WSP</td>
<td>Web Service Provider</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
</tbody>
</table>
Acknowledgements

First and foremost, I am more than grateful for the hard work, dedication, and patience of Mazen Manasseh, Mario Harik, and Ephraim Tekle without whom this thesis and underlying project could not have come to life. I am indebted to you all for a wonderful academic experience this year capped with such a gratifying achievement.

Many thanks go to Professor George Kocur for his invaluable advice and recommendations on best software development practices throughout the different stages of this project. I would also like to thank Professor John Williams for his support, namely in kindly satisfying our endless software and hardware requirements. My sincerest gratitude also goes to Hasan Osman for his insightful comments, Charles Assaf for his unremitting encouragement, and Sarah Summers for her boundless care and forbearance during the course of writing this thesis.

No acknowledgement could be wholesome without extending my utmost gratitude to Taleb, Najla'a, Bashar, Anas, and Dana, my blessed family. Dad, words are too little to thank you for your sea of insight, encouragement, and confidence in my capacities. Mom, your love and care remain the excess of an impulse behind all that I am and all that I do.

Loai
Chapter 1  Introduction

1.1 Prelude

Studies show that PIMs (Personal Information Managers) are becoming as ubiquitous as email was several years ago. In fact, over 90% of the 55 million U.S. Internet users (connected from work) use Microsoft Outlook or Lotus Notes (eMarketer, August 2000). ARC Group (December 2000) predicts PIM applications will continue to be the most used application beyond 2003. The calendaring and event management solution presented in this thesis and developed in the Department of Civil and Environmental Engineering Department at MIT will enable businesses to target their customers in a manner that is consistent with their organizational habits by reaching them right at their one portable central PIM.

Most people today keep track of as many calendars and event postings as they have sources of events. Consider a certain night club that posts events on its website calendar for its patrons to check as they browse along, just as many other clubs are obviously doing. By implication, this means than an avid clubber would have to track as many calendars as there are clubs that are of interest to him/her. This is not to mention the calendar imposed by the company for which he/she works, other nearing recreational events like a movie release, the time constrains imposed by his own personal schedule, etc... And ironically, the more socially/professionally active or involved one is, the more distinct calendars he/she has to keep track of.

Based on a survey conducted at MIT (miSchedule Electronic Survey, January 2003), more than half the emails received by the community (students, faculty, and staff) on a given day are meant to promote some form of event taking place on- or off-Campus that is academic- or non academic-related; be that a seminar,
free food offering, or even a walkout. The unlikely scenario of a student ‘saving’ an event would involve either keeping it in memory, writing it on a paper, or manually transcribing it to his/her own favorite PIM, all of which are inherently not the most optimal ways of tracking events. This attests to the increasing ineffectiveness of email as a means of communicating events over the Internet.

Neither should people organize their lives around their inboxes, nor should they assume the unnecessary effort of migrating events from there to their personal calendars. Furthermore, as email spam exponentially increases, both end-users and businesses will require a more targeted and permission-based form of communicating events. Similarly, generic portals and Web sites, although ideally suited for aggregating content in a fashion customizable by their large user-base, they still fall short of proactive and semantically personalized event communication.

1.2 Motivation

This thesis and the primary concepts behind its underlying project did not emerge out of seeking an innovative area to research or problem to tackle, but out of an imposing need to overcome a lot of the angularity in maintaining our day-to-day schedules. We, all members of the team conceiving and developing this system, were facing the same scheduling-related problems throughout our campus lives at MIT: Calendar centralization was a critical issue; so was the automation of saving events described in text from emails and webpages without having to manually rewrite them in our favorite PIMs. It is out of such driving needs that the need to act came to be.

Our research and conceptualization of the system was primarily reinforced by the enabling computing technologies becoming widely prevalent nowadays, namely with the emergence of Web Services and Microsoft’s .NET platform. The latter constituted the Rapid Application Development (RAD) foundation on which most
system components were designed to seamlessly interact within the very short timeframe allocated to developing the system. Web Services served as the core integrating technology without which many facets of the system could not have been conceived some time ago.

Finally, urged to explore the business potential and real-life implementation feasibility of the conceived system, a comprehensive study of Web Services’ market trends and possible pricing models was conducted in the light of rethinking software as a service instead of a product. The result was a diversified e-business model aggregating a number of viable revenue streams from the different system components and offerings.

1.3 Thesis Organization and Guidelines

After this introductory chapter, Chapter 2 (p.19) spells out the need for the calendaring and event management initiative presented in this thesis and lays out its preliminary cornerstones and end objectives. It also introduces the system-specific terminology to be used throughout this document along with an overview of the adopted architecture.

Chapter 3 (p. 31), Chapter 4 (p.42), and Chapter 5 (p.62) introduce the different software clients constituting the front-end of the system. Each chapter starts with briefly presenting the intended medium, use, and functionality associated with the specific front-end client, before presenting the technologies used in its development. Design considerations and leverages or difficulties encountered implementing a certain technology are also highlighted.

The back-end of the system is presented in depth in Chapter 6 (p.75). This backbone is divided into a central data repository and a set of core processes serving all front-end clients and devices, both of which are discussed in-depth
along with the different technologies and design practices adopted in their development.

**Chapter 7** (p.90) provides the reader with the necessary background information on the emerging trend of conceiving software as a *service* rather than a *product*. It outlines the basic principles of the unique architecture required to enable servicing software. It then introduces Web Services technology from a practicable perspective and explores the huge leverage it brings to the aforementioned architecture. Potential Web Services markets and possible pricing models are also briefly discussed in this chapter.

With the background information in Chapter 7, the thesis shifts back in **Chapter 8** (p.105) to the Web Services component of the system under study and delineates the different offered services. A detailing of the actual Application Programming Interfaces provided follows, along with a discussion on the best message exchange practices adopted in designing the Web Services interfaces.

**Chapter 9** (p.113) finally takes the knowledge accrued in Chapters 8 and 9 to study the business potential for the calendaring and event management solution presented in this thesis. The different offerings and corresponding revenue streams are then extensively discussed and contrasted in the light of viable pricing models that can be adopted.

The **Appendices** (p.128) offer additional information that could be invaluable to readers interested in specific topics raised in this thesis. The material provided in the appendices ranges all the way from screenshots of front-end system clients to summary statistics on the Application Service Provider industry.

Given the diversity of technologies utilized in bringing this system to life, it is assumed that the reader is sufficiently informed about most of the technologies discussed herein (namely the Microsoft .NET platform) and would rely on
external sources for fundamentals and additional information. Technology background information available in this thesis is very concise, and is only meant to provide an introductory inlet to the system-specific implementation. The shift from generic information to specific design criteria, constraints, or leverage as applied to system development is directly made.

Finally, most source code listings have been slightly modified or abridged to give the reader a better contextual understanding of the listings. By this, focus is shifted to the intended source code sample as it relates to a given technology free from the complexity and cluttering introduced by actual listings incorporating full runtime considerations. Also, the font used for any syntactic terms and keywords introduced inline is varied as such.
2.1 Justifying a Need

It has been the trend rather than the exception to have online calendar/schedule services boasting of full personalization features, customizable themes, far-reaching alert/notification capabilities, etc... without offering users the true essence of individualization: the ability to seamlessly channel events from diverse sources into 'one' schedule revolving around the same recipient of those distinct events, and not around the source(s) of events it contains. It is in this sense, and this sense only, that user-centricity can be justifiably acclaimed.

As applied in the context of e-Learning, take for example the current interaction schema between a typical MIT student and the websites (any form of presence on the web) of 3 of his/her registered courses, A, B, and C:

- Student logs on to Course A at 'Stellar' (MIT's Learning Management System) using personal MIT digital certificate and checks out scheduled Course A-specific assignments, readings, and postings.
- Student then logs on to Course B at 'Course Portal' (another web-based course management site) using personal MIT email address and another password of his/her selection, and checks out scheduled Course B-specific assignments, readings, and postings.
- Student then logs on to Course C at 'SloanSpace' (another portal-based course management intranet for students of MIT Sloan School of Management) using personal MIT email address and a possibly third password of his/her selection, and checks out scheduled Course C-specific assignments, readings, and postings.
- Finally, retaining a mental/written record of all different course-related events and due dates, student reverts to his/her preferred PIM (Personal Information
Manager), online schedule, PDA, and/or piece of paper and hard codes those various events into one forcedly user-centric schedule. Figure 1 is an illustration of the above scenario:

![Figure 1: Current Student-Learning Management System Interaction at MIT](image)

Figure 1 Current Student-Learning Management System Interaction at MIT

Although each of the 3 'learning management' systems mentioned above provide a lot of functionality in terms of communicating course material to students and flexibility in terms of tracking course documents, postings, and events, they are still self-involved, in a sense that 'SloanSpace' would only provide a 'multi-course schedule combining' feature for registrants enrolled in a number of Sloan courses, which are required to, in turn, be registered with 'SloanSpace'. Most other systems don't even recognize that, and treat the same registrant for different courses hosted by this one system as a different entity with a one-to-one relationship with every course.

The need for miSchedule (see 2.2.1 Codename and Logo p. 23) arises with the need to get around the angularity described above, and make the service user the center of a schedule that ought to channel different events from a diversity of
sources. Before delving into how this has been achieved, I would like to describe another scenario that we would want miSchedule to cater for; this will also help justify the adopted architectural design considerations described later on (see 2.4 Architectural Overview p. 26):

- More than half the emails received by MIT students every day can be classified as 'mass email' or 'impersonal' (miSchedule Survey, January 2003), in a sense that it has been addressed to many recipients belonging to a class list, MIT club, sports group, etc... with the intent of advertising or bringing a certain event to the attention of those assumed to be concerned.
- A recipient of such promotional email would quickly skim through the highlighted event and, if interested, would go to his/her favorite PIM (Personal Information Manager) – perhaps Outlook or possibly a calendar on his/her wall – to manually add the event, and if uninterested, would simply disregard/delete the email.
- Alternatively, this user, along with any other, would appreciate the convenience of being able, with the click of a button, to autonomously log the event they just read (textually, in the body of the email) into their favorite 'one' schedule without having to revert to another environment and re-enter it in some special format.

At this point, the 2 scenarios described above ('student-course interaction' and 'manual vs. automatic email event logging') should help highlight the following characteristics of miSchedule (please proceed sequentially):

- Establishing such a facile scheduling system obviously requires that both the 'senders' and 'recipients' be recognized (at a minimal level) by miSchedule, and hence the necessity of having them registered beforehand.
- The 2 scenarios described above demonstrate both extremes of a user-set permission scheme describing his/her relationships with the event sources (a sender-recipient relationship). In the case of a student-course relationship,
the student (recipient) would be giving the professor/course (sender) 'full permission' to automatically log events into his/her calendar. On the other hand, a student receiving random emails with events from different sources would obviously want to force events to pass through him via email first, where he/she would choose whether to have events logged or not.

- It is obvious at this point that miSchedule is not a solution only targeting students and their many course schedules, but targeting individuals and their many/any schedules. That is, 'senders' need not be restricted to course websites (but can represent any 'sending' entity; be that a company with a conference to announce or an individual with a party to inform his/her friends about), nor do 'recipients' need be students (but can represent any 'receiving' entity that would like to view a diversity of events in a calendar-like format).

- miSchedule does not compete with or attempt to replace systems such as 'Stellar', 'Course Portal', etc... as it is only concerned with schedule/event-related postings of such systems, and would want to facilitate: (1) the user experience, by aggregating his/her 'Stellar' and 'Course Portal' events, and (2) the system (i.e. 'Stellar' in this case) experience, by storing and managing its events and event-related user preferences on its behalf. This would require that 'Stellar' and the likes implement miSchedule methods to programmatically 'push' their events to miSchedule. miSchedule would, in turn, reroute those events to the relevant recipients' online schedules hosted by miSchedule (in highly customizable views with a very ergonomic interface) on the web.

- Any miSchedule 'event' would bear basic attributes such as a 'unique identifier', 'sender', 'time sent', 'event class', 'event name', 'event description', 'optional link' to external coverage of event (to sender's site for instance), all of which would be defined by the 'sender' upon sending his/her event.

- In the case of the 'sender' being an individual communicating an event via email, he/she would be able to embed the event in a self-describing hyperlink (in miSchedule 'format' using miSchedule 'tools') to be contained in his/her outgoing email (using any web or offline mail client). The 'recipient' would
receive and read the email, and with a single click, be able to log the event into his/her miSchedule (and also to his favorite desktop PIM).

Figure 2 is an illustration of the previously described scenarios in the light of adopting miSchedule:

Figure 2 Desired Student-Learning Management System Interaction at MIT

2.2 System-Specific Terminology

2.2.1 Codename and Logo

I would like to draw your attention to the subtleties of the chosen 'miSchedule' codename and logo (below). Analogously, Microsoft's .NET My Services (formerly codenamed "HailStorm" — see 9.2 What happened to .NET My Services?, p.117) promoted the same essence of user-centricity, and hence the use of the keyword 'My' to emphasize individualization. The 'y' has been replaced with the internet 'i' denoting web portability, while retaining the same 'My'
enunciation. The logo, however, signifies evolution from the ancient sun clock, just as miSchedule is the eclectic evolution of age-old PIM-centered schedules.

![miSchedule Logo](image)

**Figure 3** miSchedule Logo

### 2.2.2 Nomenclature

Extensive use is made of 'miXXX' terms and keywords throughout this thesis to reference proprietary miSchedule components. Such names have been used to codename different components and clients internally, and for other reasons such as branding and brevity. For example, instead or referring to “the miSchedule Email Tagging Specification” in full-length, the term “miTags” is used. In this same way, “miServices” replaces “the miSchedule XML Web Services” and “miLink” replaces “the self-describing miSchedule hyperlink”, etc. Those terms are referred to as miTerms and have been checklisted for quick reference in *Appendix II: miTerms (miSchedule Nomenclature)* p.135.
2.3 End Objectives

2.3.1 Centralization

miSchedule would help centralize users’ multipurpose calendaring service(s) around them and not the service. Calendaring should evolve in a sense that people need not track/keep as many calendars as they have sources of events. This centralized miSchedule calendar will be referred to as a user’s miCalendar hereinafter, a web-based centralized repository of events aggregated from diverse sources and hosted by miSchedule, with its primary user interface available to users at miPortal.

2.3.2 Abstraction

A user should say he/she received an event and truly have received an event, not an email describing an event. A user should see an event posted on a webpage as he/she surfs the internet, and not text about an event. This semantic notion of an event with a certain level of integrity is embodied by miEvent, the miSchedule definition of an event along with the comprehensive set of rules, process, and attributes associated with handling it. miEvents will be communicated through many channels such as miPortal (miSchedule’s primary web interface), miTags (a text-based tagging schema for defining events in emails), miAddin (a desktop client deeply integrated with Office Xp), and miServices (a set of Application Programming Interfaces that can be consumed by system clients to communicate and manage miEvents).

2.3.3 Automation

When the user receives an event in an email or comes across another on a webpage, he/she should be able to instantly save it without rewritings it somewhere else. This event logging process is automated through miLink, a special miSchedule-generated hyperlink constituting a unique reference to a certain miEvent. When clicked, the underlying event is directly saved to the
user's central miCalendar. Another facet of automation is seamlessly porting miEvents and miCalendar to other environments such as a desktop PIM or to a PDA. This is achieved via miAddin that would enable miCalendar importing into MS Outlook (and its derivatives, like Pocket Outlook and PDA synchronization), instant notification of incoming miLinks, integration with the Outlook Address Book when sending miEvents, inserting miLinks into webpages with the speed hyperlinks are inserted in FrontPage, among many other automation and integration features.

2.3.4 Serviceability

miSchedule would also package all the aforementioned system features into a tempting services to subscribing system clients. Through miServices, course portals, corporate websites, and any other system hosting a calendar or regularly addressing its patrons with upcoming events can outsource its calendaring and event management duties to miSchedule. miSchedule would, in turn, provide the necessary functionality empowering the miServices consumer to send events, retrieve events, register customers, generate miLinks, etc... via miSchedule. In a sense, miSchedule would serve as a utility provider specializing in calendaring and event management. miSchedule would analogously be to events and scheduling what Passport .NET is to passwords and authentication.

2.4 Architectural Overview

2.4.1 System Components

In tersest terms, miSchedule consists of a back-end database of registered events, senders, and recipients (along with their respective preferences and configurational attributes) and a front-end collection of diverse clients/means to communicate (send, receive, view, and configure) events via miSchedule. The system uses the highly flexible Service-Oriented Architecture (SOA) paradigm (see in 7.2.2 An Enabling Architecture, p.92) that is leveraged by loosely-coupled
Web Services of private interfaces for internal miSchedule client/server communications and public APIs (Application Programming Interface) for 3rd-party clients and web-based systems to communicate their events with miSchedule in an automated peer-to-peer fashion.

The different clients targeting different users (computing environments and needs) developed by the miSchedule Team and described in this document are: (1) the miSchedule portal/website (miPortal), (2) an Office Xp COM Add-in (miAddin), (3) a miSchedule email tagging schema (miTags), and (4) a collection of miSchedule Web Services capacitating web-based systems to send/retrieve events autonomously to/from their users' centralized miSchedule calendars (miServices). Please note that the terms 'schedule' and 'calendar' are used interchangeably.

Below is a schematic of the system architecture illustrating into which layer each of the different miSchedule clients interconnects:
miTags for text-based transactions

miPortal and miLink

System client using miServices

miAddin for Office Xp

Presentation & device or channel independence

User session & data manipulation

Business processes

Shared databases & resources

Server-Side Processing

miServer (IIS)

Core Classes

Data Adapters

Core Database

Figure 4: miSchedule's 4-Tier Service-Oriented Architecture
2.4.2 Alignment with Service-Oriented Architecture

Also note the 4-tier architecture in Figure 4 above, an evolution from the typical 3-tier model. Given the wide range of client applications, devices, and access channels miSchedule is to support, the 4-tier architecture proves to be more service-oriented [14]. In this architecture, the classical Presentation tier has been divided into User and Workspace tiers: the User tier is responsible for presentation and device independence, for example supporting both a regular web browser and a WAP-enabled version of miPortal (assuming miSchedule would later on directly support handhelds and/or 3G phones). The Workspace tier, on the other hand, is now solely responsible for maintaining user session and session-related data and delegating requests to the Logic tier. For example, both miServices and miAddin users interact with miSchedule via Web Services residing in this tier; however, the former uses the publicly exposed miServices, whereas miAddin uses another set of shielded Web Services for all client/server communication. The User and Workspace tiers together support a single user, and there would be as many instances as there are users interacting with the system at a given point in time. The Logic and Resource tiers remain one instance at any stage supporting all users/clients.

2.4.3 Implementation Technologies

At the very core exists the miSchedule database (Chapter 6 Back-End Repository and Logic, p.75) implemented in Microsoft SQL Server 2000. The database is wrapped in a set of core classes (miClasses) written in C#. To serve the front-end, 2 XML Web Services were developed: the public miServices and the private miAddinServices managing all client-server communication with miAddin (Chapter 4 Client for Office Xp, p.42). miPortal plugs right into the ASP.NET layer (Chapter 3 Primary Web Interface, p.31) which uses miClasses and the underlying data adapters to perform all necessary database calls/queries. Incoming miTagged emails, on the other hand, are directly handled
at the level of miClasses via a Windows Services listening to a POP3 server (Chapter 5 Tagging Schema and Message Handling, p.62).

2.5 System Processes

In Appendix I: Key System Process Diagrams (p.128), the diagrams provided are meant to graphically describe key internal processes that take place as different clients communicate with miServer (depending on client used, intended recipient(s), and nature of event). Those processes are distinctly described and will be profusely referred to throughout the rest of the thesis, which will proceed hereafter with detailing the design considerations and overall development of the different miSchedule clients and core.
### 3.1 Overview

miPortal is miSchedule's primary user interface offering rich functionality and extensive control to miSchedule users over their events. Through miPortal, users can create, view, send, receive, and manage events to the fullest. miSettings, miNotifications, and miPermissions, and miLists are also fully configurable from miPortal.

miPortal uses miSchedule's core classes to implement all process logic behind every user action. Only interface logic is implemented in the code behind each web form. miPortal was developed using ASP.NET and makes extensive use of this new programming model through code reusability and the elaborate consumption of user controls. miPortal consists of 31 .aspx pages and 21 user controls (listed and described in 3.4.3 Pages and Controls, p.38) that account for around 5600 lines of C# code, 10000 lines of HTML and 800 lines of JavaScript.

Before detailing the development of miPortal, a concise walkthrough of the portal is provided next (accompanied with screenshots in Appendix IV: miPortal Screenshots, p.141) to cover the main miSchedule functionality and features provided on the web.

### 3.2 Main Functionality

#### 3.2.1 Calendar Component

miCalendar is the default interface users are presented with upon login to miPortal. miCalendar refers to the calendrical container of all events a given user has added to his/her miSchedule calendar. Those can be events created by the
user (and sent or just kept for personal use) and/or events created by others that the user has added to his/her miSchedule calendar. Among many other features of miCalendar, users can view events in multiple views (monthly, weekly, daily, and list view) and update events they own (cancel, postpone, and edit). Users can also filter miCalendar events by sender and/or category for a less crowded and more selective display of their events; that is, they can specify to display all events in their miCalendar that have been sent by this and that user(s).

### 3.2.2 Events Management

miEvents refers to all events a user has created (irrespective whether they've been sent or simply added to miCalendar). The generic miEvent term also refers to the proprietary miSchedule-defined event with all its attributes and handling mechanisms.

miEvents Management contains only events created/sent by a user, irrespective of whether he/she added them to miCalendar or not. miCalendar, however, contains only events explicitly added to it by the user irrespective of their sources.

All views, functionality, and features available in miCalendar are also available in miEvents management with the addition of event update privileges (since all events in this view are owned by the user) such as postponing, canceling, and editing the event. The default view in miEvents Management is List View since the user is expected to be performing batch operations over multiple events (such as group deletion or event grouping).

### 3.2.3 Settings and Preferences

miSettings comprise a user's full range of miSchedule-related settings: miPermissions, miNotifications, miPassport, miDefaults, and miInfo.
miPersmissions are the set of Yes/No permissions given by user A to users B and C depending on whether A wants the events they send to be automatically logged into A's miCalendar. For example, giving a school course management system a Yes permission would automatically save all events the system sends user A into A's miCalendar (A can toggle 'Notify me when a new event is added to miCalendar' to be notified of that). On the other hand, A wouldn't want to give a mailing list to which A is subscribed this same privilege. Of course, miPersmissions only work in the context of miSchedule users.

miNotifications are the set of email notifications a user wants miServer to send him/her every time a certain event-related scenario occurs. A user can set whether to receive notifications if an event in his/her miCalendar is cancelled, edited, and/or postponed by its owner, if it conflicts with another overlapping event, and/or when an event is automatically added to miCalendar by another user given the appropriate level of permission.

miDefaults are the set of default values a user can preset for certain event attributes that seldom change so that he/she would save time defining new miEvents and send them on the fly. A user's contact info, for instance, need not be written every time an event is defined; most probably he/she will be the same person to contact most of the time.

3.3 The Advent of ASP.NET

In the development of miPortal, extensive use was made of the many benefits ASP.NET introduces over older web development models such as ASP or CGI.

The main goals of ASP.NET are to:

- Make code cleaner
- Improve deployment, scalability, security, and reliability
- Provide better support for different browsers and devices
- Enable a new breed of Web applications

Many of these features cannot be directly conveyed through the development of miPortal, as they inherent to the Common Language Runtime (CLR) operating in the background. A brief explanation of some of the features utilized to leverage miPortal development, especially as compared to older web development environments, is provided in the this section:

### 3.3.1 Event-Driven Model

Unlike ASP where there is no event handling, rather form submissions through `Http get` or `post` methods, ASP.NET now supports event handling. The event model for ASP.NET is very similar to the event model for Windows Forms. It is this similarity that makes programming with ASP.NET extremely easier. However, the difference in the case of Web Forms is that events get raised on the client and processed on the server (unlike windows form where the events are handled on the same machine). Because round trips to the server are expensive, events do not automatically cause a postback to the server. Server controls have what is known as an intrinsic set of events that automatically cause a postback to the server. The most common intrinsic event is a button click. Other events, such as selecting an item in a list box, do not cause an immediate postback to the server. Instead, these events are cached, until a button click causes a post to the server. Then, on the server the various change events are processed, in no particular order, and the button-click event that caused the post is processed. In the case of miPortal, this event driven model for web development was crucial in facilitating the process of displaying and manipulating complex data such as displaying events in a flexible calendar. This event driven model is exemplified at miPortal when switching calendar months, submitting partial forms in miSettings, and managing events in the event view.
3.3.2 Server-Side Processing

One of the big complications in ASP is that pages simply define one big function, which starts at the top of the page and ends at the bottom. The page content is rendered in the page order, whether it is straight HTML or ASP-generated HTML. Therefore, the loading of a given control is dependent on its position in the page, and there’s no way to target HTML controls except by rendering them as part of the stream. When using such a scheme with complex data representations such as daily or monthly calendar views, it becomes extremely hard to manage the code in any given page.

ASP.NET solves this problem by introducing a declarative, server-based model for controls. This is where the concept may seem alien to ASP programmers, because the controls are declared on the server, can be programmed against on the server, yet can be event driven from the client.

Making a control run on the server allows using its ID attribute to identify it directly. It is also more natural to refer to the control directly, which makes developing pages simpler. This can be extensively seen in miPortal where C# code is non-existent in the HTML/ASP pages. Not only did using server side processing make the code easier to write, but it also improved the way front-end logic was handled. Content on a given page was no longer identified by where it is positioned in the page, but rather to which logical group it belongs. An example of that is the miSettings page (screenshots in Appendix IV: miPortal Screenshots, p.141)) where many forms coexist in a single page, and the referencing of form items is bundled based on which group, such as miNotifications, it belongs to.

3.3.3 Separation of Code from Content

Typical ASP pages have a mixture of scripting code interspersed with HTML elements. In ASP.NET there is a clean separation between code and presentation content. The server code can be isolated within a single <SCRIPT
3.3.4 Server Controls

ASP.NET provides a significant innovation known as server controls. These controls have special tags such as `<asp:textbox>`. Server-side code interacts with these controls, and the ASP.NET runtime generates straight HTML that is sent to the Web browser. The result is a programming model that is easy to use and yet produces standard HTML that can run in any browser. The use of such server controls in miPortal greatly facilitated the implementation of many features. For example, the monthly view uses the `<asp:calendar>` server control to its fullest with extreme customization.

3.3.5 User Controls

In traditional ASP, code reuse and encapsulation was traditionally done using a combination of `include` files and web classes. While this worked reasonably well for business logic, it was always a little awkward for visual components. For example, to display a grid of data in many different places with the same general look and feel, while being customizable for a particular page, one would either cut and paste the HTML, use style sheets, use an `include` file, write some script to generate the HTML on the fly, or use a combination of all these methods. It could also be difficult to move these components between projects because there was the omnipresent problem of ensuring that variable names didn't conflict, and that the `include` file is included only once (and in the correct order).

ASP .NET solves many of these issues with the introduction of User Controls. These are self contained visual elements that can be placed on a web page in
the same way as a tradition intrinsic HTML control, and can have their attributes set in a similar fashion.

To use the control, the page must be made aware of the control. This is achieved using a Register directive that specifies the tag prefix to be used, the tag name and the location of the user control's page (DailyView.ascx in this case):

```html
<%@
Register TagPrefix="miSchedule" TagName="DailyView"
Src="DailyView.ascx" %>
```

Listing 1 Registering a User Control on an ASP.NET Webpage

The control is then embedded in a page using the following simple syntax:

```html
<miSchedule:DailyView Title="User Control Test"
id="DailyView1" runat=server />
```

Listing 2 Embedding a User Control on an ASP.NET Webpage

miPortal heavily depends on user controls for their reuse capability. A detailed listing of all controls constituting miPortal is provided in 3.4.3 Pages and Controls on p.38.

3.4 Design and Implementation

3.4.1 Data Access

As far as data access is concerned, all data access demand is catered for in the core classes referenced by miPortal (see 6.4 Core System Logic, p.84). This means that the code-behind in miPortal pages only addresses the visual formatting of data and implements the use-case scenarios and interaction logic.
3.4.2 Page Structure

Towards a consistent look and feel for miPortal, a master HTML design template is used by all pages. The HTML design is based on a nested table structure. CSS (Cascading-Style Sheets) was only used to define classes of text fonts and colors.

This is a sample page’s structure that shows both the HTML template’s format and the modularity in which user controls can be plugged:

![Master HTML Template with Modular User Controls](image)

**Figure 5** Master HTML Template with Modular User Controls

Notice how the user controls (the shaded gray areas) fit into the HTML template used in all the pages.

3.4.3 Pages and Controls

Below is the list of pages and user controls forming miPortal (screenshots of most pages can be found in Appendix IV: miPortal Screenshots, p.141):

<table>
<thead>
<tr>
<th>Page Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>./public/index.aspx</td>
<td>This is the main page upon accessing <a href="http://miSchedule.mit.edu">http://miSchedule.mit.edu</a>. It is an introductory page with a login form and links to all informational pages.</td>
</tr>
<tr>
<td>Path</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>./public/miaddin.aspx</td>
<td>Informational page about miAddin Xp. It also features a download link for the miAddin Windows Installer.</td>
</tr>
<tr>
<td>./public/mifaqs.aspx</td>
<td>Informational page containing frequently asked questions about miSchedule</td>
</tr>
<tr>
<td>./public/mischedule.aspx</td>
<td>Informational page about the miSchedule initiative.</td>
</tr>
<tr>
<td>./public/miservices.aspx</td>
<td>Informational page about miServices. It also features a link to download the miService SDK and documentation.</td>
</tr>
<tr>
<td>./public/mitags.aspx</td>
<td>Informational page about the miTags specification. It features a full description of all the tags recognized by miSchedule as well as a form to test sending miTagged emails.</td>
</tr>
<tr>
<td>./public/newevent.aspx</td>
<td>Event express page where unauthenticated users can, in a single trip to the server, send an event and provide their credentials for authentication.</td>
</tr>
<tr>
<td>./public/pcontactus.aspx</td>
<td>Contacts page for the unauthenticated user.</td>
</tr>
<tr>
<td>./public/register.aspx</td>
<td>miSchedule's user registration page. It also caters for special registration, where if a user was registered by a miServices client, he/she can complete their registration.</td>
</tr>
<tr>
<td>contactus.aspx</td>
<td>Contacts page for the authenticated users.</td>
</tr>
<tr>
<td>daily.aspx</td>
<td>The daily view of miCalendar.</td>
</tr>
<tr>
<td>default.aspx</td>
<td>The monthly view of miCalendar. It also redirects to the default miCalendar user view.</td>
</tr>
<tr>
<td>eventview.aspx</td>
<td>This is the event view where all information about a certain event instance is displayed. Actions such as cancel, postpone, forward, add notes, and edit are all part of this page. This page checks whether the user is the owner of the event to allow the appropriate event management privileges.</td>
</tr>
<tr>
<td>list.aspx</td>
<td>The list view of miCalendar.</td>
</tr>
<tr>
<td>loggednewevent.aspx</td>
<td>Send event page where users can send events, add them to miCalendar, or retrieve their miLinks.</td>
</tr>
<tr>
<td>login.aspx</td>
<td>The standard login page to which unauthenticated users get redirected if any secure resource is requested. It is a stripped down version of the login control available in the informational pages.</td>
</tr>
<tr>
<td>Control Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>logout.aspx</td>
<td>Transitional page that removes the authentication cookie from the client's computer and resets the session variable holding the user information on miServer.</td>
</tr>
<tr>
<td>mievents_daily.aspx</td>
<td>The daily view of the miEvents section.</td>
</tr>
<tr>
<td>mievents_list.aspx</td>
<td>The list view of miEvents.</td>
</tr>
<tr>
<td>mievents_monthly.aspx</td>
<td>The monthly view of miEvents.</td>
</tr>
<tr>
<td>mievents_weekly.aspx</td>
<td>The weekly view of miEvents.</td>
</tr>
<tr>
<td>milink.aspx</td>
<td>This is the page reached whenever a user clicks on a miLink for a certain event. The page displays information about the event and provides an 'Add to miCalendar' button.</td>
</tr>
<tr>
<td>milists.aspx</td>
<td>The miLists manager. Users can view, add, and remove emails to mailing lists they can define.</td>
</tr>
<tr>
<td>miPortal.aspx</td>
<td>Transitional page involved in fixing the authentication flow. If a user requests miSchedule's main page, he/she will get redirected to the index informational page. In case the user is authenticated, he/she will get redirected to his/her default calendar view.</td>
</tr>
<tr>
<td>misettings.aspx</td>
<td>User's settings page where changes in miPersmissions, miNotifications, miPassport, and milInfo can be made.</td>
</tr>
<tr>
<td>public_daily.aspx</td>
<td>The daily view of the public calendar.</td>
</tr>
<tr>
<td>public_list.aspx</td>
<td>The list view of the public calendar.</td>
</tr>
<tr>
<td>public_monthly.aspx</td>
<td>The monthly view of the public calendar.</td>
</tr>
<tr>
<td>public_weekly.aspx</td>
<td>The weekly view of the public calendar.</td>
</tr>
<tr>
<td>weekly.aspx</td>
<td>The weekly view of miCalendar.</td>
</tr>
</tbody>
</table>

Table 1 Partial Listing of miPortal Pages with Descriptions

<table>
<thead>
<tr>
<th>Control Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DailyView.ascx</td>
<td>Calendar daily view used in the daily views of miCalendar, miEvents and Public Events. It displays events for a given day in a spanning table format.</td>
</tr>
<tr>
<td>EventExpress.ascx</td>
<td>Provides the description and link to define a new event express.</td>
</tr>
<tr>
<td>header.ascx</td>
<td>Header available on all mischedule pages. This controls has two modes: when user is authenticated, and when unauthenticated.</td>
</tr>
<tr>
<td>Control Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>InfoMenu.ascx</td>
<td>The menu showing the links to the informational pages.</td>
</tr>
<tr>
<td>ListView.ascx</td>
<td>Calendar list view used in all the calendars available on miPortal. It displays events in a list format.</td>
</tr>
<tr>
<td>LoginBox.ascx</td>
<td>Provides the authentication form and links to the register and forgot your password pages. This control is mainly used in the informational pages and morphs into a go to miCalendar button whenever a user is logged in.</td>
</tr>
<tr>
<td>miAddinInfo.ascx</td>
<td>Contains the information pertaining to miAddin Xp. Also features a link to download miAddin.</td>
</tr>
<tr>
<td>miAddressBook.ascx</td>
<td>Users can add/remove contacts from their mischedule addressbook using this control.</td>
</tr>
<tr>
<td>miCalendarMenu.ascx</td>
<td>The menu in the secure pages providing links to the different views, adding new events and managing miLists.</td>
</tr>
<tr>
<td>miContactUs.ascx</td>
<td>Contains the information pertaining to the Contacts page.</td>
</tr>
<tr>
<td>miFAQ.ascx</td>
<td>Contains the miSchedule FAQs presented in a list format.</td>
</tr>
<tr>
<td>miFilter.ascx</td>
<td>Provides the list boxes to filter the events displayed in any monthly calendar view. This control updates its lists based on the loaded events in the page.</td>
</tr>
<tr>
<td>milndex.ascx</td>
<td>Contains the information available on the index page.</td>
</tr>
<tr>
<td>miServicesInfo.ascx</td>
<td>Contains the information pertaining to miServices; also has links to all SDK documentation files.</td>
</tr>
<tr>
<td>miTagsInfo.ascx</td>
<td>Contains the information pertaining to the miTags specification.</td>
</tr>
<tr>
<td>miTips.ascx</td>
<td>Displays miSchedule tips and nomenclature.</td>
</tr>
<tr>
<td>PublicSearch.ascx</td>
<td>Provides information and links to the public calendar.</td>
</tr>
<tr>
<td>QuickDayView.ascx</td>
<td>Features a small calendar that shortcuts to the daily view of any clicked day.</td>
</tr>
<tr>
<td>UserInfo.ascx</td>
<td>Displays the information about the logged user such as email, last login and the number of logins.</td>
</tr>
<tr>
<td>WeeklyView.ascx</td>
<td>Calendar weekly view used in all the calendars.</td>
</tr>
<tr>
<td>WhatsMiSchedule.ascx</td>
<td>Contains generic information about the miSchedule initiative.</td>
</tr>
</tbody>
</table>

**Table 2** Partial Listing of miPortal Controls with Descriptions
4.1 Overview

A COM Add-in has been developed for Microsoft Office Xp to bring miSchedule functionality in a context-sensitive fashion to the powerful Office Suite. Upon installation, miAddin will load in any instance of MS Word, MS FrontPage, and MS Outlook and reside passively in the system tray, toolbar, and/or context menus of the host application. Depending on the host application, miAddin supports different environment-specific features.

Key miAddin features available in MS Word and MS FrontPage:

- Create and send miEvents (with access to miLists)
- Retrieve miLinks to include in documents and webpages
- Edit your notification preferences
- Save miCalendar to local file (.csv format – compatible with most PIMs)
- Add an event to miCalendar from any miLink in a webpage or document

In addition to the features supported in MS Word and MS FrontPage, the following features are available in MS Outlook:

- Automatic checking of all incoming email for miLinks
- MSN Messenger-style notification of incoming miLinks
- Import miCalendar into any local Outlook calendar (full field mapping; customizable using Outlook’s ‘Import and Export…’ wizard)
- Add an event to miCalendar and Outlook calendar from any incoming miLink
- Interact directly with your address book when specifying new miEvent recipients
The design and development of miAddin required extensive knowledge of the interoperability between COM and .NET. Although the .NET Framework has made provisions for this interaction by implementing various wrappers for COM objects to allow exposure of their properties and methods to .NET components (and vice versa), a number of problems were encountered during development in this not as thoroughly explored and documented area. This chapter will start with concisely introducing different key aspects involved in integrating miAddin with Office, and then describe the different classes constituting miAddin while skimming through its main interfaces and features.

4.2 COM and .NET Interoperability Difficulties

4.2.1 Detecting Host Application

miAddin is an in-process COM (Component Object Model) server or DLL (Dynamic Link Library) specially registered for loading by Microsoft Office Xp applications (Word, FrontPage, and Outlook). In the days before COM add-ins, to create a utility that would work across different Office applications, each application-specific portion of the utility in a programming language unique to the application in which it would be used. Beginning in Office 2000, applications support add-ins implementing a common COM interface called IDTExtensibility2 (described in the Microsoft Add-in Designer type library, Msaddndr.dll). All COM add-ins inherit from this interface and must implement each of its five methods.

The IDTExtensibility2 library provides a programming interface for integrating COM add-ins with their host applications. The IDTExtensibility2 library provides five events used by miAddin to detect, adapt, and integrate with the different host applications: OnConnection, OnDisconnection, OnAddInsUpdate, OnStartupComplete, and OnBeginShutdown. For
example, miAddin implements the OnConnection method as follows (edited) to detect the host application through Reflection:

```csharp
public void OnConnection(object application, Extensibility.ext_ConnectMode connectMode, object addInInst, ref System.Array custom)
{
    //Host application
    applicationObject = application;
    addInInstance = addInInst;
    //Detecting host application using Reflection
    this.applicationName = this.applicationObject.GetType().InvokeMember("Name", BindingFlags.Get
    Property,null, applicationObject, null).ToString();
    if (connectMode != Extensibility.ext_ConnectMode.ext_cm_Ext)
    {
        OnStartupComplete(ref custom);
    }
}
```

Listing 3 Implementing the IDTExtensibility2 to Integrate with Host Application

After detecting the host application, miAddin’s interface, functionality, and host application event handling is accordingly adapted.

### 4.2.2 Event Handling in Host Application

Choosing the Office platform to host miAddin instead of building a stand-alone application was primarily due to the driver in exploiting the relevant functionality readily available in Word, FrontPage, and Outlook. Regular Office users would hence welcome the advent of miAddin to maximize their miSchedule experience
and bring it to their desktops. However, for miAddin to seamlessly integrate with the different Office applications while residing unobtrusively in the background, total control over event handling and communication with the host application should be achieved. We faced a number of problems working towards that end due to the still immature .NET/COM interoperability and lack of support and documentation in this area.

For example, an instant notifier (Messenger-style) was implemented to alert users to incoming miLinks:

![Image of instant miLink Notifier in MS Outlook]

Figure 6 Instant miLink Notifier in MS Outlook

To achieve this, the **NewMail** event in Outlook is handled as shown below by the **NewMailHandler** function. An instance of the delegate (**Outlook.ApplicationEvents_10_NewMailEventHandler**) bound to this function is made and then attached to the new Outlook mail trigger:
Although everything seems clean and simple here, for this to properly work, the Outlook interoperability assembly (Interop.Outlook.dll) had to be disassembled, modified, and reassembled again. It turns out that the event wrapper classes in the interop assembly generated by Outlook have overly strict access privileges (inaccessible, in other words). Because these wrapper classes are marked private, the universal runtime disallows an IUnknown: :QueryInterface call to the wrapper classes and returns E_NOINTERFACE. This translates to an exception error. The only way to work around that was using the ildasm/ilasm tools provided with the .NET Framework to disassemble/reassemble the interop assembly for modification and reuse.

### 4.2.3 Other COM-Related Issues

Many other difficulties are introduced to Office development at this transitional stage between COM and .NET. For programs that are written with the .NET Framework to be more stable and securable than earlier programs (hence safer to run), the price is that there are structural differences in how the code is managed by the common language runtime as opposed to in the COM world. And because Office was built around COM and OLE, COM interfaces must still

```csharp
...
// Checking if correct host application and
// instant notification is desired by miAddin user
if (this.miSettings.IsAutoCheckForMiLink &&
    frmMain.miCaller.applicationName == "Outlook")
{
    frmMain.miCaller.OutlookApp.NewMail +=
    new Outlook.ApplicationEvents_10_NewMailEventHandler(frmMain.miCaller.NewMailHandler);
}
```

Listing 4 Handling the NewMail Event in Outlook
be used, yet adversely affected by core .NET features such as deterministic programs (like an Office COM server) residing longer in memory after the application shuts down due to the non-deterministic .NET garbage collector. Just-In-Time compiling and the absence of variant introduce other constraining implications as well.

4.3 Design and Implementation

The miAddin solution consists of 3 distinct components: (1) the miAddin front-end application, (2) the miAddin Web Services used to communicate user and event information between miServer and miAddin, and (3) the miAddin Setup for packaging and deploying miAddin. This section will outline the internals of the first 2 components through highlighting the functionality of their constituents:

Listing 5 Libraries Referenced by miAddin’s Front-End (right)

Listing 6 Structure of miAddin’s Front-End Solution (left)
4.3.1 Front-End Application

This application consists of 8 classes, 5 of which are windows forms (frmXXX) constituting miAddin's functionality and interface, 1 of which implements the IDTExensibility2 interface to communicate with the host application (Connect), and the last 2 managing the miAddin settings and user preferences through writing and reading from the system registry (Savior and Settings).

- **Connect**: This class implements the five IDTExensibility2 methods OnConnection, OnDisconnection, OnAddInsUpdate, OnStartupComplete, and OnBeginShutdown described earlier. It is responsible for detecting the host application and directing the miAddin loading accordingly. It then attaches the miAddin interface buttons to command bars and context menus of the host application, and defines their corresponding event handlers. For example, in OnStartupComplete it detects if the host application is FrontPage and attaches the buttons accordingly:
public void OnStartupComplete(ref System.Array custom)
{
    //Detecting host application
    switch (this.applicationName)
    {
    case "Outlook":
    {
    -
    }
    case "Microsoft Word":
    {
    -
    }
    case "Microsoft FrontPage":
    {
    this.FPApp = (FrontPage.Application)this.applicationObject;
    //Checking if the button is still there from a previous
    startup and abrupt shutdown
    try
    {
        miStandardButton =
        (CommandBarButton)this.FPApp.CommandBars["Standard"].Controls["&miAddin"];  
        //Adding a new button if not available
        catch
        {
        miStandardButton =
        (CommandBarButton)this.FPApp.CommandBars["Standard"].Controls.Add(1, "omissing",
        omissing, omissing, IsTemporary);
    -
    }
    //Defining button properties
    miStandardButton.Caption = ";miAddin";
    miStandardButton.ToolTipText = "Open miAddin for Office Xp";
    miStandardButton.BeginGroup = true;
    miStandardButton.FaceId = 2608;
    miStandardButton.Style =
    MsobuttonStyle.msoButtonIconAndCaption;
    miStandardButton.Tag = "&miAddin";
    miStandardButton.Visible = true;
    ton_Click);

Listing 7 Adapting miAddin Interface in FrontPage Host Application
If the host application is Outlook, it also attaches delegates to the events of interest (NewMail triggered whenever new email arrives and NewInspector triggered every time the user opens an inspector window to read/write an email). On host application shut down, Connect is also responsible for freeing resources and forcing garbage collection so that miAddin does not force host application to remain in memory. Below is a listing of the Connect class methods:
- **frmMain**: This is miAddin’s main interface, a tabbed form providing access to all miAddin functionality. It consists of around 60 methods and event handlers, and around 120 fields and controls. Upon loading, it uses the Settings class to read user settings from the registry and customize the application accordingly (see Settings class). All other frmXXX classes are accessible from this class. The following are snapshots of the different frmMain tabs:
The Matrix

Fields are case-sensitive

Email: mischedule@mit.edu
Password: 

Time Zone (GMT -05:30)

Subject: You're Invited to my Birthday!
Category: Party
Start: Mon 05/12/03 12:00 A
End: Wed 05/14/03 12:00 A
Location: 77 Mass. Ave., Cambridge, MA 02139

Contact: 

Body:

Checkbox options:
- This event is an invitation
- This is a public event
- Add to miCalendar as well
- This is a recurring event

URL: http://mischedule.mit.edu

Clear  Send miEvent  Get miLink  Add to miCalendar

Any additional information you may wish to provide about your new miEvent
miAddin Settings
- Automatically check incoming mail for miLinks
- Remember my password for the next sign-in
- Load miAddin as soon as I open Outlook (for faster performance)
- Add event from miLink to Outlook calendar as well

miNotifications
- Notify me when an event I have added is: cancelled, postponed, updated, conflicting with others, automatically added to miCalendar

miCalendar
- Save miCalendar to a .csv file
- Import miCalendar into Outlook
- Starting: Sunday, February 02, 2003
- Ending: Saturday, June 28, 2003
- Include attributes with imported events

Advanced
- Direct link to miSettings (more preferences)
- Direct link to miEvents Management
- Direct link to miPortal
**Settings** and **Savior**: Those 2 classes are responsible for writing and reading miAddin settings from the registry. Many of the settings are interdependent, especially when authentication comes into play: does the user want miAddin to save his/her password? And if so, is automatic sign-in also desired? Whereas, if not, does the user want miAddin to be loaded in the background as soon as the host application is started to avoid a performance hit due to JIT-compilation (Just-In-Time) later on? Etc... Below is a listing of the different settings saved to the registry:

![Settings Tree](listing_9_settings_tree.png)

**Listing 9** Settings Class Fields

Outlook-specific settings consist of the user's default Outlook calendar to import to when adding events from miLinks, and whether to automatically import events he chooses to add (to miCalendar) locally to Outlook as well.

**frmConnecting**: This form is a semi-transparent splash-like form with a progress bar displayed every time a connection to miServer is made via miAddin Services. It contains a public label accessible from calling methods to customize and update the status of the connection with miServer. For
example, below is a snapshot of the frmConnecting as the user is importing a calendar:

![Connecting to miServer](image)

**Figure 12** frmConnecting While User Imports miCalendar

- **frmDefaultCaledar**: This form prompts the user to choose the default calendar into which he would like to import his events (in addition to miCalendar). Upon loading, all calendar folders ('Appointment Item') in Outlook are detected and listed in the following list box:

![Default Outlook Calendar](image)

**Figure 13** frmDefaultCalendar Showing Detected Outlook Calendar Folders

- **frmMiLink**: This form retrieves a new event's miLink and gives the user the option of copying it to the clipboard or inserting it into host application's primary workplace (email in Outlook, webpage in FrontPage, or document in Word):

![miLink](image)

**Figure 14** frmMiLink Displaying Retrieved miLink
- **frmSend**: After defining an event, the user can choose to send it. The recipient options involve: (1) miLists (retrieved instantly from miServer), (2) the Address Book when (only functional in Outlook), and/or (3) email addresses defined on the fly.

![frmSend](Send_miEvent.png)

**Figure 15** frmSend Used to Select miEvent Recipients

A help file built using the HTML Help Compiler was also packaged with miSchedule:
4.3.2 Server-Side Component

miAddin Services constitute the server-side Web Services required by miAddin to communicate user and event information with miServer. In addition to the web methods, several classes are defined to wrap the corresponding miClasses (see 6.4 Core System Logic, p.84):
miAddin Services communicate with the database via miClasses then send/receive information to/from miAddin via new classes adapted and packaged for optimal communication with miAddin. For example, consider the 2 different versions of an event, one sent by miAddin (miAddinEvent) when the user creates and sends an event, and the other used by miAddin Services to return an array of events as part of the miCalendar import feature. The latter is structured in a way compatible with Outlook event fields for instant mapping:

**Listing 10** miAddin Server-Side Classes and Web Services
Listing 11 miAddinEvent vs. miAddinImportEvent in miAddin Services

Listing 12 Stripped-Down Version of Core miUser Class

Listing 12 (top right) is an abridged version of the core miUser class implementing only the fields and data necessary in the context of miAddin:

As for the miAddinServices class, it constitutes the set of web methods consumed by miAddin to fulfill the following functions: (1) authenticating the user (involves retrieving miNotifications, miDefaults, and other user information right upon authentication), (2) returning use’s miLists when needed, (3) importing miCalendar, (4) updating miNotifications, (5) retrieving a miLink for an event, and
(6) sending a new event. Those methods and their signature are provided below (synchronous and asynchronous versions):

```csharp
Listing 13 Synchronous and Asynchronous miAddin Web Methods
```

4.3.3 Deployment

A deployment solution for miAddin was also configured. Besides the .NET Framework prerequisite to using miAddin (should be available at user's computer), miAddin Setup packages all libraries (especially the reassembled interop assemblies) into one .msi file. This file also includes the miAddin Help file.
and other proprietary material (license information, logos, etc...) required during installation. The different components referenced by miAddin Setup are provided below:

Listing 14 Dependencies Included in miAddin Deployment Package
Chapter 5  Tagging Schema and Message Handling

5.1 Overview

"The problem with your proposal in q.5 is that it is web-based. If you developed a command-line based Unix client, I will definitely use it."

- miSchedule survey participant (Jan '03)

miTags is a comprehensive tag schema that miSchedule users can use in emails to define, register, and send a miEvent. This mean of communicating events is very handy for power users using text-based email clients or tend to use the command-line (console window) as their primary computing environment. With a few tags, this user can send a fully qualified event to people and/or predefined miLists in no time and with minimal constraints.

To handle miTags, an incoming POP3 mail server is controlled by miMailHandler via a Windows Service running on miServer. This email handler is also responsible for all auto-generated outgoing email from miServer, such as in the case of updated notifications, new event sending (rerouted via email to unregistered recipients and permission-denying users), and other promotional outgoing email.

This chapter will define in detail the miTags specification, briefly introduce the technologies used to handle, extract, and reroute miTagged events, and overview other aspects of the central email handler, such as outgoing notifications.
5.2 Tag Specification

5.2.1 Flexibility

The miTags specification is very flexible and has been designed to minimize the amount of time and text that is required to fully qualify an event:

- In-line tags are allowed. For example, you can write: "... <miCategory> academic <miIsPublic> yes </miIsPublic></miCategory>..."
- Shorter alias versions of miTags are provided
- miTags and their contents are not case-sensitive
- Most miTags are optional, many others can have default values preset by users once and for all (using miDefaults)
- Intuitive parsing/validation is done at miServer to the end of minimizing chances a miTagged email would bounce back to its sender

5.2.2 Tag Description

A miTagged email is sent to miTags@mischedule.mit.edu with a set of valid miTags where it would be parsed for the new miEvent which is in turn rerouted to its respective recipients (miLists and/or email addresses specified in the <miEventRecipients> tag). The table below contains all miTags; their aliases, descriptions, formats, and qualifiers:

<table>
<thead>
<tr>
<th>miTag</th>
<th>Alias</th>
<th>Description</th>
<th>Format/RegExp</th>
<th>Qualifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>miSender</td>
<td>Sender</td>
<td>Sender's email address with which he/she has registered with miSchedule</td>
<td>RegExp: \s*([^&lt;\miSender&gt;[w-]+({1}([w-]+)<em>@([w-]+{1}+[w-]+)</em>@)*</td>
<td>Required; yet if omitted or invalid, email's &quot;From&quot; field will be searched for alternative registered sender. Otherwise email bounces back to sender</td>
</tr>
<tr>
<td>miRecipients</td>
<td>Recipients</td>
<td>List of intended miEvent recipients</td>
<td>One or more valid email addresses separated by</td>
<td>Required; one at least, yet if omitted, event is logged into sender's</td>
</tr>
<tr>
<td>miPassword</td>
<td>Password</td>
<td>Sender's partial password (first four letters)</td>
<td>Four-letter string</td>
<td>Required</td>
</tr>
<tr>
<td>------------</td>
<td>----------</td>
<td>-----------------------------------------------</td>
<td>-------------------</td>
<td>----------</td>
</tr>
<tr>
<td>miEventSubject</td>
<td>Subject</td>
<td>miEvent subject</td>
<td>String (line breaks, leading and trailing white spaces, and double white spaces are removed)</td>
<td>Required</td>
</tr>
<tr>
<td>miEventStartDate</td>
<td>StartDate</td>
<td>The date on which the new miEvent starts</td>
<td>MM<em>DD</em>YYYY format, where * is either - or /</td>
<td>Required</td>
</tr>
<tr>
<td>miEventStartTime</td>
<td>StartTime</td>
<td>The time of day on which the new miEvent starts</td>
<td>0-23 hour format. HH*MM, where * is either - or /</td>
<td>Required</td>
</tr>
<tr>
<td>miEventCategory</td>
<td>Category</td>
<td>miEvent category</td>
<td>String; must be a valid category available in miSchedule event categories</td>
<td>Optional; set to user's default if missing or invalid</td>
</tr>
<tr>
<td>miEventEndDate</td>
<td>EndDate</td>
<td>The date on which the new miEvent ends</td>
<td>MM<em>DD</em>YYYY format, where * is either - or /</td>
<td>Required</td>
</tr>
<tr>
<td>miEventEndTime</td>
<td>EndTime</td>
<td>The time of day on which the new miEvent ends</td>
<td>0-23 hour format. HH*MM, where * is either - or /</td>
<td>Required; yet if omitted, event is set to end at end of day on which it starts; email bounce back if invalid</td>
</tr>
<tr>
<td>miEventTimeZone</td>
<td>TimeZone</td>
<td>miEvent time zone</td>
<td>(GMT *HH:MM); where MM is 00 or 30, HH is between 00 and 23, and * is + or - (with the exception of 12:30 and 00:00)</td>
<td>Required; yet if omitted, time zone is set to that in sender's milinfo; email bounce back if invalid</td>
</tr>
<tr>
<td>miEventBody</td>
<td>Body</td>
<td>miEvent body</td>
<td>String (line breaks, leading and/or white spaces &quot; &quot; )</td>
<td>Optional</td>
</tr>
<tr>
<td>miEventLocation</td>
<td>Location</td>
<td>Location at which miEvent is taking place and double white spaces are removed</td>
<td>Optional; if empty, default 'Location' in miSettings is used (if predefined)</td>
<td></td>
</tr>
<tr>
<td>-----------------</td>
<td>----------</td>
<td>--------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>miEventURL</td>
<td>URL</td>
<td>Hyperlink for further miEvent reference and double white spaces are removed</td>
<td>Optional; if empty, default 'Url' in miSettings is used (if predefined)</td>
<td></td>
</tr>
<tr>
<td>miEventContact</td>
<td>Contact</td>
<td>miEvent contact information</td>
<td>Optional; if empty, default 'Contact' in miSettings is used (if predefined)</td>
<td></td>
</tr>
<tr>
<td>miEventIsPublic</td>
<td>Public</td>
<td>Whether miEvent is public and searchable by any miSchedule user</td>
<td>Yes or No</td>
<td></td>
</tr>
<tr>
<td>miEventIsInvitation</td>
<td>Invitation</td>
<td>Whether miEvent is an invitation to which recipients should respond with their attendance status</td>
<td>Yes or No</td>
<td></td>
</tr>
</tbody>
</table>

Table 3 miTags: Aliases, Descriptions, Formats, and Qualifiers

Recurring events are not supported in the current miTags specification.

A simplified, yet elaborate, description of miTags is provided in Appendix V: Full miTags Description, p.151.
5.2.3 Example

A sample email with the few miTags (aliases are used) required to successfully communicate an event could look like:

From: kobayashi@mit.edu
To: miTags@miSchedule.mit.edu
Subject: Recitation Session

... (body of email)

Dear TAs,

There will be a recitation session for the 1.954 class this Thursday at 5pm. Please come an hour earlier so that we can discuss the recitation material before class.

Listing 15 miTagged Email Example

5.3 Tag Handling

5.3.1 Regular Expressions

Before delving into the detailed cycle of email handling, it should be mentioned that Regular Expressions is the technology used at the very core of incoming email parsing, miTag detection, and event integrity validation. The .NET Framework has a wealth of powerful classes and methods to deal with Regular Expression. These are found under the System.Text.RegularExpressions namespace. A simple example illustrating the regular expression used to validate the <miEventTimeZone> miTag would be:
Listing 16 Sample Regular Expression Used to Validate TimeZone

Now for any given string (call it A), applying the method IsMatch on reg with A as an input (r.IsMatch(A)), returns true if A is in the form of:

(GMT +HH:MM) or (GMT -HH:MM)

where HH:MM is any valid time zone difference. The RegexOptions.IgnoreCase flag specifies that case does not matter (i.e. instead of "GMT", "gMt", "Gmt", etc. is acceptable), and the RegexOptions.Compiled options specifies that the regular expression is compiled to an assembly. This yields faster execution but increases startup time (advised in the case of multiple calls making the startup hit insignificant).

The regular expressions for other miTags vary in complexity and length of course, yet proved indispensable for efficient and effective miTag parsing and, more importantly, validation.

5.3.2 Process Flow

The steps required to fully handle miTagged emails from receipt to notification are delegated to 3 main components: (1) miMailHandler, which contains the central classes used to parse, validate, and extract event information from miTagged emails and then issue outgoing notifications, (2) miService, a Windows Service running in the background checking for incoming email and making the necessary calls to miMailHandler, and (3) miMailService, a user interface used to
control miService and keep logs and temporal statistics relating to incoming and outgoing email.

All 3 components are described below with a brief overview of the responsibilities allocated to their underlying classes and members.

5.3.3 Handler Logic

miMailHandler consists of 8 classes with a total of around 2000 lines of code. Each of its classes is responsible for the following tasks:

Listing 17 Classes in the miMailHandler Solution

- **miMail**: An email wrapper class. The constructor has 11 overloads that account for almost all outgoing email needs of miSchedule: notifications, alerts, miTag errors, registration, new events, promotional email, and forgotten password.

- **miSMTP**: Contains a static `SendMail` method with 7 overloads. The method sends e-mail using the `System.Web.Mail` namespace. One of the overloads takes a `miMail` object as input.

- **miPOP3**: Has only one public static method: `GetMiMail`. This method connects to miPOP3 server and retrieves email in a `miMail` array. Has a
number of private methods that parse received email and makes the actual connection to miClasses. It uses System.Net.Sockets namespace.

- **miMailFileReader**: An exact replica of miPOP3, yet reads from the file system instead of an incoming mail server. It also has a public static method: GetMiMail. This method takes a path to a folder, reads all files, and returns a miMail array. It uses the System.IO namespace.

- **miTagFlags**: A miTags wrapper classes. All miTag flags, 16, along with their aliases, are specified. An indexed full access property is also provided. The ToString method is also overloaded to return two columns and multiple rows for each tag with its value. This is a list of miTagFlags fields (mapping to the aforementioned miTags):

### 5.3.4 Handler Windows Service

A Windows Service (formerly known as NT service) was used to create the long-running miService component that operates in its own Windows sessions. By this, the service can be automatically started when miServer boots, can be paused and restarted, yet does not show any user interface. In this sense, a Windows service was ideal for use on miServer; it runs in the background, consumes minimal resources, and can be configured to function uninterruptedly even when different Windows accounts are in session.

Although via the Service Controller interface provided with the Operating System (see Figure 17 Service Controller Showing miMail Service, p.70), a Windows Service can be started, stopped, paused, continued, restarted, refreshed and cancelled, important statistical information (such as how many e-mails were parsed, how many were logged in miCalendar, etc...) are hard to retrieve. In order to accommodate for this, a delegate-enabled Remoting scheme was used to share vital information across processes. Simple Remoting will require a
timer to check every interval if the information is updated, but with delegates, an event is fired across process boundaries whenever the information sought is updated.

miService consists of 2 classes roughly amounting to around 1000 lines of code:

- **miService**: This is the Windows Service that runs in the background listening to incoming email (either from POP3 server or folder) and triggers the required methods in miMailHandler to parse emails for miTags and log events. It extends System.ServiceProcess.ServiceBase.


![Figure 17 Service Controller Showing miMail Service](image)
5.3.5 Handler User Interface

This component constitutes the user interface used to interact with miService via delegates. It consists of 2 main classes: (1) miMailServiceSettingsForm, a graphical user interface mimicking the functionality provided by miTagConsoleClient, and (2) FolderBrowser, a dialog window used to browse for the logs and email folders:

Listing 18 Structure of the Mail Handler User Interface Component

![Image of miMailService Settings dialog window]

Figure 18 'miTag General' Tab of the Mail Handler User Interface
Running since: 01/05/03 00:01:15

Total number of e-mail received: 6
Successfully parsed: 6
Successfully pushed to miSchedule: 2
Total bounced: 2
Invalid user: 4
No recipient: 0

Outgoing Setup
SMTP Server: miserver.mit.edu Port: 25

Outgoing Activity
Last outgoing message: 05/05/03 23:40:45
From: support@mischedule.mit.edu
To: mitag@mischedule.mit.edu
Subject: miTag Error
Total outgoing message: 2
5.4 Outgoing System Messaging

As previously discussed, miMailHandler does not only handle incoming mail for miTags, but is also responsible for all outing mail. Outgoing miSchedule-generated email is triggered in any of the following scenarios:

- When an event is updated (cancelled, postponed, or edited) by its owner, users wishing to be notified are notified.
- When a new event is received by a user, he/she is notified via email (if that has been set in his/her miNotifications – see 3.2.3 Settings and Preferences, p.32).
- When an event reroutes through miSchedule to an unrecognized user, the event is explicitly spelled out in text with a miLink and sent in an email generated by miSchedule and addressed to the intended unregistered recipient (the From field of the email would bear that of the original sender).
- When a user registers, his/her password is auto-generated and sent via email.
- When a miServices consumer registers a user and chooses to notify him/her, miSchedule sends an informational email about the service inviting the user to complete registration at miPortal.
- When a miTags error occurs in a required miTag (see appendix for elaborate miTags description and qualifiers) necessitating that the sender be informed, an email is sent with the source of error in his/her original email.

The different overloads of the miMail class offer a number of central email templates shared in scenarios with similar email requirements, such as how textual content is structured, what variables and hyperlinks to include, etc... Those overloads use a predefined format when instantiating the appropriate
miMail object. For example, this is the parameterized body of a typical update notification (abridged):

```
miSchedule Event Updated Notification:

The event `event_subject` taking place on `date` has been updated. The event has been updated at your miCalendar; follow the link below to view the updated event:

   `event_link`

This event was sent by `event_sender_name` (<`event_sender_email`).
```

Listing 19 Update Notification Template
6.1 Overview

The miSchedule back-end consists of a data repository implemented in Microsoft SQL Server 2000 and a set of core classes written in C#. The interactions of all front-end miSchedule components (miPortal, miAddin, miServices, and miTags) with the database are routed through various channels always based on a central core class wrapper. This wrapper, referred to as miClasses, provides the means to retrieve data from and update the database by invoking stored procedures precompiled in the database. miClasses consists of 12 core classes with around 50 methods totaling to approximately 3500 lines of code. The database consists of around 25 entities and 60 stored procedures.

Figure 21: miSchedule's Back-End Repository and Logic
6.2 Data Model

This section provides an in-depth understanding of the various data model components that reflect the design approach to miSchedule's fully normalized database (5th normal form). The roles of four key data entities – Users, Events, EventInstances, and miCalendar – are first described and then other model subtleties and highlights are illustrated in relationship to the aforementioned main entities. Figure 22 (below) provides an overall detailed presentation of the various entities and entity relationships:
6.2.2 Key Entities

- **Users**: All miSchedule registrants who may play the role of an event sender or an event recipient are included within the same Users entity. User records include all relevant information (optionally filled out by a user) constituting the user profile.

- **Events**: This entity includes general event parameters that pertain to any of the events created by registered users. These events may be peculiar to a user in case they are added to his/her own calendar only, or they can be shared among several users as in the case of an event being sent from one of the registrants and logged into the calendars of various recipients. Events are owned by one of the registered users (UserID) who is the creator of the event.

In order to protect the privacy of users in terms of exposing their personal events while still permitting dynamic public searches for events, an event is described as either public (using the IsPublic option) or private. A private event is visible only to its owner – and the overall recipients of the event, if any. That is, a private event can be added by whoever explicitly references it via miLink or receives it from its creator (or from an intermediary recipient who, in turn, forwards it). A public event, however, is visible outside the scope of its owner and recipients and can used in the miSchedule Public Calendar or public event searches that can be performed on public events only.

Another characteristic of an event is it being an invitation or not as specified by the IsInvitation field. An event deemed an invitation by its creator makes the system prompt its recipients to confirm their attendance status. The system also keeps track of such responses for updating the sender and displaying in the event view of the invitation. (Refer to system process SP3: Event Adding via miLink on p.128 for additional information on the implications of an invitation event.)
Finally, RecurrenceType stores the recurrence of an event as validated by V_Recurrence (SingleInstance, WeeklyDaily, Monthly or FixedPeriod); whereas the Group field allows users to group their events in custom ways according to their personal tastes. A group can contain many events, and an event can be part of many different groups.

- **EventInstances**: Events can be conceptually categorized into two types, both of which are handled by very similar processes. The first type constitutes single instance events that occur only once. The second type covers all sorts of events that have multiple occurrences. A simple example of such events is a course class schedule. The course itself may be defined as a single event (described once in the Events table) with several child instances that represent distinct class dates. Accordingly, each event, whether consisting of a single instance or multiple instances, has one or more records in the EventInstances table that specify the date and time of each instance. For maximum flexibility, instances of the same event occurring on distinct dates can be scheduled at different times on different days.

- **miCalendar**: Events are associated with users through the miCalendar entity. Association is done between users and their event instances. Consequently, a user is not assigned an event – which might incur several instances that users decide to remove from their own miCalendar – instead, they are assigned instances of events. In this manner, a user can edit his miCalendar by removing particular instances of an event without deleting all instances of the same event. This design approach also allows for a very flexible handling of alerts that are specific to an instance rather than to a series. (Refer to system process SP5: miEvents Management on p.128 for a clearer illustration of miEvents handling.)
6.2.3 Other Model Subtleties

- **Permissions:** Users can grant specific senders permission to automatically log events into their own calendars (recipients'). Accordingly, a Me field in the Permissions table represents the person granting permission, while the To field represents the person to whom permission is granted. In addition, a Notify option is provided so that a user can choose to be notified by email whenever a new event is automatically added to his miCalendar (obviously, this allows new event notifications to be set on a per sender basis, instead of Yes or No to all new events). Refer to system process *SP5: miEvents Management on p.128* to see how sender-recipient permissions come into play as miSchedule dispatches events.

- **UserPreferences:** This holds user preferences related to many user-centric aspects of miSchedule. Fields consist of custom portal display preferences and notification settings (miNotifications); users can choose whether to be notified by email when an event is canceled, postponed, edited or conflicts with times of previously added events. (Refer to system process *SP5: miEvents Management on p.128* for a clearer illustration of miEvents handling.)

- **V_Categories:** This serves as a validation table for categories that a sender might categorize his outgoing event(s) with. This allows for sorting, filtering, and searching publicly visible events based on one or more of the categories specified in this table. The function of *V_Categories* differs from that of the Group field in the miCalender table as the latter is filled out by event recipients as they choose to regroup their own events based on their personal inclinations. (Groups can be created by users; categories, however, are preset by miSchedule.)
- **Alerts**: Alerts are caution messages that serve to notify users of event instances that have been cancelled, postponed, or edited by their owner or have been automatically received yet are in conflict with other events in their miCalendar. In case an instance is postponed, the new event that serves as a replacement is linked to the old event instance in order to make the user aware of the complete status of a particular event without the need for user intervention in associating postponed events with their corresponding replacements. (Refer to system process *SP5: miEvents Management on p.128* for an illustration of what happens when an event's integrity has been modified by the sender and the possible alerts resulting therefrom.)

- **Attendance**: This entity keeps track of recipient responses to invitation type events. (Refer to system process *SP3: Event Adding via miLink on p.128* for an illustration of how invitation events are handled upon being added by different recipients.)

- **RegisteredBy Attribute in Users**: Some users may hold a special registration status due to automated registration by miServices system clients. Such users are specially registered by an outside system which is itself registered with miSchedule and has the privilege of specially registering other users. (Refer to *SP6: New Event via miServices on p.128* for a clear illustration of special user registration via miServices.) This will allow the outside system to add and retrieve events for its users. However, users will only have access to events that were logged in by the outside system (that originally special registered him/her) from that system's website or system. Access from other miSchedule clients (miPortal and miAddin, for instance) to special registered user accounts will be restricted until such users fully register (by completing registration at miPortal) with miSchedule.
6.3 Data Binding

Although miClasses constitutes a direct contact point with the database, it is entirely based on stored procedure calls precompiled therein. SQL statements are therefore not only absent in the front-end clients, but in miClasses itself as well. Stored procedures have been implemented to provide the following benefits:

- **Preventing duplication in query commands between miClasses and the SQL database:** Consider for example the essential process of authenticating based on an Email and Password. The stored procedure miSchedule.ProcAuthenticate eliminates the need to restate the necessary conditions every time a method requires user authentication. Instead, the conditional statements are incorporated within the stored procedure.

- **Reducing database connections:** Several query statement are combined within a single stored procedure which decreases the number of trips from the miClasses assembly to the database.

- **Enhanced querying performance:** A major advantage of using stored procedures is their execution in precompiled state. Accordingly, data retrieval and update occur faster than in the case of random query statements incorporated in C# code when needed.

For example, consider the following Proc_GetInstance stored procedure:
**Listing 20** Sample Core Stored Procedure *(Proc_GetInstance)*

*Proc_GetInstance* is one of the commonly used stored procedures to retrieve event instance information relative to a user. Conditional selection is conducted based on whether the requesting user already has the event instance in his miCalendar. In case the user has the event instance, selection is done from the miCalendar entity, otherwise it is done from the generic EventInstances entity. In addition, an OUTPUT parameter specifies whether the instance belongs to the user or not.
6.4 Core System Logic

A system design based on the aforementioned 4-tier approach has proved its effectiveness in the development and maintainability of miSchedule. miClasses constitutes a single access point to the database which eliminates the possible incompatibility between stored procedures and their corresponding calls from code. Since all stored procedure calls are contained in the same project environment (the SQL Server database), development on the front-end components progressed independently of low level database querying. Accordingly, front-end development (miPortal, miAddin, miServices and miTags) were served by shared public methods exposed in miClasses.

Another advantage in adopting a database wrapper approach is minimizing duplication in code with the same functionality. To illustrate, consider the functionality of authenticating a user. Each of miPortal, miAddin and miTags require the same process of user authentication. Consequently, a single basic method (with overloads to account for more demanding requests, such as checking whether a user is special registered by a miServices consumer) is provided instead of having three methods at the various front-ends. This approach has efficiently sped up the progress of front-end developers as they implemented reusable methods across the various channels.

6.4.1 Key Classes

- **miUser**: This class provides the functionality required to manage user information and settings. This includes registration, authentication, permissions, personal information, and user defaults and notification settings (miSettings, miDefaults, and miNotifications – see 3.2.3 Settings and Preferences, p.32). Users handled by this class are either regular users, specially registered users (when registered by a miServices consumer), or miServices consumers (which have privileges of registering and accessing
events of special users). The following are the various methods associated with the `miUser` class:

```java
miUser

Bases and Interfaces

Methods

- Authenticate(string,string)
- AuthenticateTags(string,string)
- ChangePassword(string)
- GetPermittedEmailList()
- GetUser(string)
- GetUserEmail(int)
- GetUserID(string)
- GrantPermission(string)
- Register(string,string,string,string,string,DateTime,string,string,string,string,string,string,int)
- RegisterUsers(string[],int,bool,bool)
- RevokePermission(string)
- SetPermission(int,int)
- UpdateDefault()
- UpdateNotifications()
- UpdatePersonalInfo(string,string,string,string,string,DateTime,string,string,string,string,string,string,string)
```

**Listing 21 miUser Class Methods**

- **Event**: This class handles event management: registering new events, sending events in the form of emails to recipients, automatically logging events to calendars of registered users with permissions granted to sender. `ImportCalendar` overloads (see Listing 22 below) are used for importing events within a limited date frame and based on specified filtering criteria (by category or event owner). `ImportPublicCalendar` feeds `miPortal` with all events logged into `miSchedule` and declared as publicly visible.

The `RegisterEvent` method only stores the event description without adding it to any user-specific calendar. Several intermediary `SendMiEvent` methods are provided to send out events. This process handles recipient categorization based on their status: registered, not registered, registered with permission to sender, and registered without permission to sender.

Below are the sets of methods and fields in the `Event` class:
Event

**Bases and Interfaces**

**Methods**

- `FabricateEventInstances()`
- `GetEvent(int)`
- `GetEventID(int)`
- `GetEventInstances(int,int,out bool)`
- `GetEventInstances()`
- `GetEvents(int,string,DateTime,DateTime)`
- `GetEvents(int,DateTime,DateTime)`
- `GetMiLink()`
- `GetV_Categories()`
- `ImportCalender(int,DateTime,DateTime,ArrayList,ArrayList)`
- `ImportCalender(int,DateTime,DateTime)`
- `ImportPublicCalender(int,DateTime,DateTime,ArrayList,ArrayList)`
- `ImportPublicCalender(int,DateTime,DateTime,ArrayList,ArrayList)`
- `IsEventCancelled(int)`
- `IsEventPostponed(int)`
- `RegisterMEvent()`
- `SendEvent(string[],int[],bool)`
- `SendMiEvent()`
- `SendMiEvent(string[])`
- `SendMiEvent(int[])`
- `SendMiEvent(int)`

**Fields**

- `Attendees`
- `Body`
- `Category`
- `Contact`
- `EndDate`
- `EventID`
- `EventInstances`
- `FixedPeriod`
- `IsCancelled`
- `IsInvitation`
- `IsPostponed`
- `IsPublic`
- `Location`
- `Monthly`
- `OwnerEmail`
- `OwnerFullName`
- `OwnerID`
- `RecurrenceType`
- `StartDate`
- `TimeZone`
- `Url`
- `V_Categories`
- `WeeklyDailyArrayList`

**Listing 22** Event Class Methods

**Listing 23** Event Class Fields
**EventInstance:** This class handles instances of events by providing functionality to edit, cancel, and postpone events at the event instance level without affecting other instances belonging to the same parent event object (in case of a recurrence type event series). Instances of this class correspond to either EventInstances or miCalendar records. EventInstance methods and fields are:

- **Methods:**
  - CancelInstance(string)
  - Postpone(DateTime, DateTime, string)
  - RemoveMiCalendar(int)
  - UpdatePersonalNote(int, string)
  - UpdateSenderNote(string)

- **Fields:**
  - AlertArray
  - EndDateTime
  - InstanceID
  - PersonalNote
  - SenderNote
  - StartDateTime

**Listing 24 EventInstance Class Members**

**Alert:** Alert objects are accessed mainly through an array belonging to an EventInstance object. Conceptually, each instance can have none or many alerts, whereby each alert has a date and a status (cancelled, edited or postponed) and notes possibly provided by the owner at the time of update. Alert methods and fields are:

- **Methods:**
  - GetAlertArray(int, double, double)

- **Fields:**
  - AlertDate
  - AlertStatus
  - NewEventID
  - Note

**Listing 25 Alert Class Members**
- **MailingList**: This class caters for the functionality required in creating new mailing lists by a user. Users are able to manage their mailing lists by adding and removing contacts to and from a certain list. **MailingList** methods and fields:

```
MailingList
+ Bases and Interfaces
+ Methods
   - AddEmails(ArrayList)
   - CreateMlist(int,string,string[])
   - CreateMlist(int,string)
   - GetMMailingListNames(int)
   - RemoveEmails(ArrayList)
+ Fields
   - DateCreated
   - EmailArray
   - ListID
   - ListName
```

**Listing 26 MailingList Class Members**

### 6.4.2 Sample Central Method

Retrieving an EventInstance from the database is associated with the following processes: loading the parent event description, converting instance time to the user’s time based on the event time zone with respect to that of the requesting user, and finally importing the alerts associated with the returned event instance. The `GetEventInstance` method using the `Proc_GetInstance` stored procedure (**Listing 20, p.83**) is provided in
Appendix VI: GetEventInstance Method from miClasses, p.153 for extended illustration. GetEventInstance has this signature:

```java
/// <param name="UserID">A registered UserID</param>
/// <param name="InstanceID">A valid InstanceID</param>
/// <param name="IsInTheCalendar">A bool that is true if this instance is present in miCalendar of the user</param>
/// <returns>An event object with one instance in the EventInstances ArrayList corresponding to the requested InstanceID</returns>
```

Listing 27 Signature of GetEventInstance Core Method
7.1 Overview

At one extreme, *software-as-a-service* can imply nothing but a radical billing mechanism used by conventional software vendors to position their offerings as subscription-based *services*. At the other extreme, into which this chapter anchors, servicing software is an evolution of the monolithic application development, purchase, and implementation models to a service-oriented offering promoting business process reuse and extreme interoperability. In this sense, selling software as a service is proving to be a viable business strategy for the sustainable revenue stream it can produce based on providing value to users. In order to service software, however, it must be designed from the ground up to perform as a service and not a product. The Service-Oriented Architecture discussed herein and adopted by miSchedule (see 2.4.2 Alignment with Service-Oriented Architecture, p.29) furnishes the underlying design methodology and exploits Web Services as an ideal platform for deploying the intended software services.

Web Services promise to deliver all four generic business benefits of Information Technology: cheaper, faster, more reliable, and more functional [4]; a stage purportedly further than conventional technologies. After helping the reader discern what is truly differentiating about Web Services and applying that to the miServices offering, this chapter will tackle the different entry points into the Web Services markets in the context of its service provider-broker-requestor architecture, before focusing on miSchedule’s provider position in Chapter 9 Outsourcing Calendaring and Event Management, p.113.

From a global perspective, Web Services can generally be categorized into three interrelated deployment or end-usage types: (1) traditional software vendors
adding a Web Service interface onto existing products to unlock portions of a database, messaging system, or Enterprise Resource Planning (ERP) tools and the like, (2) an enterprise customer consuming such interfaces to solve current integration needs internally or externally with partners, and (3) a pure service provider offering interfaces to modular content and business processes ranging from daily weather and traffic updates to hotel reservations and credit-card authorization. The contrast between the three types is interestingly portrayed in their different business models: software vendors make money from serviceable product sales; enterprise customers gain return on their investment from increased efficiency and expanded customer revenue; and service providers prosper on recurring ‘rental’ revenue of the Web Services themselves. It is the standpoint of the latter, the service provider, that this part of the thesis takes in attempting to analyze the business potential and viable revenue streams generated by miSchedule’s miServices offering.

7.2 Software-As-A-Service Background

7.2.1 From a Product to a Service
Software has always been a book one buys, reads, and shelves away when done, and not a magazine one subscribes to. It has always been a product bought from a computer store or downloaded off the Internet, installed on the local machine, and used until upgraded or outdated. Not only consumers, but also businesses share the same viewpoint: software is a purchasable asset.

Now consider Microsoft’s Hotmail system: It’s no more than an email client the user chose not to purchase nor install. On this mass-market scale, the individual user is being offered software as a service instead of a purchased product. Hotmail also provides the added convenience of anywhere/anytime-accessibility to emails that users need not store locally. For basic service it is free of charge, but for more functionality and storage capacity, a subscription fee is required.
On the enterprise level, should software be viewed as a service instead of a product, a company can simply rent usage of an Enterprise Resource Planning (ERP) system from an Application Service Provider (ASP) and save itself the up-front expense, trouble, and time of installing, configuring, and maintaining it.

The business case for Service-Oriented Software (SOS) is fairly strong and easy to imagine, as there are inherent advantages for both the customer and the vendor in this business model. Customers do not have to make large cash outlays up-front, but instead can pay for a service as they use it and stop paying when it's no longer required. The vendor, on the other hand, gets a constant predictable income stream as long as customer loyalty is maintained through a useful, modern, and competitively-priced service.

To service software, the vendor must carefully instate a Service-Oriented Architecture (SOA) that would enable deploying software to provide a service rather than perform a function [2]. An SOA is fundamentally more flexible and adaptable promoting reuse and interoperability with the target service user base, which is usually diverse in essence. In fact, it is this diversity that led us to adopting the 4-tier architecture discussed in 2.4 Architectural Overview, p.26 and developing the different miSchedule clients; and eventually serve a larger customer base.

7.2.2 An Enabling Architecture

Although Gartner Group, claims to have first coined the term in 1996, it is believed that SOA has been used for a couple of decades to describe particular properties of a software system [15]. However, with the emergence and rapid adoption of a powerful industry trend, Web Services, SOA has taken on renewed importance in the software world.
Yates defines SOA: “Essentially, SOA is a software architecture that starts with an interface definition and builds the entire application topology as a topology of interfaces, interface implementations and interface calls... SOA is a relationship of services and service consumers, both software modules large enough to represent a complete business function. Services are software modules that are accessed by name via an interface, typically in a request-reply mode. Service consumers are software that embeds a service interface proxy (the client representation of the interface).” [21] – Hence making reuse, encapsulation, and agility (on the long run) the cornerstones of this software-as-a-service enabling architecture.

SOA brings these benefits to enterprise IT [20]:

- Incremental development and deployment of business software
- Reuse of business components in multiple business experiences
- Low-cost assembly of some new business processes
- Clarity of application topology

To service software and generate revenue, it must be designed to conform to two primary SOA principles: **system interoperability** and **location transparency**.

Two systems are said to be interoperable if they can communicate with each other. Systems that make use of the same middleware, although interoperable among themselves, may not be interoperable with other systems without the use of adapters or bridges.

Location transparency, on the other hand, is a property of distributed software components where one does not know the location of its peer [15]; the goal is that the client will be ignorant of the location of the server. Towards this end, two problems arise: (1) finding the server, and (2) communicating with the server (obstructed by routers, firewalls, and bridges). The first problem is solved using directories or registries, and the second using a usually unblocked protocol such
as the Hypertext Transfer Protocol (HTTP) that almost all firewalls allow through; both of which are defining characteristics of Web Services.

Once a system/architecture allows a server and client based on completely different technologies and platforms to communicate (i.e. interoperable) and removes the need for the client and server to know the location of one another (i.e. location is transparent), it can be rightfully considered Service-Oriented.

Gartner predicts that “by 2008, SOA will be a prevailing software engineering practice, ending the 40-year domination of monolithic software architecture (0.7 probability).” [20]

7.2.3 SOA with the Advent of Web Services

It is important that the reader understand the synergetic relationship between SOA and Web Services, just as important as it is to differentiate between the two.

As evidenced in the forthcoming section of this chapter devoted to Web Services (7.3 Web Services: The Technology and Beyond, p.96), Web Services are about technology standards, whereas SOA is a software design principle. And although Web Services do not necessarily translate into SOA, and not all SOA is based on Web Services, the relationship between the two technology directions is important and mutually influential: Web Services momentum will bring SOA to mainstream users, and the best-practice architecture of SOA will help make Web services initiatives successful. [20]

Web Services provide the ideal deployment environment for an SOA and the revenue-generating services the architecture is to host. Web Services, deployed correctly, provide the necessary system location transparency and system interoperability to meet the required criteria of an SOA. To address the first ‘finding the server’ problem threatening location transparency, Web Services use
the ‘Universal Discovery, Description, and Identification’ (UDDI) as a directory standard (just like clients and servers on the web all use the Domain Name Service (DNS)). All the more, Web Services use the HTTP protocol, an inherent advantage as it relates to crossing firewalls; no special tunneling or bridging mechanism is needed.

Gartner predicts that “through 2008, SOA and Web Services will be implemented together in more than 75 percent of new SOA or Web Services projects (0.7 probability).” [21]

7.2.4 Impact on Current ASPs

SOS can be considered an evolution of the traditional application management outsourcing model to a full one-to-many ASPs offering, whereby the same application running on the ASP’s platform is shared by many clients. The ASP would be characterized by pre-engineering the solution before offering it on a rental basis to many clients with limited or no investment and customization requirements on their behalf.

The age-old problem has been, however, that this solution cannot easily enable the customer/enterprise to differentiate itself in the marketplace. Accordingly, this model is best suited for horizontal applications such as calendaring or email, or for niche applications designed for a highly specific vertical market, yet not closely integrated with core and differentiating processes and applications of the enterprise. [6]

When it comes to Web Services, ASPs have been providing services over the web to end-users for many years (dating back to 1997 [2]). In fact, ASPs even provided APIs for parts of their services such as provisioning, security, billing, etc… to consumers. And if one is to believe in the potential for a Web Services Provider (WSP) business model, then he/she also has to believe in that of the
ASP, as there is little difference (of significance, nonetheless) between the two models. (The reader is first advised to skim through Appendix VIII: Executive Summary on ASP Industry, p.156, compiled from different sources to provide a succinct overview of the industry's characteristics, benefits, evaluation metrics, and market estimations.)

The ASP market may benefit from the use of Web Services in reducing the cost of creating and operating their own provisioned services. Nevertheless, given that the full componentization of applications is still in its infancy, it will be some time before Web Services are exhaustively utilized. Even more substantial than the technical barriers are those relating to who will buy the services, how to evaluate them, and how to price, market, and sell them... all of which are concerns extensively addressed, as applied to miSchedule, in the last chapter of this thesis.

7.3 Web Services: The Technology and Beyond

7.3.1 No Misnomer

When I first heard of the term 'Web Services', I actually didn't realize I heard it. For I thought I had already come across the term a hundred and one times before. Nothing struck me as peculiar: neither 'Web' nor 'Services'. And perhaps the whole notion would have been much easier to come to grips with had a term less ambiguous, misleading, and misused than 'Services' been used. But then again, I cannot think of a name that is as short, crisp, and to the point: Web Services are software constructs that expose business functionality as a 'Service' over the 'Web'.

In the context of a Web Service, 'exposing' would mean starting with: (1) identifying valuable reusable business processes, then (2) defining service-oriented interfaces to those processes, and finally (3) describing and publishing those interfaces in a standardized web-based format.
Those Service-Oriented Interfaces (SOIs) would fit into the foregoing SOA by defining systems in terms of reusable business services rather than business data, so that later changes in the data do not require changes in existing or interconnected systems. Usually, smaller units of functionality are recombined into several different larger business applications. Examples of such units that are widely prevalent nowadays would be a stock-quote provider, local weather or traffic update, shipping-rate calculator, delivery package locator, credit-card authorization... among many others [3].

As for describing the SOIs, interfaces are expressed in an industry-standard XML-based format called the Web Services Description Language (WSDL). A complete description of the operations available, parameters required, and means to bind to the exposed services is supported by WSDL. This description is then published to a repository that conforms to the UDDI (Universal Description, Discovery, and Integration), which can be queried by a client searching for an appropriate service based on a category or depending on the application.

7.3.2 Defining Characteristics

The easiest way to discern what can Web Services truly bring (and do not bring) is to enumerate their key distinctive attributes. Web Services are:

- **Platform-independent** (maximizing and building on the diversity of potential consumers)

- **Language-independent** (minimizing the implementation constraints on potential consumers)

- **Self-contained** (delivering entire expected functionality with no or minimal external dependencies)
- **Self-describing** (via the WSDL format to enable autonomous ‘binding’ when consumed)

- **Self-publicizing** (via the UDDI registry to facilitate advertising, discovering, contracting, and integrating the service)

- **Standard-based** (signifying wide adoption and reducing vendor-specific dependency and monopoly; no ‘vendor lock-in’)

- **Loosely-coupled** (allowing system interoperability yet with sufficient independence; changes in one part would not necessarily entail changes elsewhere; end-points are ‘black boxes’ where only interface signatures need be preserved)

- **Function-oriented** (performing a process in a result-oriented fashion; this is in sharp contrast with the objected-oriented paradigm they’re frequently yet mistakenly associated with)

- **Modular** (can be easily mixed-and-matched with other modular components to deliver real business value)

But how does this differentiate Web Services from ancestral technologies such as COM, COM+, DCOM (Distributed Component Object Model), CORBA (Common Object Request Broker Architecture), and the like that have claimed and promoted the same characteristics? Yes, all the latter also componentize software making it self-contained and described using interfaces/APIs. But these components are based on the underlying technology and, although resemble building blocks that can fit into a specific system or framework, they are not universal blocks like Web Services. It is somehow the difference between jigsaw puzzles and LEGO: while the parts of the former are fitted to form the predefined picture, the latter removes the angularity of those parts with new blocks that
would readily snap together with much less effort and time to make a robot (*not only* that preconceived by the vendor).

### 7.3.3 Potential Markets

According to Gartner [1], Web Services will be involved in more than 40 percent of the revenue growth in IT-related markets through 2006 (0.6 probability), including incremental growth in already existing markets and catalyzing some models that are in decline such as the ASP’s. To answer the critical question of how to make money from Web Services, the different potential markets have to be identified. Each player in an SOA market can play one (or more) of three roles [23]:

- **Service provider** publishing applications or business processes as Web Services along with their interface requirements to a service broker. The service provider is thus the *owner* of the Web Service.

- **Service broker** registering and categorizing published services and providing search services enabling requestors to locate, bind to, and invoke services. The service broker is thus the *promoter* of the Web Service and serves as an intermediary between the other two players. (Of course this presumes a provider-requestor relationship via a broker; the requestor can always have a direct pre-established relationship with the provider).

- **Service requester** using broker services to locate and consume available Web Services that fulfill a certain business need.
Gartner predicts that by the end of 2005 "new licenses for software that uses Web Services standards will represent a $21 billion market (0.7 probability) [1]. These markets are the best-established and are dominated by the major platform vendors". To name a few, representative vendors would be BEA with WebLogic, HP with NetAction, IBM with WebSphere, Microsoft with .NET, and Sun with ONE.

In the Web Services provisioning market (personified in the service providers role), new opportunities will arise for companies – even individuals – to supply business processes internally and/or over the web. ASPs will seek out this market leverage their business model or build a new one, especially in the area of commodity Web Services such as reservations, package locators, and most importantly miSchedule’s calendaring services (miServices) that are discussed in-depth in the following two chapters. Note that Web Services alone are not a new delivery model, yet they do have the potential of becoming the basis of on-demand delivery of application functionality and the foundation for the next-generation SOS models preached in the beginning on this chapter. Even former application vendors (e.g. SAP, Siebel, and PeopleSoft) will provide an increasing amount of their applications functionality in a service-based model. And given the relative ease with which Web Services can be gathered by aggregators, even enterprises are expected to redefine and sell themselves as Web Services.
Brokering of preexisting Web Services, however, will provide a model for an enterprise that chooses to become an indirect supplier channel. Service brokers will not create services, but rather focus on building a large supplier network and mine the network to generate incremental revenue by connecting the service requester and supplier for a given fee. Garner Group expect that commodity Web Services suppliers "will deliver more than 70 percent of their services through a Web Services broker by 2006 (0.7 probability)" [1]. Although some of the potential value that can be offered by brokers is diluted by the inherent dynamic discovery capability of Web Services (via UDDI), the main value that a broker is likely to provide is that of handling the billing and collection between the service requester and provider [3].

Like in all integration projects, the need for professional consulting services will continue to be necessary. Along with the professional service arms of platform vendors (e.g. IBM Global Services), Gartner Group believes service providers like Accenture and PricewaterhouseCoopers will typify the companies providing Web Services consulting and integration.

7.3.4 Pricing Models

The pricing model for Web Services at present is arguably still in its embryonic stage with most people and businesses still taken by the technical hype of this new technology. It is abundantly clear, however, that there will be a very wide spectrum of schemes ranging from freeware to premium offerings characterized by many permutations as to how pricing will be structured. The options available are likely to include one-time charge schemes, periodic licensing, and umpteen usage-based options. The following figure highlights some of the possible models depending on who is offering the service [3]:
Taking a more methodical approach to pricing a service provider's product, it must be offered at sufficiently low cost to maintain a market but high enough to allow a preferably quick Return on Investment (ROI) and positive cash flow. Every component of cost and revenue for hosting the serviced application/process has to be considered. The cost, which varies with the complexity of the serviced application, the hosting SOA, and the level of support stipulated in the Service Level Agreement (SLA), can be divided into 3 major components:

- **The software component** accounting for licenses (if any) bought from Independent Software Vendors (ISVs) and Web Service platform providers, for infrastructure software, for network/server monitoring tools, etc. Alternatively, if software is developed in-house, this component would also cover the total cost incurred for the software development.
- **The professional services component** catering for usually one-time migration, integration, installation, and customization costs that are
dependent on the consulting and labor input needed to implement a given service.

- **The hosting component** derived from the labor costs, cabinet costs, depreciation of equipment, costs of bandwidth, monitoring costs, etc... This component varies widely with the nature of the service, number of subscribing clients, and terms of the binding SLA [2].

In terms of revenue, the service provider would have 2 main sources of earnings: (1) the periodic usage-based revenue collected per client, and (2) the initial installation fees which cover the integration, customization, and possibly some professional services (in the case of extensive customization or migration from legacy systems). This latter component could also cover training and educational costs. As for the periodic revenue, it can be either or a combination of:

- **Monthly flat rate** that would cater for all costs of software, upgrades, connectivity, monitoring, etc... This structure is suitable if the usage trend can be pre-specified easily and represents a feasible calculation basis. It also makes it easy for the service consumer to calculate their expenses and handle cash flow planning.

- **Variable use-dependent rate** preluded by low or no upfront cost. The rate is based on transaction volume or storage capacity to represent a revenue- and risk-sharing model used by service providers to link the ultimate pricing of their services to the success (and failure) of the serviced customer.

Usually a combination of the fixed and variable rate models is negotiated with the customer, for both parties favor predictable costs and revenue streams. Practicable measurements to obtain adequate pricing for usage-dependent fees are yet to be discovered; IBM is currently paving the way in analyzing complex
usage-based pricing models [23]. The quest for viable WSP pricing models is continued in 9.4 Revenue Streams and Pricing, p.122, as applied to the feasible options available for calendaring services provisioning.
Chapter 8  System Web Service APIs

8.1 Offered Services

miServices is the set of Web Services that system clients can consume to communicate and manage their events and event-related user preferences in an automated system-to-system fashion. Five main miServices categories are offered to web-based systems and 3rd party clients (any system or website that hosts a calendar and/or sends events to end users, customers, students, patrons, etc...):

- a ‘register’ miService (RegisterUsers) enabling authorized sites/systems to seamlessly register their users with miSchedule. Users will have a special registration status that differs from those registering directly with miSchedule. That is, their calendars will be receiving events that could be retrieved for viewing on miServices consumer's website; however, they'll be prompted to register formally the first time they attempt to log into miSchedule or use any other miSchedule client such as miPortal or miAddin.

- a ‘create miList’ miService (CreateMiList) enabling authorized systems to compile/store a list of emails at miSchedule for quick access when sending new events to a class or customer list.

- a ‘send event’ (SendEvent) miService enabling authorized systems/clients to send events to their users via miSchedule (by addressing the event to specified recipients and/or to a preset miList registered using the previous miService). Refer to System Process 6 to see how miServer handles a new event sent via this miService.
- a ‘get events’ (GetEvents, 2 overloads, and GetMiEvents) miService enabling authorized systems/clients to retrieve events from miSchedule (such events can constitute all events pushed by the site, hence making for a public all-inclusive calendar for site surfers; and/or constitute all events pushed by the site to a specific user, hence making for the feature of conveniently viewing part of his/her miCalendar at the miServices-enabled site)

- a ‘get link’ miService (GetMiLink) enabling authorized systems/clients to register events and retrieve the self-describing miSchedule hyperlink (miLink). The miLink can now be posted on a website or sent in an email. Surfers or email recipients benefit from the automatic logging of the event in their miSchedule calendar upon clicking the miLink.

### 8.2 miServices SDK

The miServices SDK consists of the Web Services API to miSchedule (concisely described above) and the accompanying documentation for consumers to self-implement miServices. The SDK is offered in the form of an intuitive web interface and an HTML Help compiled file (displayed below):
miSchedule is an idiocentric portable scheduling service. The miServices class is exposed to authorized system clients to manage their events and users.

**Classes**

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global</td>
<td>Summary description for Global.</td>
</tr>
<tr>
<td>miEvent</td>
<td>Class definition of miSchedule event returned to miServices user via GetEvents() and GetMILink() web methods.</td>
</tr>
<tr>
<td>miServices</td>
<td>miServices are a set of web methods used by authorized system clients to manage their miSchedule events and users.</td>
</tr>
</tbody>
</table>

Below is an internal overview (private and public methods; only visible on miSchedule end) of miServices:
The private methods (IsValidEmailAddress, IsValidPassword, IsValidMiListName) are server-side methods shared by many of the services to validate parameters passed by consumers through the public web methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateMiList</td>
<td>Creates a new miList with new name (miListName) and consisting of valid email addresses.</td>
</tr>
<tr>
<td>Dispose</td>
<td>Overloaded. Releases all resources used by the MarshalByValueComponent.</td>
</tr>
<tr>
<td>Equals</td>
<td>Determines whether the specified Object is equal to the current Object.</td>
</tr>
<tr>
<td>GetEvents</td>
<td>Overloaded. Returns an array of miEvent objects comprising all events in user's miCalendar (UserEmail) sent by authenticated miServices consumer (EmailID/PartialPassword) and falling within specified date range (StartDate-EndDate).</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GetHashCode</td>
<td>Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.</td>
</tr>
<tr>
<td><em>GetMiEvents</em></td>
<td>Returns an array of miEvent objects comprising all events created by authenticated miServices consumer (EmailID/PartialPassword) and falling within specified date range (StartDate-EndDate).</td>
</tr>
<tr>
<td><em>GetMiLink</em></td>
<td>Retrieves a miLink for a properly qualified miEvent object.</td>
</tr>
<tr>
<td><em>GetService</em> (inherited from MarshalByValueComponent)</td>
<td>Gets the implementer of the IServiceProvider.</td>
</tr>
<tr>
<td><em>GetType</em> (inherited from Object)</td>
<td>Gets the Type of the current instance.</td>
</tr>
<tr>
<td><em>RegisterUsers</em></td>
<td>Registers a list of users with valid email addresses with miSchedule; registration status would allow miServices consumer to add/retrieve events to/from registrant's miCalendar.</td>
</tr>
<tr>
<td><em>SendEvent</em></td>
<td>Sends a properly qualified miEvent object to specified miList(s) and/or recipients email addresses.</td>
</tr>
</tbody>
</table>

Table 4 miServices Public Instance Methods

To facilitate the implementation job of the miServices consumer, miServices return detailed status flags to confirm the success or failure of a given call. For example, if for some reason the SendEvent method fails, the SDK would offer invaluable help on why and what to do for a given return value:
Sends a properly qualified miEvent object to specified miList(s) and/or recipients email addresses.

```java
public int SendEvent(
    string EmailID,
    string PartialPassword,
    miEvent EventToSend,
    bool AddToMiCalendar
);
```

Parameters

- **EmailID**
  - Email address with which miServices consumer has registered with miSchedule

- **PartialPassword**
  - The first 4 letters of the password with which miServices consumer has registered with miSchedule

- **EventToSend**
  - Fully qualified event to send to RecipientMiLists and/or RecipientEmails

- **AddToMiCalendar**
  - Sets whether new Event is added to miServices consumer’s calendar as well

Return Value

SendEvent() returns one of the following integer values:

- 0 - Successful sending event
- 1 - Successful sending event (forwarded identical old event; exact match found)
- 2 - Cannot authenticate - Invalid EmailID/PartialPassword combo
- 3 - Error in event properties: Subject missing
- 4 - Error in event properties: Invalid StartDate/EndDateTime value and/or order
- 5 - Error in event properties: No recipients have been specified. You need to provide at least one RecipientEmails or RecipientMiLists
- 6 - Error in event properties: One or more of the specified RecipientMiLists is/are invalid or non-existent
- 7 - Error in event properties: One or more of the specified RecipientEmails is/are invalid

Listing 29 miServices.SendEvent Web Method Signature as Provided in SDK

8.3 Granularity in Interface Design

The most important goal of high-level interface design for Web Services is to increase request granularity. That is, a Web Services must provide more value and pass more information in a single response-request over the Web. Sending and receiving more information in a single request is much more efficient in a network environment than sending many fine-grained messages that result in
increased network traffic and make error handling more difficult. In fact, this is one of the most common mistakes made with CORBA and COM programming: designing distributed objects as thought they were the same as local objects, hence resulting in too much (mostly unnecessary) network interaction [14].

A top design consideration taken into account while developing miServices was creating high-level coarse-grained interfaces that implement a complete transaction or business process. Take, for example, the service used by miServices clients to create a miList with miSchedule:

**Parameters**

- **EmailID**
  Email address with which miServices consumer has registered with miSchedule

- **PartialPassword**
  The first 4 letters of the password with which miServices consumer has registered with miSchedule

- **miListName**
  Name for new miList (case-insensitive). Can contain 'a-z', 'A-Z', '0-9', and '_' characters only (no '@' or white spaces)

- **Emails**
  Email addresses of users that miServices consumer wishes to register with miSchedule. Those addresses will become their miSchedule EmailIDs

- **OverrideIfExists**
  Sets whether old miList with similar miListName should be overridden or not. True to override

- **RegisterIfNew**
  Sets whether an unrecognized user in Emails[] should be registered with miSchedule. If set to true, RegisterUsers() is called on new user(s) with the specified options: NotifyAboutMiSchedule and GiveFullPermission. (The latter 2 options are neglected if RegisterIfNew is set to false)

- **NotifyAboutMiSchedule**
  Sets whether miSchedule should notify new users of their registration status and the miSchedule service. (Neglected if RegisterIfNew is set to false)

- **GiveFullPermission**
  Sets the permission level in new users' (Emails) miSchedule accounts to have new events sent by registering miServices consumer automatically logged into their calendars. (Neglected if RegisterIfNew is set to false)

**Return Value**

`CreateMiList()` returns one of the following integer values:

- 0 - Successful miList creation
- 1 - Cannot authenticate - Invalid EmailID/PartialPassword combo
- 2 - Invalid new miList name - New miList name can only contain 'a-z', 'A-Z', '0-9', and '_' characters; '@' sign and spaces are not allowed
- 3 - Error in Emails: One or more of the specified emails is/are invalid
- 4 - miList with a name similar to miListName already exists. Either set OverrideIfExists to true, or choose a new miListName

**Listing 30** miServices.CreateMiList Web Method Signature as Provided in SDK
In one method, miServices clients can: (1) register new users, (2) notify them about miSchedule, (3) and include them in a new miList that can override an existing one with a similar name.

In other words, instead of the miServices client having to register new users before hand, handle a similar miList name conflict error, etc... via different methods, miServices were designed with the initial intent of providing the client with access to a complete service, rather than getting and setting a specific data value. Enabling businesses communication to take place via a single message exchange is the best way to design Web Service interfaces.
Chapter 9  Outsourcing Calendaring and Event Management

9.1 Evaluating miServices

9.1.1 Tangible Benefits

Among the current trends in business is the trend to focus on core capabilities and the trend to access rather than own capabilities or processes peripheral to the core. And what process could be as peripheral to most businesses as calendaring, notification, and event management? The five basic service categories elaborately discussed in the previous chapter's 8.1 Offered Services, p.105, translate into the following tangible benefits to miServices consumers and their respective customers:

- miServices consumer need not worry about storing corporate events and customer information/preferences relating to that internally anymore, as miSchedule would be handling all this.

- miServices system client can seamlessly register its corporate clients with miSchedule as recipients to its own events by incorporating the miSchedule register service into its system processes and user registration form.

- miServices consumer can programmatically send events to customers using the send event miService, and guarantee that it would reach all recipients at their one preferred calendar.

- When clients are browsing a miServices consumer's site, it can always offer them to view its events (those in their miCalendar) by programmatically using the miSchedule get events service to retrieve all events in a given recipient's
calendar sent by the miServices consumer, and have them displayed internally (at consumer's website) using its own custom calendar view.

- miServices consumer can miSchedule-enable any event (make it a miLink) on its website granting users the convenience of clicking the announced event and have it directly saved into their centralized calendars.

- miServices consumer would be offering its clients or users the additional satisfaction of reaching them at their favorite centralized calendar, without forcing the customer to separately track his/her events at the consumer's own site.

9.1.2 Assessment Framework

To help the potential miServices client evaluate and decide on the right Web Services to consume, Forrester Research [12] offer the following framework (adapted):

| Criteria | Questions | Score
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1. Recurring</strong></td>
<td>Are there more than 10 users?</td>
<td>1 = Yes, 0 = No</td>
</tr>
<tr>
<td>Syndicated and repeated across a large base of users</td>
<td>Will they use it at least once a day?</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Are the underlying elements the same for all?</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Does the info change more than once a day?</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Does the info feed into a specific user process?</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Does the user benefit from tighter update schedule?</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Is the communication of this information manual?</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Does the user have the application that this information feeds?</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Is the user unwilling to invest in EDI or integration?</td>
<td></td>
</tr>
<tr>
<td><strong>2. Dynamic</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Syndicated and repeated across a large base of users</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>3. Disconnected</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Syndicated and repeated across a large base of users</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Target level of Web Service deployment</strong></td>
<td>0 to 3</td>
<td>4 to 6</td>
</tr>
<tr>
<td>Do not consume</td>
<td>Pilot</td>
<td>Consume</td>
</tr>
</tbody>
</table>

Table 5 Web Service Opportunities Must Meet Three Criteria [12]
It can be easily seen how, due to the nature of calendaring, the specified criteria match or are in alignment with the miServices offering, namely in the case of its *Recurring* and *Dynamic* attributes: all employees, customers, students, patrons, users etc… are offered their schedules in the same way and probably more than once a day, and the information changes frequently and end-users would benefit from regular updates.

### 9.1.3 Comparing Current Calendaring Technologies

The figure below shows a detailed comparative analysis of the features differentiating current web-based calendaring solutions. While portal calendars such as those offered by MSN and Yahoo! offer a very intuitive interface and featureset to end-users, they lack the system serviceability component. On the other hand, Infotriever ([www.infotriever.com](http://www.infotriever.com)) offers supreme compatibility with different PIMs and devices, yet it’s mainly targeted at calendar synchronization with neither offering users a true PIM replacement nor businesses the full-fledged event management outsourcing solution offered by miSchedule:
Figure 26: Distinctive Aspects of Different Calendaring Technologies

(Note that Infotriever's assessment has been solely based on the information available at the company's website, which does not offer the necessary details about their proprietary technology that would make this comparison wholesome and fully justified.)
9.2 What happened to .NET My Services?

9.2.1 Initiative Introduced

At its Professional Developer Conference (PDC) in October 2001, Microsoft introduced one of the first and most comprehensive Web Service initiatives to date [10]: .NET My Services (also known as ‘HailStorm’), a collection of services aimed at giving the end-user unprecedented control over his/her identity and personal data from arguably any device or computing environment. These services were aimed at e-commerce sites and other online businesses; they provided basic building blocks that would snap together forming 3rd-party solutions (Windows-/Web-based) centered on managing the personal information of end-consumers. One of the services (see exhaustive list in Appendix VII: Listing of .NET My Services, p.155), .NET Calendar, is believed to have promoted the same notion of user-centricity and service-based calendaring as miSchedule, should it have actually passed the drawing board and been rolled out by Microsoft. But why didn’t it?

9.2.2 Initiative Remodeled

According to Adam Sohn, product manager for .NET platform strategy, “Microsoft quickly learned that the [business] model didn’t appeal to most of its enterprise customers, mainly because they didn’t want to have their customer data stored with MSN... [they] asked Microsoft to offer a packaged version of the technology that they could purchase to build and host services on their own. Examples of possible operators of such services included corporations that would create these Web Services for their employees, as well as Internet service providers and Web portals that could offer services to their subscribers.” [9] – Bottom line: It wasn’t the technology so much as the business model. The concern in industry, besides security and trust, was that companies don’t necessarily want to give up control of their customers’ sensitive information to a central source [8].
Accordingly, while Microsoft recasts .NET My Services into a technology that corporations or service providers will control and manage independent of Microsoft (host at their own premises), miSchedule has learned the lesson. A new miLicense offering would be targeting a thin market segment with a fully modular ‘light’ version of miSchedule that customers can deploy and host independently. The peculiarities and pricing scheme of this offering are described and contrasted with those of miServices in the 9.4 Revenue Streams and Pricing, p.122, of this chapter.

9.3 Viable e-Business Model

9.3.1 Model Semantics

In [25], Weill and Vitale define an e-business model as “a description of the roles and relationships among a firm’s consumers, customers, allies, and suppliers that identifies the major flow of product, information, and money, and the major benefits to participants”. Their e-business model semantics have been slightly adapted to the case of miSchedule as depicted in the following figure (see legend):
Note that the use of the term *Premium* with the different entities is to signify substantial versatility in the offering that it can be correlative packaged and priced.

**9.3.2 Ownership of Customer Data and Relationship**

The fine distinctions in relationships with end-user and flows of product, money, and information are critical in analyzing each of the different revenue channels they are associated with.

In all channels, except in that of the miLicense offering, miSchedule owns *all* customer information (as related to events, personal information, preferences, and credentials) and *can* own the customer relationship as well. Perhaps the only case where miSchedule might remain transparent to end-users and host no relationship would be in the case of interacting with Premium miServices clients who could unbrand and fully customize their front-end use of miServices (see *9.4.1 Web Services Implementation, p.123 for more details*). An end-user to end-user relationship (flow of information), as depicted in Figure 27 (above), can exist amidst members in the virtual community tracking the Public miSchedule Calendar and participating in forum discussions and/or event threads.

**9.3.3 Market Segmentation**

To start with, let us partition the targeted market of system clients according to the user-base they wish to serve: corporate portals, business-to-employee (b2e) intranet portals, business-to-business (b2b) extranet portals, business-to-consumer (b2c) e-commerce portals, or self-service portals. The term ‘portal’ is used to refer to the system’s on-line presence because of the core functions a portal is expected to provide (aggregation, personalization, search, collaboration, security, etc...), setting it apart from a regular ‘website’.
The basic taxonomy of portals showing the separation of corporate portals from public Internet portals is displayed below [3]:

Figure 28 Basic Portal Taxonomy

There will always be, however, functional commonality across portals no matter what they are called or in what category they fall [3], and calendaring is one of those commodity functions. It is important to appreciate this commonality, since corporate portals, namely the new generation of XML-centric and Web Services-enabled ones will begin to consolidate what were previously positioned as different portal types into a single unified entity; hence exploiting authentication-bases personalization. For example, one of the first system clients attempting to implement miServices was a course portal supporting both students and more privileged users such as administrators, faculty, and research/teaching assistants. The latter category, once authenticated, had the 'send event' and 'register user' miServices displayed/enabled, whereas regular students would only have the portal trigger the 'get events' miService on their behalf to display their calendars.
The portal partitioning was primarily conducted to show what potential systems are more likely to implement miServices as opposed to those who would favor the miLicense offering. For example, public portals and the public portions of corporate portals might simply want to implement the generic ‘get events’ miService for all events it sent out this past month and have them similarly displayed to all visitors. If the portal recognizes the user, perhaps it would offer the added convenience of triggering the user-specific ‘get events’ service to retrieve events as he/she has them managed in his miCalendar (including updates, deletions, annotations, permissions to retrieve events from other sources, etc...). On the other hand, a corporate portal wishing to implement an effective collaborative solution to serve its employees and/or partners (intranet and/or extranet), miLicensing might seem much more suitable: the corporate client would fully deploy, host, and manage a modular miSchedule solution internally with full customization and storage capabilities.

Another conceivable implementation of miLicensing could be in the case of a Vertical Service Provider (VSP) wishing to license the miSchedule technology and build on it to provide a more accommodating collaborative solution to its vertical industry. For example, imagine a complex patient scheduling solution for the healthcare industry: the resulting rule-based system would have to find and book appointments, eliminate conflicts between different healthcare facilities, provide the patient with appointment reminders, etc... As organizations try to maximize their resources, patient scheduling conflicts remain huge administrative problems for many healthcare providers and miSchedule would be providing a viable solution.

**9.4 Revenue Streams and Pricing**

The possible pricing schemes for miLicensing and miServices implementation, among other revenue channels, are now discussed in more detail:
9.4.1 Web Services Implementation

Continuing from the discussion in 7.3.4 Pricing Models, p.101, the miServices client can pay on a per-transaction basis (very small fee per transaction) that adds to up to some reasonable pay-for-usage cost [15]. In other words, miSchedule would be tying its revenue to the success of the client; the more traffic there is, and assuming proportional calendar requests, the more pay-for-usage revenue miSchedule would earn. A better-established and easier to deploy model is subscription based pricing. The miServices consumer would pay a monthly fee for access to a set of services. A parameter easier and more reasonable to incorporate to track service consumption than transactions would be storage. That is, the variable pay-for-usage cost can incorporate total events owned + total users registered + total miLists created by a given system and have it charged accordingly instead of tracking service calls. This will be easier to implement and monitor from a technical standpoint. That said, the analogy between usage-based micorpayments versus subscriptions is long-distance phone service as compared to local phone service (as implemented in the US); the former offers a consumption-based model, whereas the latter offers unlimited use for a fixed period of time.

The premium as compared to regular offering is related to miSchedule’s branding on the system client’s front-end: the miServices consumer would have the option of either being restricted to using miSchedule-designed controls (with miLogo) with event links bringing the user back to miPortal for event details, or unbranded interaction making miSchedule completely transparent to the end-user.

Another pricing factor would be the binding SLA between miSchedule and miServices consumers. The SLA may range from anything as simple as specifying a minimum and maximum response time, to something as complex as having different service guarantees for different pricing levels that would differentiate its premium from regular consumers. Those who opt to pay for a premium SLA would pay a higher price and in return get, for instance,
guaranteed throughput, response time, and/or up time. They might also want to monitor the service in some way to ensure they're getting the agreed upon premium level of service.

9.4.2 Licensing the System

A licensed miSchedule would consist of a deployable lighter version of miSchedule (stripped of miServices, for instance) that would be owned, installed, hosted, and managed in fully by the licensee. The value proposition would be full front-end customization of all miSchedule components, no storage capacity limit, ownership of customer relationship, and most importantly, ownership of sensitive customer information. miSchedule would be placing a lock on the number of total registered users, however. That is, the pricing plan would be based on a per-number-of-users basis. Other fees might consist of professional services provided by miSchedule Consulting & Integration (see below) for system installation, customization, and basic training. Other long term variables would be those related to system maintenance, support, and upgrades.

9.4.3 Consulting and Integration

Revenues in this component accrue from miSchedule implementing client-side miServices for a customer (i.e. developing the client-side controls and binding with the Web Services for the customer), and from any other deployment, installation, integration, migration, customization, and/or support services for miLicense and miServices clients. Perhaps this branch can also, at some point in time, develop the capability to undertake vertical or industry-specific rollouts of miSchedule such as those discussed towards the end of 9.3.3 Market Segmentation, p.120.
9.4.4 Premium End-Usage

Of course, all other components of miSchedule (miPortal, miAddin, miTags, etc...) are offered to end-users with full functionality free of charge. However, there are quotas on total events owned, total miLists created, and other storage-variable features that would otherwise be open-endedly generated by users. Just like many other free services offered nowadays (MSN Hotmail, for example) a minimal fee will be charged to premium users wishing to increase their storage limits and receive advanced features, such as miCalendar import, and an advertisement-free miPortal and miAddin.

9.4.5 SOR Derived from Information Ownership

miSchedule owns a huge informational asset. Event categories, times, locations, and body keywords with highest percentages in a given user’s miCalendar characterize his/her event usage trends. For any given user, miSchedule knows what’s the category with most events in his/her calendar, where are they taking place, when, etc... and advertisement listings can be adapted accordingly. Such trends, coupled with other user profile information, provide very valuable information for marketers and targeted advertisement. Besides users’ event usage trends and characteristics, miSchedule knows how ‘popular’ a given event is by knowing how many users have it added to their miCalendars and/or how much traffic the event is generating; banner advertisements presented to users as they browse very popular events are reserved for proportionately premium advertisers. Other information-based revenues can be accrued from targeted email campaigns to users indicating willingness to receive such promotions and from the consented resale of event trends and usage characteristics.
References

1) D. Plummer, Gartner Research, “Key Entry Points Into Web Services Markets” (February 2001)
2) T. Falkowski, “Feasibility Study on the Use of the ASP Business Model for Enterprise Application Software” (March 2002)
4) D. Plummer, M. Blosch, and R. Woolfe, Gartner Research, “Executive Summary: Untangling Web Services” (March 2002)
14) M. Rosen and J. Parodi, Iona Technologies, PLC, “Architecting Web Services” (December 2001)


19) S. McConell, “Rapid Development” (June 1996)

20) D. McCoy and Y. Natis, Gartner Research “Service-Oriented Architecture: Mainstream Ahead” (April 2003)


22) G. Flurry, IBM, “Applying Web Services to the Application Service Provider Environment” (January 2001)

23) S. Burbeck, IBM, “The Tao of e-Business Services” (October 2000)

24) S. Pathack, eAI Journal, “The Promises of Web Services” (December 2001)

25) P. Weill and M. Vitale, “Place to Space” (2001)

Appendix I: Key System Process Diagrams

The following are six key miSchedule system processes (Figure 29, p.129, to Figure 34, p.134) depicting the handling of various triggers by different clients and use case scenarios. The processes listed below are described in-full at each process diagram:

SP1: Event Dispatching

SP2: miLink Generation

SP3: Event Adding via miLink

SP4: miTags Handling

SP5: miEvents Management

SP6: New Event via miServices
System Process 1: Event Dispatching

Illustrates the dispatching of an event (new or old) all the way from authenticating event sender, to ensuring event integrity, and to delivering event to intended recipients via their means of preference.

Figure 29 SP1: Event Dispatching
System Process 2: Event miLink Generation

- **Start**
- System Receives Request for an Event miLink
  - **Yes** Authenticate User
  - **No** SP2-Err1
  - SP2-Err2
  - Check Required Event Attributes
  - **Yes** Missing
  - New Event
  - Register New Event in miSchedule DB
  - Error Occurred (Exception Handling)
  - Generate and Return miLink to Requesting Entity
  - **End**

Figure 30 SP2: Event miLink Generation
Event Adding via miLink

Illustrates the detailed process of adding a new event into a user's calendar upon recipient clicking an email- or webpage-contained miLink.

**System Process 3: Event Adding via miLink**

- **Start**
- Recipient Clicks miLink
  - **Yes**: Is the Event an Invitation?
  - **Yes**: Event sender can deem event as 'invitation', hence requiring miSchedule to prompt and track recipients' confirmation.
  - **No**: Denotes recipient's response upon being prompted by miSchedule to confirm attendance.
  - **No**: Confirm Attendance?
    - **Yes**: Send Confirmation Email to Sender and Update Invitee Status
    - **No**: Enter Login Information if Available
      - **Authenticated?**
        - **Yes**: Register?
          - **Yes**: Complete Registration
          - **No**: Store Event in miCalendar
        - **No**: Complete Registration
      - **No**: Special Registered
        - **Yes**: Check Sender's Time Zone and Adjust Event Time
          - **Yes**: Store Event in miCalendar
          - **No**: End

**Figure 31 SP3: Event Adding via miLink**
System Process 4: miTags Handling

Illustrates the system handling/parsing of an event miTags upon receipt of a miTagged email.

Start

System Receives miTagged Email

Email Body Parsed for miTags

Check if there is a Parsing Error

Yes

Error Occurred (Exception Handling)

SP4-Err1

No

System Process 1
Events Dispatching

End

Figure 32 SP4: miTags Handling
System Process 5: miEvents Management

Illustrates how the system responds to different miEvents management requests, and the consequent processes that may result on the recipient and depending on his permission and notification preferences.

Figure 33 SP5: miEvents Management
System Process 6: New Event via miServices

Illustrates the handling of new events sent to the system via miServices users. New miSchedule user registration and notification are services optionally made implicit to SP6.

Start

System Receives Request to Add an Event via miServices

Authenticated?

Yes

Register New Recipients?

Yes

miServices sender can choose whether unrecognized recipients be automatically registered with miSchedule (special registrations status)

No

Iterate Over Recipients

Yes

New?

Yes

Special Register

A special registered user will have an account with miSchedule that a miServices user can send/retrieve events to/from, however, user has to complete registration to use miPortal and enjoy full miSchedule functionality.

No

Notify About Special Registration?

Yes

Send Notification Email About Special Registration

No

System Process 1

Events Dispatching

End

Figure 34 SP6: New Event via miServices
Appendix II: miTerms (miSchedule Nomenclature)

What is miSchedule?
miSchedule is an idiocentric portable scheduling service initiative dedicated at taking event management and calendaring to the next level. The end-objectives of miSchedule are: (1) achieving true calendar centralization, (2) automating the formerly manual event logging and porting process, and (3) offering all this as an attractive service to other systems wishing to outsource event management and calendaring to a specialist.

What is miPortal?
miPortal is miSchedule’s primary web-based user interface offering rich functionality and extensive control to miSchedule users over their events. Through miPortal, users can create, view, send, receive, and manage events to the fullest. miSettings, miNotifications, miPermissions, and miLists are also fully configurable from miPortal.

What is miAddin?
miAddin is a COM Add-In for Microsoft Office XP (Outlook, FrontPage, and Word). miAddin is intended for users with private computers and with Outlook as their favorite PIM (Personal Information Manager). Of the many miAddin features, users can import subsets of their miCalendar, be notified of incoming miLinks MSN Messenger-style, insert miLinks into an email or a webpage with the click of a button, etc... miAddin offers the user a truly Office context-sensitive interaction with miSchedule.

What are miServices?
miServices are a set of Web Services offered by miSchedule on a subscription basis to system clients wishing to leverage their event management practices, maximize their reach and range, and better serve their customers/users with cutting-edge calendaring and notification enablement.

What are miTags?
miTags are a set of tags used to define a miSchedule event in text and communicate it via email. Using a very flexible tagging specification, the user can easily define and send events from any text-editor without any prior communication with miSchedule over the web. A fully-qualified miTagged email would reach miServer, who in turn would parse, validate, and then register the event before rerouting it to the intended recipient(s).

What is miCalendar?
miCalendar refers to the calendrical container of all events to a miSchedule user's hosted calendar. Those can be events created by the user (and sent or just kept for personal use) and/or events created by others that the user has added to his/her miSchedule calendar. Among many other features of miCalendar, users can view events in multiple views (monthly, weekly, daily, and list view), update events they own (cancel, postpone, and edit), and filter event views by event category and/or event sender.

What are miEvents?
miEvents refer to all events a user has created (irrespective whether they’ve been sent or simply added to miCalendar). The generic miEvent term is the name of the proprietary miSchedule-defined event with all its attributes and handling mechanisms.

What is miLink?
miLink is a special hyperlink that refers to a registered miSchedule event. Any miSchedule user, upon clicking miLink, will have this event automatically logged into his/her miCalendar. For any new event created, a corresponding miLink is provided for the creator to send in an email or post on a webpage. This would make it very convenient for everyone else interested in the event; they can save it by not more than clicking it!

What is miServer?
miserver refers to the remote backend of miSchedule that is invisible to users and clients. For example, it's at miServer that miNotifications are issued, miTags are parsed, and events are rerouted to their predefined recipients.

What is miList?
mislist is a list of email addresses a miSchedule user can group together under one name for efficient later use. Email addresses in a miList need not be those of miSchedule users, but any random addresses frequently addressed by the user as part of a larger group of recipients.

What are miSettings?
misSettings comprise a user's full range of miSchedule-related settings: miPermissions, miNotifications, miPassport, miDefaults, and miInfo.

What are miPermissions?
mipPermissions are the set of Yes/No permissions given by user A to users B and C depending on whether A wants the events they send to be automatically logged into A's miCalendar. For example, giving a school course management system a Yes permission would automatically save
all events the system sends user A into A’s miCalendar (A can toggle ‘Notify me when a new event is added to miCalendar’ to be notified of that). On the other hand, A wouldn’t want to give a mailing list to which A is subscribed this same privilege. Of course, miPermissions only work in the context of miSchedule users.

**What are miNotifications?**

miNotifications are the set of email notifications a user wants miServer to send him/her every time a certain event-related scenario occurs. A user can set whether to receive notifications if an event in his/her miCalendar is cancelled, edited, and/or postponed by its owner, if it conflicts with another overlapping event, and/or when an event is automatically added to miCalendar by another user given the appropriate level of permission.

**What are miDefaults?**

miDefaults are the set of default values a user can preset for certain event attributes that seldom change so that he/she would save time defining new miEvents and send them on the fly. A user’s contact info, for instance, that need not be written every time an event is defined; most probably he/she will be the same person to contact most of the time.
Appendix III: miFAQs (Frequently Asked Questions)

Can I fully block events from a given user?
Yes and no. You can prevent any person from automatically logging events into your miCalendar (by default). In fact, you only give permission to those from whom you would want events automatically logged. And you can always revoke permission you’ve previously given to a certain user. After denying a given sender permission, events he/she sends you would reach you in the form of emails rerouted via miSchedule or containing a miLink. That is, your control over what to block now is proportional to that over blocking email from a given email address.

What happens when I send an event?
When you send an event (not a miLink of an event) if is first registered at miServer as a new event owned by you and the rerouted to the recipients you’ve specified. Of those recipients, some might not be miSchedule users of course, and they would receive your event fully described in text in an email addressed from you; a hyperlink to a page informing them about miSchedule and inviting them to get the privileges you have is also attached. As for the recipients that are miSchedule users, those who have given you permission to auto-log events would have your event directly placed in their miCalendar (they would get accordingly notified if that has been set in their miNotifications). Users not granting you permission, however, would be receiving an email addressed from you with the event fully described in text along with a miLink for quick logging.

Is there a way for me to know what permissions other users have granted me?
Technically, you can’t. Let us know if you get around that.

What is the difference between sending a miEvent and sending a miLink of an event?
An email containing a miLink does not go through miServer; it reaches your recipients right from your outgoing mail server in the exact format you sent it. You would simply be writing a personal email and attaching a miLink for the recipients’ convenience. As for sending miEvents right away, that is best suited for contexts where a miServices system has clients regularly and autonomously informed of new events, or a teacher notifying his/her student list of the date of a recitation session. This is where the benefits auto-logging and miPermissions surface; your event-by-event approval is time-consuming and unnecessary in the case of sources you trust and events you’re always interested in.

What is the difference between a regular and a public event?
A regular event is one that is available only to users explicitly adding the event. A personal event you want to add to your miCalendar or send to a specific number of recipients should not be set as public, as it is of no concern to others. Public events, however, are searchable by anyone and could be used by miSchedule to selectively populate the miSchedule Public Calendar. Make an event public when you think the more eyeballs it gets the better; we'll see to it that it gets the attention you want.

**What is special about an event being an ‘invitation’?**
When an event is an invitation, persons (miSchedule users and others) attempting to add this event will be asked for their attendance status (attending, not attending, undecided, unresponsive). miSchedule users can then proceed to adding the event, whereas unrecognized recipients would register with miSchedule then have the event added to their miCalendar. Meanwhile, miSchedule would be tracking those responses for your and displaying them in the event view for your convenience.

**What happens if I add the same event more than once?**
If the events are identical, no new event will be added; no identical overlap. Nonetheless, any slight change (a single varying letter in the event body) would deem the other event different and make it in conflict with the original event.

**Can two events at the exact same time coexist in miCalendar?**
Definitely. You can also set the corresponding conflict notification on to know of it beforehand.

**What is the difference between the sender note and the personal note?**
You can add a personal note to any event in your miCalendar, whether you have created the event or added it. This note would simply serve as a personal remark with which you annotated some event for later review; it is only visible to you. However, you can only add a sender’s note to miEvents; that is, events you have created/sent. A sender’s note has global implications and would be reflected in any user’s calendar who has added the event. It’s your means to append a note to an event after sending it.

**When I update an event I originally sent to other users, how are those changes reflected to them?**
Depending on how you update the event, changes and notifications will be conveyed throughout. To start with, according to the miNotifications settings of users who have your event in their miCalendar, they will be accordingly notified (or not) via email of your changes. The event itself will display the corresponding alert status (and alert note if specified by you) on users’ end. The
date and time at which you made the update is also reflected, along with the actual changes or updates you made.

**Why can't I update all events in miCalendar?**
Because you haven’t created them all. You can only update events you have created. You can also add/update personal notes of any event (irrespective of who created it). And of course, you can’t update an old instance of an event you cancelled or postponed.

**What is the difference between events in miCalendar and events in miEvents Management?**
miEvents Management contains only events you created/sent, irrespective of whether you added them to miCalendar or not. miCalendar contains only events explicitly added to it by you irrespective of their sources.

**What is the difference between canceling and removing an event?**
Removing an event only removes it from your miCalendar. It is not displayed there anymore. This has no effect on the event itself or on any other user who has added it. Canceling an event, however, infiltrates throughout your miCalendar and that of all other users who have added it, with miSchedule notifying each according to his/her miNotifications preferences. Note that you cannot un-cancel an event after canceling it.

**Can anyone know what events I have added to miCalendar?**
No. But keep in mind that granting permission to a miServices system client enables it to retrieve events from your miCalendar that it at some point created and sent you. That is, at any point in time, it can technically know what of its events are still in your miCalendar.

**Can I restrict the viewing and adding of an event (not public) I create to certain people I specify?**
Your events are intended only at the recipients you originally specify. However, just like you may send someone an email that he/she can later on forward or show to a third party without your knowledge, your event can also be forwarded or simply added via its corresponding miLink. But it will always bear your identity as owner and not that of the second party who forwarded it.

**What is New Event Express (miPortal)?**
This is a quick way of creating and sending an event or retrieving its corresponding miLink without prior sign-in. You provide your miPassport along with the new event properties, and you are authenticated in the same trip to miServer.
**Figure 35** Event View After Being Postponed

**EventView.aspx** - This is the common event view showing how alerts are displayed. In this case, a postpone alert provides a link to the new event representing the postponed instance.
Welcome To miSchedule

miSchedule is neither evolutionary nor revolutionary in its approach to helping you better manage your time; it's simply... practical.

This practicality derives from miSchedule's commitment to serving you wherever you are and whatever environment you use to manage your everyday schedule. You already have a calendar? A web-enabled one? And you also use a powerful PIM (Personal Information Manager)? miSchedule is still just for you. For it's on the very day that miSchedule came to be that your calendars lost an 'a'... once and for all.

Skim through our difference services (top-right) and suit yourself with the solution that best serves your needs.

Figure 36 miSchedule Main Page

index.aspx - This is the first page users access when they point their browsers to miSchedule.mit.edu.
**Figure 37 Event View (by event owner)**

**EventView.aspx** - This is the common event view. It provides all the actions that can be performed on an event. Access to those actions is toggled based on different scenarios such as the user owning the event or having it in his/her calendar.
**Figure 38** miCalendar Monthly View

default.aspx - This is the miCalendar monthly view. The same view is used by miEvents and Public Calendar but with different design themes.
**Figure 39** miCalendar Weekly View

**weekly.aspx** - This is the miCalendar weekly view. The same view is used by miEvents and Public Calendar but with different design themes.
Figure 40 miCalendar Daily View

daily.aspx - This is the miCalendar daily view. In this view, events are optimally aligned to fit the day table. The same view is used by miEvents and Public Calendar but with different design themes.
Figure 41 miCalendar List View

list.aspx - This is the miCalendar list view. The same view is used by miEvents and Public Calendar but with different design themes.
Figure 42 New Event View

loggednewevent.aspx - This is the page where users can define events and get their milinks, add them to their calendars, or send them to recipients.
misettings.aspx - This is the page where users can view and update all their settings. These include permissions, notifications, personal information, default values and authentication information.
Public events constitute the entire set of events defined as public by all miSchedule users. You can access this events repository.

You can get a unique clickable event identifier (miLink) to send in your events by creating a "New Event" and copying/pasting the system generated URI into any of your emails and/or web pages.

Figure 44 miLists Management View

miLists.aspx - This is the page where users can define and edit miSchedule mailing lists.
Appendix V: Full miTags Description

- `<miSender>...<miSender>`: Contains the sender’s email that is identified by miSchedule (in case the sender is sending from any other address or email account). If this miTag is omitted, the system will check whether the original ‘From’ field of the email is identified. If it isn't, the event does not qualify and the system will bounce the email back to the sender notifying him of the error.

- `<miRecipients>...<miRecipients>`: Contains a comma delimited list of intended recipients’ email addresses. At least one address is required. If this miTag is omitted, the system will attempt to log the event into the sender’s miCalendar.

- `<miPassword>...<miPassword>`: This is a required field and cannot be omitted. The sender must supply the first 4 letters of his miSchedule password (so that in case the email is sniffed before reaching the system, the sender’s account would not be jeopardized).

- `<miEventSubject>...<miEventSubject>`: This is a required field and cannot be omitted. It contains the ‘Subject’ designated to the event.

- `<miEventStartDate>...<miEventStartDate>`: This is a required field and cannot be omitted. It contains the ‘Date’ on which the event takes place.

- `<miEventEndDate>...<miEventEndDate>`: This field contains the ‘Date’ on which the event ends, if omitted the ‘event’ is assumed to end on the same day specified in the miEventStartDate tag.

- `<miEventStartTime>...<miEventStartTime>`: This is a required field and cannot be omitted. It contains the ‘Start Time’ at which the event takes place.

- `<miEventEndTime>...<miEventEndTime>`: This field contains the ‘End Time’ of the event; if omitted, it is assumed this is a full-day event and it ends with the end of the day.

- `<miEventTimeZone >...<miEventTimeZone>`: This is an optional field. It contains the ‘Time Zone’ at which the event takes place. If omitted, the ‘Time Zone’ specified in the sender’s preferences will be used as default.

- `<miEventBody>...<miEventBody>`: This is an optional field. It contains the ‘Body’ of the event that the sender can use to elaborately describe the event.

- `<miEventLocation>...<miEventLocation>`: This is an optional field. It contains the ‘Location’ at which the event takes place.

- `<miEventURL>...<miEventURL>`: This is an optional field. It contains an external ‘Hyperlink’ linking the recipient to additional online coverage of the event.
<miEventCategory>...<miEventCategory>: This is an optional field. It contains the 'Category' in which the sender wants to place his event. If omitted, the event will be categorized according to the 'Default Category' preset in the sender's preferences.

<miEventContact>...<miEventContact>: This is an optional field. It contains the 'Contact Info' the sender designates to his event. If omitted, the 'Contact Info' preset in the sender's preferences/profile will be used.

<miEventIsPublic>...<miEventIsPublic>: This is an optional Yes/No field. A 'Yes' denotes the event as 'Public' and can be searched by anyone. 'Public' events will also show in public miSchedule calendar. A 'No' denotes the event as 'Private' and available to only to recipients who add the event manually or have it auto-logged into their calendars. Omitting this miTag makes the event 'Private', by default.

<miEventIsInvitation>...<miEventIsInvitation>: This is an optional Yes/No field. A 'Yes' denotes the event is an 'Invitation' and will prompt those who add it to reply to the event sender. The sender will be able to keep track of his/her invitees. Omitting this miTag makes the event a non-'Invitation', by default.
Appendix VI: GetEventInstance Method from miClasses

```csharp
static public Event GetEventInstance(int UserID, int InstanceID, out bool IsInMiCalendar)
{
    SqlConnection connection = new SqlConnection(miUser.ConnectionString);
    Event ReturnedEvent = new Event();
    try
    {
        connection.Open();
        SqlCommand command;
        //Gets the Instance Description
        command = new SqlCommand("miSchedule.miSchedule.Proc_GetInstance", connection);
        command.CommandType = CommandType.StoredProcedure;
        SqlParameter InstanceIDP = command.Parameters.Add("@InstanceID", SqlDbType.Int);
        InstanceIDP.Value = InstanceID;
        SqlParameter UserIDP = command.Parameters.Add("@UserID", SqlDbType.Int);
        UserIDP.Value = UserID;
        SqlParameter IsInMiCalendarP = command.Parameters.Add("@IsInMiCalendar", SqlDbType.Int);
        IsInMiCalendarP.Direction = System.Data.ParameterDirection.Output;
        command.ExecuteNonQuery();
        //If the user has it in his calendar or not
        if (((int)IsInMiCalendarP.Value == 0))
            IsInMiCalendar = true;
        else
            IsInMiCalendar = false;
        SqlDataReader reader = command.ExecuteReader();
        reader.Read();
        int EventIDTemp = (int)reader["EventID"]; ReturnedEvent = Event.GetEvent (EventIDTemp);
        string StrZoneE = ReturnedEvent.TimeZone;
        double EventTimeZone = Convert.ToDouble(StrZoneE.Substring(6, 2)) + (Convert.ToInt32(StrZoneE.Substring(9, 2)) / 60);
        if (StrZoneE.Substring(5, 1) == "+")
            EventTimeZone = EventTimeZone;
        else
            EventTimeZone = -1 * EventTimeZone;
        ReturnedEvent.EventInstances = new ArrayList();
        EventInstance EI = new EventInstance();
        EI.InstanceID = (int)reader["InstanceID"];
        if (IsInMiCalendar == true)
            EI.PersonalNote = (string)reader["PersonalNote"];
        EI.SenderNote = (string)reader["SenderNote"];
        EI.StartDateTime = (DateTime)reader["StartDateTime"];
EI.EndDateTime = (DateTime)reader["EndDateTime"];  
reader.Close();  

double UserTimeZone = EventTimeZone;  
if (IsInMiCalendar == true)  
{  
  // Gets User Time Zone  
  SqlCommand command = new SqlCommand("miSchedule.miSchedule.Proc_GetUserTimeZone", connection);  
  command.CommandType = CommandType.StoredProcedure;  
  SqlParameter UserIDTP = command.Parameters.Add("@UserID", SqlDbType.Int);  
  UserIDTP.Value = UserID;  
  SqlParameter TimeZoneOutP = command.Parameters.Add("@TimeZoneOut", SqlDbType.VarChar);  
  TimeZoneOutP.Direction = ParameterDirection.Output;  
  TimeZoneOutP.Size = 50;  
  command.ExecuteNonQuery();  
  string StrZone = TimeZoneOutP.Value.ToString();  
  UserTimeZone = Convert.ToDouble(StrZone.Substring(6, 2)) + (Convert.ToInt32(StrZone.Substring(9, 2)) / 60);  
  if (TimeZoneOutP.Value.ToString().Substring(5, 1) == "-")  
    UserTimeZone = UserTimeZone;  
  else  
    UserTimeZone = -1 * UserTimeZone;  
  EI.StartDateTime = EI.StartDateTime.AddHours(UserTimeZone - EventTimeZone);  
  EI.EndDateTime = EI.EndDateTime.AddHours(UserTimeZone - EventTimeZone);  
}  

// Adds the Alerts array  
if (IsInMiCalendar == true)  
{  
  EI.AlertArray = Alert.GetAlertArray(EI.InstanceID, UserTimeZone, EventTimeZone);  
}  
else  
{  
  EI.AlertArray = Alert.GetAlertArray(EI.InstanceID, EventTimeZone, EventTimeZone);  
}  

ReturnedEvent.EventInstances.Add(EI);  
connection.Close();  
return ReturnedEvent;  
}  
catch (Exception ex)  
{  
  IsInMiCalendar = false;  
  return null;  
}  

Listing 31 GetEventInstance Method from miClasses
Appendix VII: Listing of .NET My Services

The following are the initial service offerings [10]:

- **.NET Alerts** — Provides subscription-based announcements, alerts, reports, management, and routing
- **.NET ApplicationSettings** — Provides settings for applications
- **.NET Calendar** — Provides time-management functions such as setting tasks and scheduling appointments
- **.NET Categories** — Provides the means to create personalized categories to group information
- **.NET Contacts** — Offers address and contact management, similar to ‘Contacts’ in Outlook
- **.NET Documents** — Provides universally accessible a secure hard drive for document storage
- **.NET Devices** — Used to contain device settings and capabilities
- **.NET FavoriteWebSites** — Provides a universally accessible way to store, retrieve, and manage favorite URLs and other Web identifiers (bookmarks will be available to the user no matter which computer or device is used)
- **.NET Inbox** — Similar to online mail today but offers more generic message management for e-mail, voice mail, faxes and more
- **.NET Lists** — A service for creating general purpose lists (e.g. a shopping list or a "to do" list)
- **.NET Locations** — Repository for electronic and geographical locations and rendezvous
- **.NET Presence** — Online, offline, busy, free, and which device(s) to send alerts to
- **.NET Profile** — Hosts user’s personal profile: name, nickname, special dates, picture, address, etc...
- **.NET Services** — All services provided for an identity
- **.NET Wallet** — Holds receipts, payment instruments, coupons, and other transaction records
Appendix VIII: Executive Summary on ASP Industry

Primarily adapted from [2] and other sources mentioned herein:

Defining Characteristics
- Delivery over a network
- Externally managed
- One-to-many service
- Service-fee-based pricing
- Contract fulfillment

Facts and Figures
- Peak value for number of ASPs reached 800 in 2000 [IDC]
- No vendor claims market dominance [Gartner]
- Top 10 ASPs accounted for 30% of total US market [Gartner]
- First position in revenues of $110 million by USi in 2000 [Gartner]
- US ASP market had total revenues of $1.4 billion in 2000 [Gartner]
- Top 20 ASPs have about 1.6 million users [ASPnews]
- 20% of ASP users are high technology businesses [PMP Research]
- 16% of ASP users are retail organizations [PMP Research]
- 12% of ASP users are public sector (healthcare, gov.) [PMP Research]

ASP Evaluation Metrics
- Key Financial Metrics (quantitative determinants of ASP value):
  - Number of ASP clients
  - Average monthly recurring revenue per client
  - Average length of contract
  => Total ASP contract backlog
- Key Strategic Metrics (qualitative determinants of ASP value):
  (Higher relative value vs. Lower relative value)
  - Specialist ⇄ One-stop-shop
  - Proprietary hosted solution ⇄ 3rd party hosted solution
  - Specific industry/Functional expertise ⇄ Generalist
  - Channel partnerships ⇄ "Do-it-yourself" [2]

Top-Line Benefits
- Rapid deployment
- Primary business focus
- Affordable access to high-end applications and technology
- Improved total IT performance
- IT staff recruitment and retention
- Scalability to meet business growth
**Bottom-Line Benefits**

- Avoid capital investments
- Raise return on investment (ROI)
- Lower cost of entry for applications
- Predictable costs

<table>
<thead>
<tr>
<th>Considered benefits using an ASP</th>
<th>Companies currently using an ASP and plan to adopt additional ASP offerings in the future</th>
<th>Companies currently not using an ASP but plan to adopt ASP offerings in the future</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access to high-end applications</td>
<td>66.1%</td>
<td>62.0%</td>
</tr>
<tr>
<td>Increased flexibility</td>
<td>49.2%</td>
<td>47.7%</td>
</tr>
<tr>
<td>Reduced initial capital outlay</td>
<td>51.7%</td>
<td>62.0%</td>
</tr>
<tr>
<td>Guaranteed performance/uptime</td>
<td>46.6%</td>
<td>52.6%</td>
</tr>
<tr>
<td>Alleviates shortage of IT personnel</td>
<td>55.9%</td>
<td>57.0%</td>
</tr>
<tr>
<td>Reduced length of implementation</td>
<td>57.6%</td>
<td>56.4%</td>
</tr>
<tr>
<td>Lower IT costs over time</td>
<td>51.7%</td>
<td>60.0%</td>
</tr>
<tr>
<td>Security</td>
<td>25.4%</td>
<td>21.6%</td>
</tr>
<tr>
<td>Other</td>
<td>1.7%</td>
<td>1.7%</td>
</tr>
</tbody>
</table>

**Table 6 Benefits of Using an Application Service Provider**

Source: ITAA ASP Customer Demand Survey, 2000 (The 3 most frequently mentioned benefits are varied *as such*.)

**Market Estimations**

<table>
<thead>
<tr>
<th>Source</th>
<th>1999</th>
<th>2000</th>
<th>2001</th>
<th>2003</th>
<th>2004</th>
<th>2005</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDC</td>
<td>$300 million</td>
<td>$770 million</td>
<td></td>
<td>$8 billion</td>
<td>$13-24 billion</td>
<td></td>
</tr>
<tr>
<td>Forrester</td>
<td>$700 million</td>
<td></td>
<td></td>
<td>$6.4 billion</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Datamonitor</td>
<td></td>
<td></td>
<td></td>
<td>$5 billion</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aberdeen</td>
<td></td>
<td></td>
<td>$3 billion</td>
<td></td>
<td>$16.1 billion</td>
<td></td>
</tr>
<tr>
<td>Gartner</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$25.3 billion</td>
<td></td>
</tr>
</tbody>
</table>

**Table 7 Market Estimations for the ASP Industry**
Revenue Forecast

![Revenue Forecast Graph](image)

**Figure 45** Worldwide ASP Revenue Forecast

Source: IDC, 2001