Rochester Institute of Technology

RIT Scholar Works

Theses

2007

An SOA-based tennis club reservation system

Alan Amin

Follow this and additional works at: https://scholarworks.rit.edu/theses

Recommended Citation

Amin, Alan, "An SOA-based tennis club reservation system" (2007). Thesis. Rochester Institute of Technology. Accessed from

This Master's Project is brought to you for free and open access by RIT Scholar Works. It has been accepted for inclusion in Theses by an authorized administrator of RIT Scholar Works. For more information, please contact ritscholarworks@rit.edu.

An SOA-based Tennis Club Reservation System

By

Alan K. Amin

Project submitted in partial fulfillment of the requirements for the degree of Master of Science in Information Technology

Rochester Institute of Technology

B. Thomas Golisano College
of
Computing and Information Sciences

May 10, 2007

Rochester Institute of Technology

B. Thomas Golisano College
of
Computing and Information Sciences

Master of Science in Information Technology

Project Approval Form

Student Name: Alan K. Amin

Project Title: An SOA-based Tennis Club Reservation System

Project Committee

Name	Signature	Date

Stephen J. Zilora .
Chair

Ed Holden .
Committee Member

_____ .
Committee Member

© Copyright 2007 Alan K. Amin

All Rights Reserved

Abstract

This paper illustrates the design and implementation of an SOA-based tennis club reservation system using Web services. It reconciles best views from both the practical and theoretical worlds to yield a reasonable solution that conforms to essential design patterns and exhibits an acceptable performance. The server-side system consists of three abstract layers for data store, data access, and Web services. On the client-side, three applications are developed to consume the Web services: a Web application, a desktop application, and a PDA application.

Contents

Introduction

Web services have acquired a fairly wide acceptance in the IT world for developing internet applications in general and business-to-business applications in particular. Major software vendors like Microsoft, IBM and Sun have espoused Web services; they have been trying to set a solid foundation on which this technology would evolve. Although achievements are not made as fast as if the technology were a proprietary one, Web services are steadily approaching maturity.

In common usage, Web services are service-oriented systems in which clients and servers communicate through HTTP using SOAP (Simple Object Access Protocol) envelopes which are XML (eXtensible Markup Language) messages encoded in a special format. Using open standards and plain-text messages give Web services their main strength in interoperability.

The goal of this study is to implement a reservation system for tennis clubs using Web services. The system inherits the characteristics of Web services in client-independence, platform-independence, and extensibility. In addition, three client applications are developed to consume the server-side services. Additional new clients of various types and platforms can be developed with minimal effort.

Service-Oriented Architecture (SOA)

SOA is an architectural scheme in which software components are loosely coupled, which in turn reduces artificial dependency among interacting systems. “Artificial dependency is the set of factors that a system has to comply with in order to consume the features or services provided by other systems. Typical artificial dependency factors are language dependency, platform dependency, API dependency, etc. Artificial dependency always exists, but it or its cost can be reduced”[1].

SOA systems achieve loose coupling by employing:

1- ubiquitous interfaces among software components
2- descriptive messages constrained by an extensible schema to be delivered through the interfaces

The ubiquitous interfaces among components guarantee a universal method of communication which ensures that all components can interact. In addition, the extensible schema ensures that newer versions of the services will not break the existing ones.

Web Services as a Service-Oriented Architecture

For an SOA system to be considered as a Web services implementation, the system must comply with the following constraints:

1- The interfaces among components must be Internet protocols like HTTP or FTP.
2- The descriptive messages must be in XML

There have been many implementations of Web services, but the de facto and most important implementation is SOAP Web services which have the following constraints:

1- The exchanged messages are encoded in SOAP
2- Services are described using WSDL (Web Services Description Language)

For the rest of the paper, the term "Web services" is used to refer to "SOAP Web services".

Putting it all together, "The term Web services describes a standardized way of integrating Web-based applications using the XML, SOAP, WSDL and UDDI open standards over an Internet protocol backbone. XML is used to tag the data, SOAP is used to transfer the data, WSDL is used for describing the services available and UDDI is used for listing what services are available. Used primarily as a means for businesses to

communicate with each other and with clients, Web services allow organizations to communicate data without intimate knowledge of each other's IT systems behind the firewall"[2]

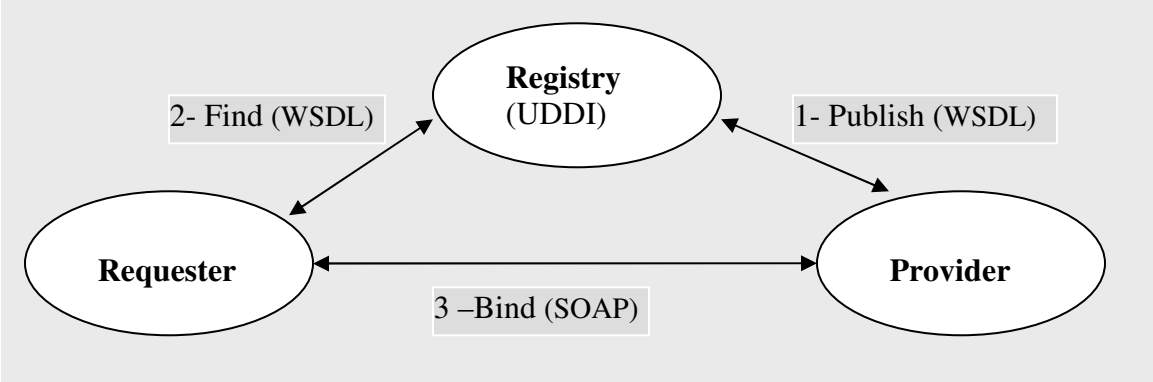

Fig. 1: Web services

In Fig.1, Provider and Requester are roles played by software components. The service provider software publishes its services using WSDL which includes detailed information about the location (URI) of the provider and the contracts (interface) of the services to be consumed by clients. Ideally, service requester components look up a registry of Web services at runtime or design time. The registry is defined using UDDI (Universal Description Discovery and Integration) which is a protocol to describe and publish Web services. After the requester has found an appropriate service, it uses the metadata in the UDDI to find the URI and the interface of the service. Communications between the requesters and providers are typically done in SOAP messages.

CourtTime Reservation System

Many companies offer software products that are specially designed for tennis court reservations, including TennisSource.Net, Chelsea Information Systems, Inc., and CodeASite, LLC which offers OpenCourtTime. However, none of the commercial

products available now is utilizing web services, and the only way to access them is through desktop web browsers.

CourtTime is an SOA-based tennis club reservation system implemented in Web services. The system provides seamless services to both personal computers and hand-held devices like PDAs and smartphones. In fact, implemented in Web services, the system can serve any client device that has an Internet connection regardless of the type or platform of the client device.

In addition to the suite of services on the server-side, the current version of the system also includes three client applications that utilize the services offered. The Web-application client can be used by all types of users regardless of their privileges, while the desktop application client is to be used only by admins and staff members of the clubs. The PDA client is used for basic on-the-go requests for users who travel frequently.

The functional requirements of the system are mainly derived from an existing version that was implemented as a plain Web application accessed only by desktop browsers. New requirements are added to the system as the result of two meetings with the representative of the club that is using the existing application.

CourtTime Design

Reservation systems are time-critical software products that are required to have relatively small response-times so as to be acceptable by the users; however, loosely-coupled systems use multiple layers of abstraction to decouple objects from one another, imposing a negative impact on performance. Hence, for an SOA-based reservation system to be acceptable by users and maintainable by developers, the right balance between design and performance is crucial. Considering the overall performance requirements of a reservation system and avoiding an unclean chaotic design, the server-side system is composed of three abstract layers (Fig. 2):

1- Data Store Layer

This layer stores persistent data that should be retained by the system for future use or reference. The persistent data can be club information that is essential for the proper functioning of the system, or it can be user information that needs to be stored between sessions.

2- Data Access Layer

The data access layer works as a translator between the data store layer and Web services layer (next item). When the Web services need to save data in the data store, they delegate the job to this layer which makes the Web services independent from future changes to the data store.

3- Web Services Layer

This layer abstracts the actual services offered by the system and the business logic associated with them. Business logic enforces business rules of the application domain while Web services are the interface between the server application and the clients, thus, clients have a universal access point to the provided services.

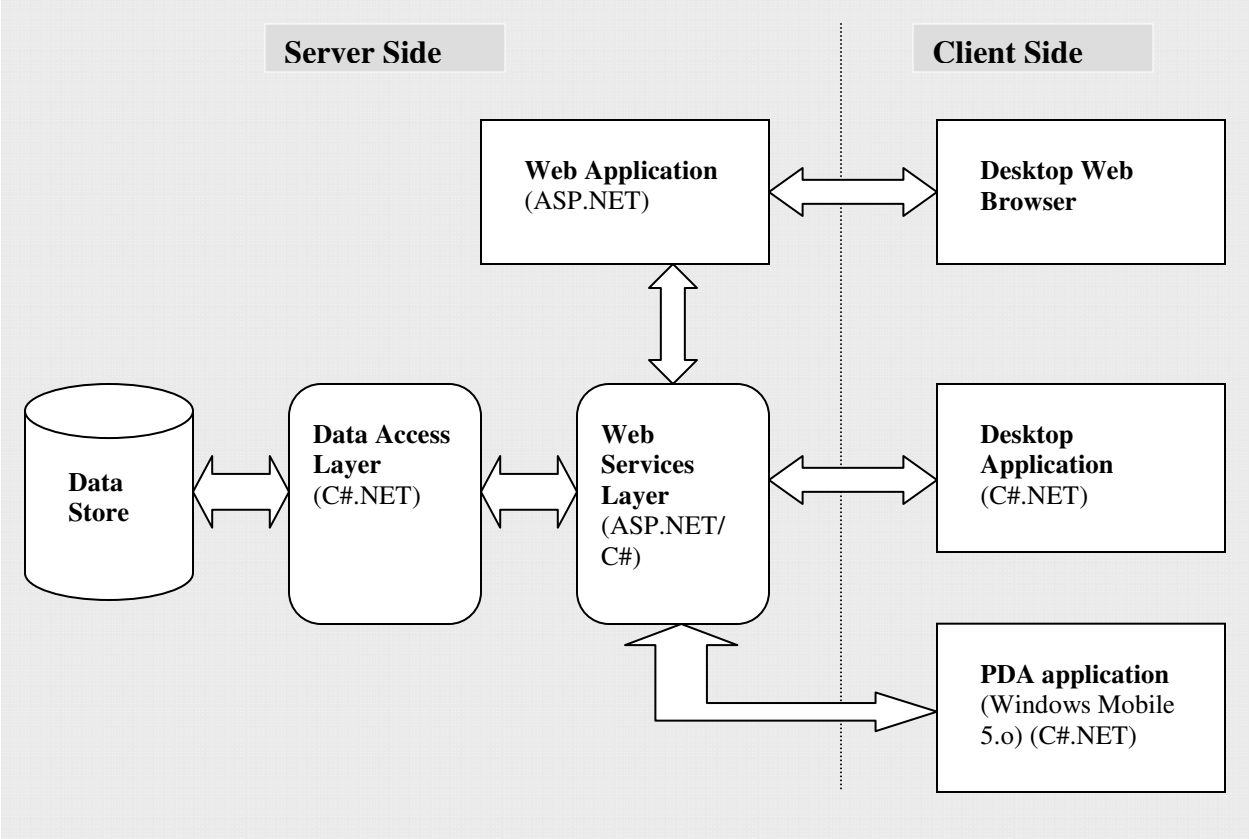

Fig. 2: CourtTime Design

CourtTime Implementation

All of the design layers mentioned in the previous section, except the data store layer, are implemented in Microsoft's .NET framework 2.0 using C# and Visual Studio 2005 (Professional Edition).

Server-side Implementation

1- Implementation of Data Store Layer

The data store is implemented using relational databases in MySQL 5. The databases were already designed for an older version of the system that was implemented as a Web application; only minor changes are introduced, like adding a new table or field, or

changing the data type of a field.

CourtTime is designed to serve a group of tennis clubs simultaneously. The data store implementation can be divided into two categories of databases: a front-end database, and a group of back-end databases (Fig. 3). The front-end database is the first database to which clients connect; it holds information about the clubs that can be served by the system, including the club databases' connection strings. Having this information, clients can connect to any of the back-end databases that store detailed information about the clubs. In other words, every club should have a separate back-end database whose connection string is stored in the front-end database. Obviously, back-end databases should all conform to the same database design that is expected by the data access layer.

The current implementation of the system has two databases: "ClubsDB" as the front-end database that holds information about club entities, and "CourtTime" as a back-end database that holds detailed information about a club. Fig. 4 and fig. 5 illustrate designs of "ClubsDB" and "CourtTime" databases, respectively. Please refer to appendix A for the descriptions of the table columns.

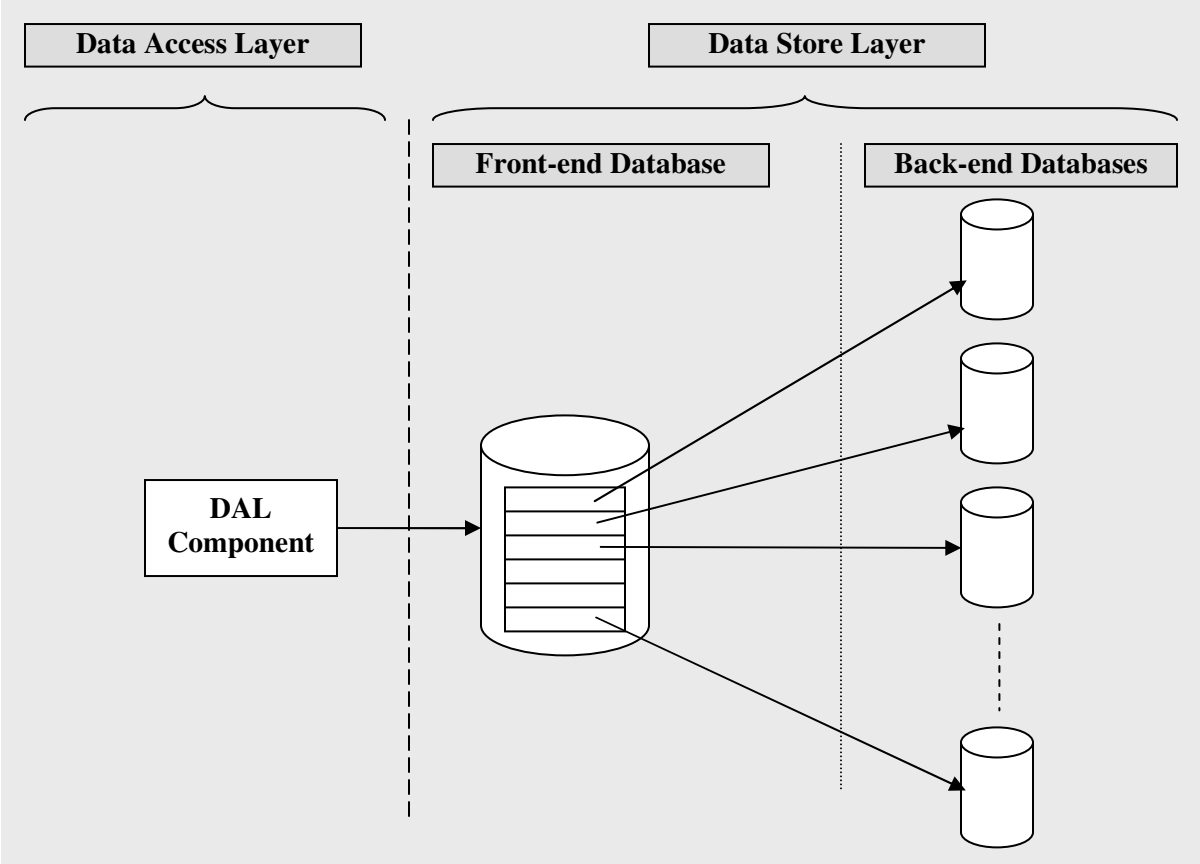

Fig. 3: Data Store Implementation

clubs		
ClubCode	Varchar(20)	NN (PK)
ClubName	Varchar(75)	
Website	Varchar(75)	
Connection	Varchar(100)	
StartTime	Datetime	
EndTime	Datetime	
Msg	Varchar(255)	
Voice	Varchar(20)	
AdvNotice	Integer	NN
OptPro	Integer	NN
OptAid	Integer	NN

Fig. 4: Design of Database “ClubsDB”

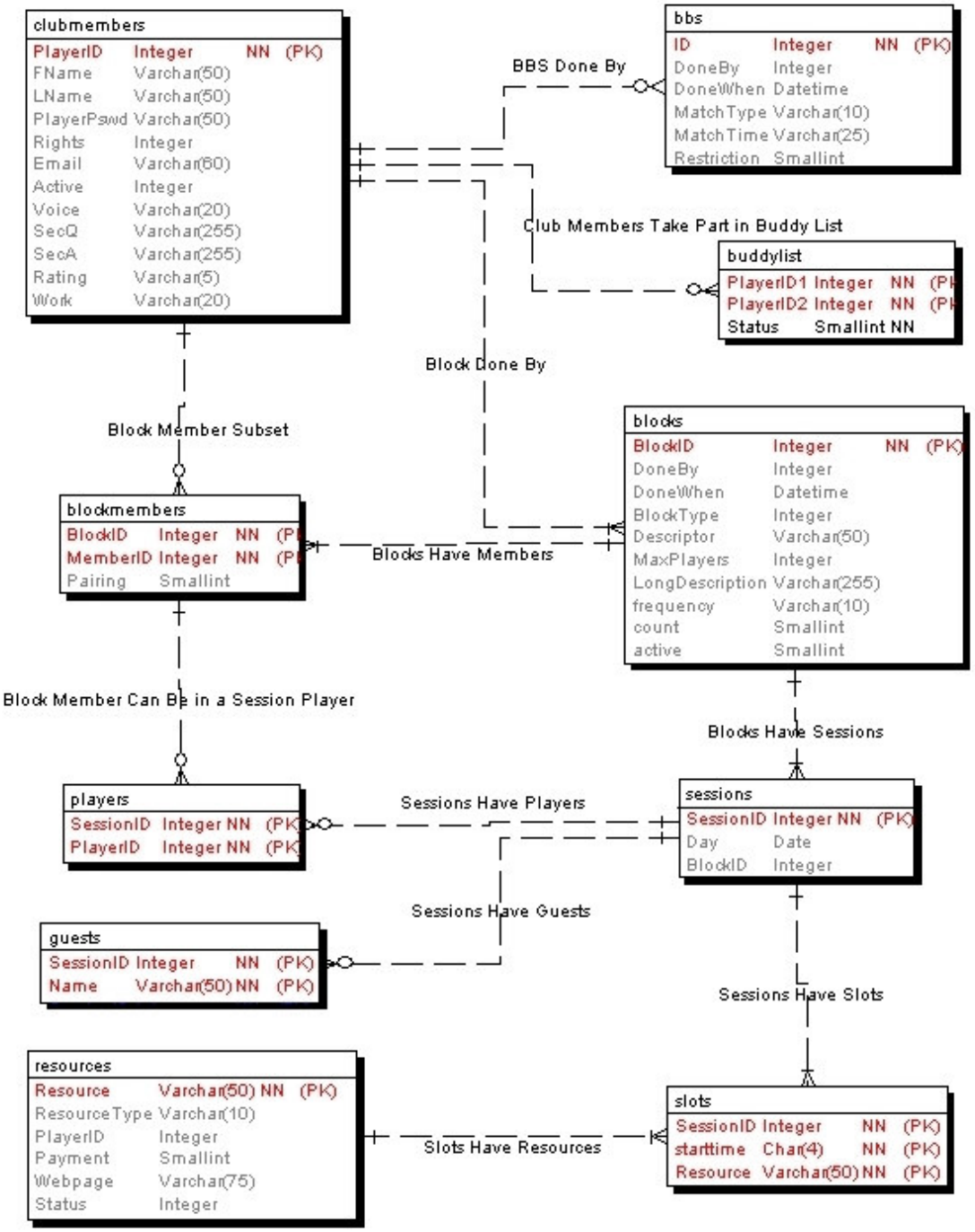

Fig. 5: Design of Database “CourtTime”

2- Implementation of Data Access Layer

Data access layer encapsulates implementation details of database queries and connections. The system implements this layer using a set of classes ending with “DAL” suffix located in the “DAL” folder. Every table in the database has a corresponding data access class with the same name as the table suffixed by “DAL”. For example, table “blocks” has a corresponding data access class named “BlocksDAL”. A class with a name corresponding to a database table does not know about any other table in the database other than its matching table, so it accesses only that table in the database. BlocksDAL class does not read or write data to any table other than “blocks”, even if the data is partially driven from “blocks”. If an operation needs access to more than a table, like in JOIN operations, a separate class named “JoinDAL” is used. This class encapsulates all the operations that populate data from more than a single table. An alternative strategy would have been to fetch parts of the required data by the other single-table DAL classes and then to join them in the business layer. This method would have burdened the database with extra operations. Each single class would have competed to get a connection and then fetch the data it needed. After each class had brought in a piece of the required data, a business logic component would have handled purifying the data by extracting only the necessary information and then maybe joining them to get the final result. Instead, now in the actual implementation, if a business function needs data that is derived from multiple tables, it uses ”JoinDAL” class that in turn uses SQL JOIN operations which have a better performance than multiple individual queries joined programmatically.

The data access layer classes use MySQL Connector/Net 5.0 driver to access the database, and the assembly for this driver is located in the “Bin” folder of the server-side project.

3- Implementation of Web Services Layer

The Web services are implemented using a single class named “Service”. All the methods

in this class are exposed as Web services methods. They accept requests from client applications, enforce business rules, delegate the requests to the related components in the data access layer, and then return the result to the client. The fulfillment of a client request can be as simple as just calling a method on an access layer component. However, some requests need to undergo a series of business logic rules and use multiple access layer components to get the result, and may require transactions.

Transactions are initiated and controlled from the Web services layer using class "TxnManager" which is located in folder "DAL". Class "TxnManager" encapsulates operations related to starting a new transaction and then rolling back or committing the changes to the database. When a Web service method needs transactional functionality to get consistent results, using "TxnManager", it first initiates a transaction, then calls the methods required to get the result followed by terminating the transaction either by a rollback or a commit operation.

Since not all clients of the Web services are powerful machines with fast internet connections, some of the Web services have two versions, one for regular clients and one for limited clients like PDAs.

4- Other Implementation Details

Utility Methods

Class **"**Utility**"** implements a group of auxiliary static methods that are used by various other classes on the server-side. The reason to create a separate class for these methods is that logically they do not fit in any of the core classes on the server-side; furthermore, they are used more than once in different parts of the server-side classes.

Data Transfer Objects

.NET's DataSet objects are serializable entities that can be used to transfer data among remote components. Client applications in .NET can consume Web services that return DataSets just like any other simple type; after all it is a native .NET type. However, other

Web service toolkits, like Apache's Axis, experience problems when they generate mappings of dynamic types like DataSet because the actual layout of a DataSet cannot be determined until runtime [3]. This shortcoming imposes difficulties on developers who need to write consumer applications for a .NET DataSet Web service on other platforms. Hence, none of the CourtTime Web services return DataSet objects; instead, data transfer objects are used.

Data transfer objects are a design pattern in which serializable objects are used to transfer data across layers of a multi-tier application. CourtTime uses simple data transfer objects to transfer business data from data access layer to client applications and vice versa. The objects consist of basic data types which makes it possible for toolkits to develop accurate mappings from a WSDL document. In addition, simple data transfer objects carry an advantage of hiding the actual implementation of the data store. These objects are not necessarily a one-to-one mapping of database tables; they may contain attributes derived from different tables. Obviously, data transfer objects that map to selective attributes from different tables would also boost the performance of the Web services since relatively smaller amounts of data are transferred through the wire.

Data transfer objects of CourtTime are located in folder "DTO". To make it easy to recognize these transfer objects when they are used in other classes, the suffix "DTO" is appended to their names; e.g. ReservationDTO.

Notification Tokens

In order for the client applications to be notified of the successfulness of the non-query actions they initiated by calling a Web service, CourtTime's services return notification tokens which are small string literals that encode information about an operation's result. Notification tokens consist of the name of the operation followed by 1 or 0 to indicate the success or failure of the operation. Some notification tokens also include additional information pertinent to the reason of the failure of an operation which in turn helps client applications know the exact reason for the unsuccessful execution of a service. As an example, the message "`clubMemberUpdate0Duplicate`" indicates that information

update of a club member was not successful because it conflicted with another member's information.

A full list of the notification tokens along with their sample corresponding messages is found in appendix B. Users of the desktop client application or admins of the Web client application can easily change the messages without having to re-compile the applications. The tokens and their associated messages are in "web.config" of the Web client application, and "app.config" of the desktop client.

Client-side Implementation

1- ASP.NET Web Application

From the users' perspective, a web application is the most convenient way of accessing online services as almost all computers now have Web browsers installed. Although CourtTime's Web client is in fact a server-side application, it is grouped here with other client applications for convenience. Actually, the Web services consider it a client application; while Web browsers consider it a server-side application. Fig.6 shows a snapshot of the Web client application in Internet Explorer 7.

Important Implementation Details

A- Master Page

The Web application is implemented in ASP.NET 2.0. It utilizes the new "master page" feature in ASP.NET 2.0 to offer consistent layout and behavior for the pages in the application. CourtTime Web application contains a master page named "InsidePage.master" which contains the user interface and behavior that is common to almost all pages in the application. For instance, it checks the session of a user before a requested page is processed and delivered. For more information about master pages please refer to http://msdn2.microsoft.com/en-us/library/wtxbf3hh(vs.80).aspx.

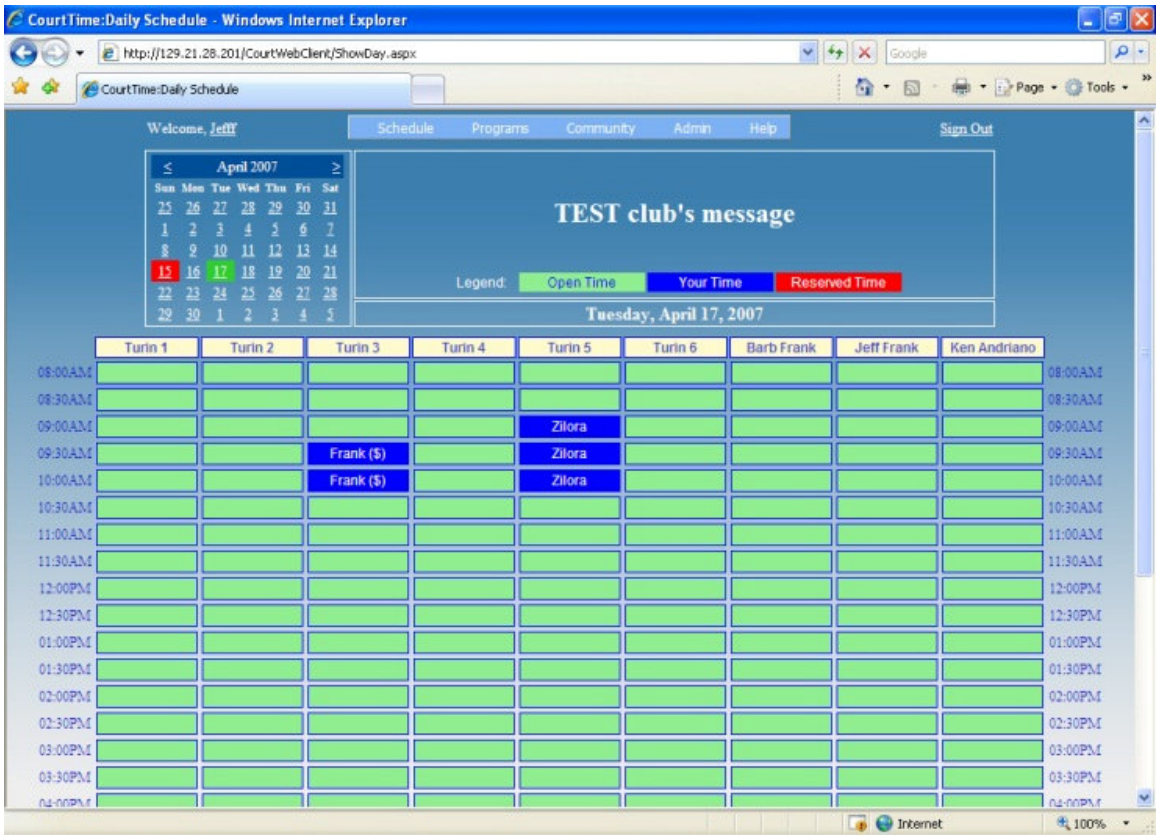

Fig. 6: Web Client Snapshot

B- Page Categories

There are three categories of pages that correspond to three categories (or groups of categories) of users in the system: admin pages to be accessed by members with at least admin privilege, staff pages to be accessed by members with at least staff privilege, and general pages to be accessed by all members. After the master page has checked user's session, control transfers to the requested page which in turn checks user's privilege to decide whether to continue processing the page or not. If the user had insufficient privilege for browsing a page, the application would redirect him to the login page.

C- SMTP Configurations

Throughout its communication with the Web services, if the Web client received an exception from a Web service method, it generates an error report including the details of the exception and sends it to an email address specified in the “Web.config” configuration file. “Web.config” contains three key-value pairs that store information about the SMTP server and the email of the error report recipient. The keys are MailTo, SMTPHost, and SMTPPort. If the key SMTPHost is deleted from the configuration file, the application will use the value “localhost”. Similarly, if SMTPPort is deleted or an invalid number is entered, port 25 is used.

Master Page Cast Problem

ASP.NET applications that utilize the master page feature introduced in ASP.NET 2.0 may crash suddenly throwing an InvalidCastException, “Unable to cast object of type 'ASP.InsidePage_master' to type 'ASP.InsidePage_master'”. The error does not occur at once when an application is deployed; the application may work properly for an undetermined period of time, and then crash while the code is not altered at all.

Microsoft has a fix for the problem, but it is not published yet. Developers can get the fix by directly contacting Microsoft. In fact, the fix is not published yet because it does not solve all occurrences of the problem. It has been a long time now since ASP.NET 2.0 was released, and the fix is still not available. Fortunately, there is a workaround to solve the issue. Disabling batch compilation in the Web.config file stops the problem. This workaround is not appropriate for development environments; it slows down the compilation of the Web application. However, for a production environment, it has no side effects.

2- Desktop Application

The desktop client is a C# application to be used by club admins and staff; other users cannot log into the system with this application. Because of the rich GUI widgets available in desktop applications, the user can perform the same operations offered by the Web client much easier and quicker. In addition, unlike a Web browser which communicates with the ASP.NET Web application to consume the Web services, the desktop application communicates directly with the Web services layer, skipping a level of indirection and enhancing the system's overall performance.

As shown in fig. 7, the application's user interface consists of two main regions: the upper half and the lower half regions. The upper region usually shows the current data in the system; while the lower one is the user's working area where (s)he can update information and commit changes to the system. Actually, the upper region consists of a single TabControl that contains a list of TabPage controls. Admin users can see all the tabs; however, staff users can only see a subset of the tabs due to the restrictions on the operations they are allowed to perform. On the lower region, the application uses layers of Panel controls. Almost every tab has a corresponding panel on the lower region that shows updateable detailed information about some data in that tab. When a tab page is selected, all panels in the lower region are hidden; only that tab's corresponding panel appears to the user.

Being solely used by admin and staff members, the application does not expire user sessions; users stay signed in until they explicitly sign out. Some of the tabs in the application have refresh buttons that help users get the latest information from the system in case they kept the application idle for a long time.

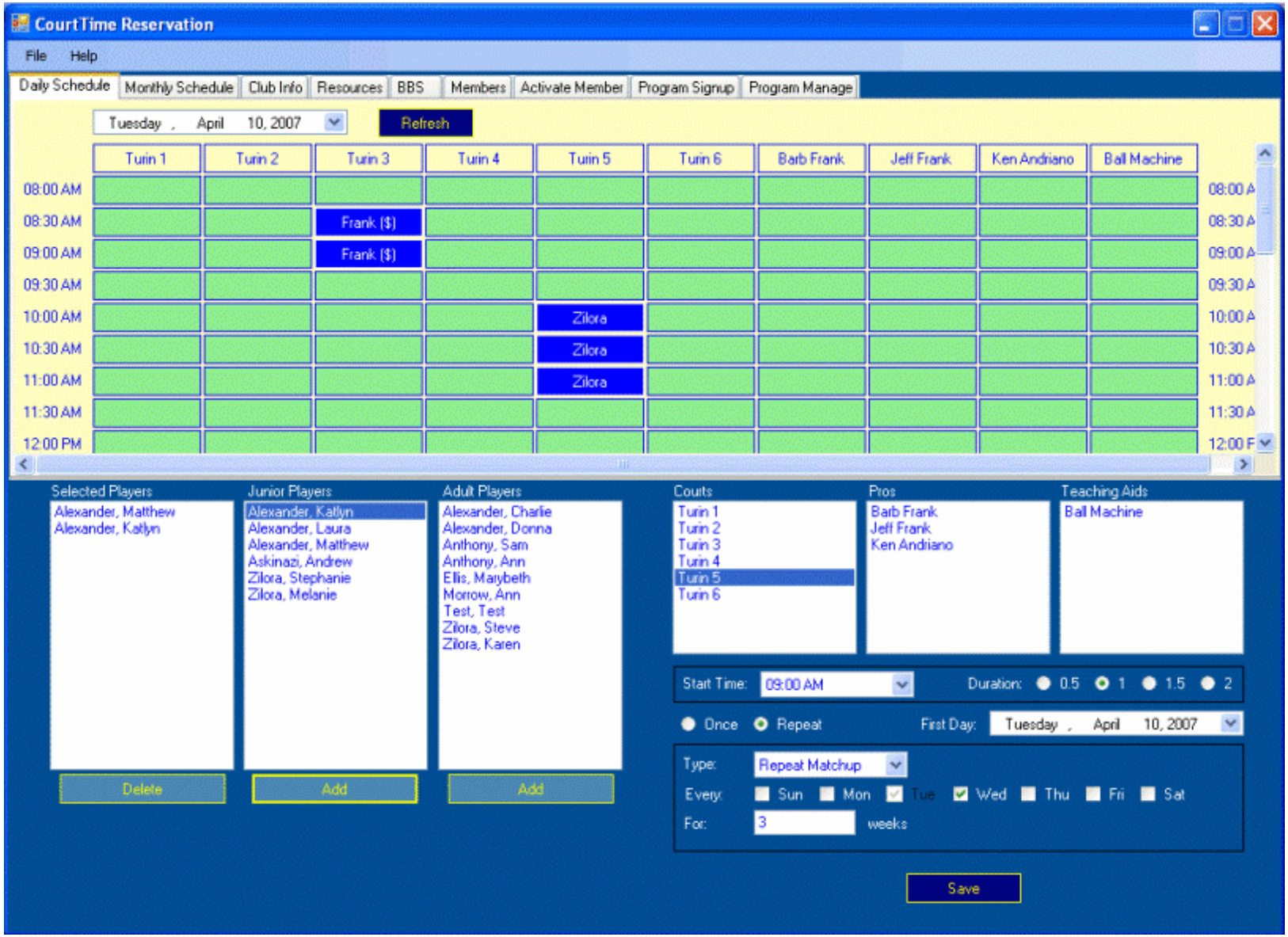

Fig. 7: Desktop Client Snapshot

3- Windows Mobile 5.0 Application

The Windows Mobile (WM) 5.0 application is a PDA client application to be used by club members who travel frequently to help them do some basic operations. It enables users to see their time in two-week frames, see their buddies, and make reservations.

The application consumes a hybrid of regular and PDA-specific Web services accessed by the tabbed GUI (fig.8).

Fig. 8: WM Client Snapshot

References

1- Haas, H., & Brown, A. (2004, February 3). *Web services glossary*. Retrieved April 16, 2007, from http://dev.w3.org/cvsweb/~checkout~/2002/ws/arch/glossary/wsa-glossary.html#loosecoupling

2- *Web services*. (2006, June 7). Retrieved April 29, 2007, from http://inews.webopedia.com/TERM/W/Web_services.html

3- Skonnard, A. (2003, April). The XML files : Web services and DataSets. *MSDN Magazine, 18*(4). Retrieved April 16, 2007, from http://msdn.microsoft.com/msdnmag/issues/03/04/XMLFiles/

Appendix A: Description of Database Table Fields

1- ClubsDB Database

Clubs
Contains all member clubs. If a club has different memberships for summer and winter season, they will be listed twice (with a unique name each time).

Fields	Type	Description
ClubCode	A20*	Key field
ClubName	A75	Name of club
Website	A75	Club's website if there is one
Connection	A100	Connection string to club's database
StartTime	Date/Time	Club's daily opening time
EndTime	Date/Time	Club's daily closing time
Msg	A255	Welcome message
Voice	A20	Club's phone number
AdvNotice	Int	Number of hours a club imposes on its members to reserve a pro beforehand
OptPro	Int	Whether basic users can reserve pros or not.
OptAid	Int	Whether basic users can reserve teaching aids or not.

2- CourtTime Database

ClubMembers
Contains all members of the club.

Fields	Type	Description
PlayerID	Int*	Key field
FName	A50	First Name
LName	A50	Last Name
PlayerPswd	A50	Password
Rights	Int	Rights of member: 40-General Adult, 41-General Junior, 30-Pro, 20-Staff, 10-Admin
Email	A60	Email address
Active	Int	Member's status 3 : Suspended , 2 : Active , 1 : Pending , 0 : Inactive
Voice	A20	Home phone number
SecQ	A255	Security question to verify identity in case user forgot password.
SecA	A255	Answer to security question.

Rating	A5	NTRP rating
Work	A20	Work phone number

Blocks

This is the main reservation element. Every reservation starts with a block and may contain 1 or more sessions.

Fields	Type	Description
BlockID	Int*	Key field
DoneBy	Int	Club member who made the reservation. Links to ClubMember table's PlayerID.
DoneWhen	Date/Time	Date and time when the reservation was made. For programs, it's the date of the first session of that program. **NOTE:** Changing the date of the first session of a program (using reservation update screen) will NOT change this field.
BlockType	Int	Type of reservation: 0 : One Time 10 : Repeat Matchup 20 : Adult Group Lessons (prog.) 21 : Junior Group Lessons (prog.) 22 : Adult League (prog.) 23 : Individual Doubles Rotation (prog.) 24 : Team Doubles Rotation (prog.)
Descriptor	A50	Short descriptor to appear on schedule. For non-programs, it's the last name of the person who made the reservation. For programs, it's the name of that program.
MaxPlayers	Int	Max number of players allowed in this reservation. Used for programs.
LongDescription	A255	Full description of this reservation. Used for programs.
Frequency	A10	This field is not used; it's retained for legacy.
Count	Int	Number of sessions in this block.
Active	Int	Whether users can signup to this reservation or not. Used for programs.

BlockMembers

This table stores players assigned to a reservation block. For programs, it holds those members who are signed up for the program; however, adding a player to a session of a program does not affect this table, the player would be added to "Players" table.

Fields	Type	Description
BlockID	Int*	Key field. Links to Blocks table's BlockID
MemberID	Int*	Key field. Links to ClubMembers' PlayerID
Pairing	Int	Used for team competition to indicate which block members are teamed up. It's not used in the current implementation; always set to 0.

Sessions

This is the individual daily piece of a reservation.

Fields	Type	Description
SessionID	Int*	Key field
Day	Date	Date of reservation
BlockID	Int	Links to Blocks' BlockID

Slots

This is the half-hourly piece(s) of a reservation. It describes what is being reserved and when.

Fields	Type	Description
SessionID	Int*	Key field. Links to Sessions' SessionID
StartTime	A(4)*	Key field. Start time of reservation
Resource	A(50)*	Key field.The reserved resource

Players

Who is playing in a specific session.

Fields	Type	Description
SessionID	Int*	Key field. Links to Sessions' SessionID
PlayerID	Int*	Key field. Indicates a player in a session. **NOTE:** For programs, this field is NOT necessarily relating to the MemberID field in BlockMembers. The new implementation of the system allows admins and staff to add players to a session of a program even if the player is not signed up for that program.

Guests
The guest who is playing in a specific session.

Fields	Type	Description
SessionID	Int*	Key field. Links to Sessions' SessionID
Name	A50*	Key field. Name of the guest.

Resources
Lookup table of available resources to reserve.

Fields	Type	Description
Resource	A50*	Name of resource.
ResourceType	A10	Type of resource. This is used by the application code and is limited to: Court, Taid, and Pro
PlayerID	Int	For pros, it links to ClubMembers' PlayerID. For non-pro resources it's set to 0.
Payment	Int	Whether the resource can be used free of charge or not.
Webpage	A75	Webpage of the resource, if available.
Status	Int	Whether the resource is active or not.

BBS
Bulletin board that stores users' published requests to find an opponent.

Fields	Type	Description
ID	Int*	Key field
DoneBy	Int	Links to the member (in clubmembers) who posted the request.
DoneWhen	Date/Time	Date and time when the request was posted.
MatchType	A10	The desired match type: singles or doubles
MatchTime	A25	The desired date and time for the match.
Restriction	Int	Whether the BBS post is intended for buddies or it's public.

BuddyList
Buddies (who are club members) of club members.

Fields	Type	Description
PlayerID1	Int*	Club member who initiated the buddy request
PlayerID2	Int*	Club member who the request is sent to.
Status	Int	Whether the buddy request is pending (0) or approved (1).

Appendix B: Notification Tokens (**token** : sample message)

reservUpdate1 : Reservation has been updated
reservUpdate0 : Reservation could NOT be updated.Sorry for the inconvenience. Please try again later.
reservUpdate0Conflict : Your request conflicts with another reservation.

reservSave1 : Reservation has been saved. In case you want to cancel your reservation, please delete it at least one hour beforehand
reservSave0 : Reservation could NOT be saved. Sorry for the inconvenience. Please try again later.
reservSave0Conflict : Your request conflicts with another reservation.

repeatingReservSave1 : Repeating reservation has been saved.
repeatingReservSave0 : Repeating reservation could NOT be saved. Sorry for the inconvenience. Please try again later.
repeatingReservSave0Conflict : Your request conflicts with another reservation.

programSave1 : Program has been saved.
programSave0 : Program could NOT be saved. Sorry for the inconvenience. Please try again later.
programSave0Conflict : Program conflicts with another reservation.

reservDelete1 : Reservation has been deleted
reservDelete0 : Reservation could NOT be deleted. Sorry for the inconvenience. Please try again later.

programDelete1 : Program has been deleted.
programDelete0 : Program could NOT be deleted. Sorry for the inconvenience. Please try again later.

clubInfoUpdate1 : Club information has been updated
clubInfoUpdate0 : Club information could NOT be updated. Sorry for the inconvenience. Please try again later.

clubMemberInsert1 : Member has been created
clubMemberInsert0 : Member could NOT be created. Sorry for the inconvenience. Please try again later.
clubMemberInsert0Duplicate : Member could NOT be created. Another member with the same first and last names already exists.

clubMemberUpdate1 : Member has been updated
clubMemberUpdate0 : Member could NOT be updated. Sorry for the inconvenience. Please try again later.
clubMemberUpdate0Duplicate : Member could NOT be updated. Another member with the same first and last names already exists.

resourceInfoUpdate1 : Resource information has been updated
resourceInfoUpdate0 : Resource information could NOT be updated. Sorry for the inconvenience. Please try again later.

resourceInfoSave1 : Resource information has been saved
resourceInfoSave0 : Resource information could NOT be saved. Sorry for the inconvenience. Please try again later.

resourceInfoSave0Conflict : Resource information could NOT be saved. Another resource with the same name already exists.

bbsDelete1 : BBS information has been deleted
bbsDelete0 : BBS information could NOT be deleted. Sorry for the inconvenience. Please try again later.

bbsSave1 : BBS information has been saved
bbsSave0 : BBS information could NOT be saved. Sorry for the inconvenience. Please try again later.

blockUpdate1 : Program info has been updated
blockUpdate0 : Program info could NOT be updated. Sorry for the inconvenience. Please try again later.

blockMemberInsert1 : Sign up successful
blockMemberInsert0 : Member could NOT be signed up. Sorry for the inconvenience. Please try again later.

blockMemberDelete1 : Drop out successful
blockMemberDelete0 : Member could NOT be dropped out. Sorry for the inconvenience. Please try again later.

blockMemberUpdate1 : Update successful
blockMemberUpdate0 : Update failed. Sorry for the inconvenience. Please try again later.

insBuddy1 : Request done successfully
insBuddy0 : Request could NOT be done. Sorry for the inconvenience. Please try again later.
insBuddy0AlreadyBuddies : Member already in your buddy list
insBuddy0AlreadyPending : Approval already pending
insBuddy0Self : Request could NOT be done. User cannot add himself to his own buddy list
insBuddy0NotFound : Member Not Found

acceptBuddy1 : Buddy accepted successfully
acceptBuddy0 : Buddy could NOT be accepted. Sorry for the inconvenience. Please try again later.

deleteBuddy1 : Done successfully
deleteBuddy0 : Operation could NOT be done. Sorry for the inconvenience. Please try again later.

memberActivate1 : Member has been activated
memberActivate0 : Member could NOT be activated. Sorry for the inconvenience. Please try again later.