Unifying the Code-base for a Client-Server E-Government Application

Vinitha Muraleedharan[1], Andrew Strelzoff[2], Tim Rehner[3], Ray Seyfarth[4]

Abstract – The Family Network Partnership at University of Southern Mississippi started Sword as a pilot project in the year 2000 with help from local government. The aim was to track juveniles through the justice system. Over the years, the system has grown organically and currently serves several agencies including Youth courts and Detention centers in southern Mississippi. The original framework has undergone several transformations to meet the needs of various counties. The major challenges with continued growth of the system are maintenance of multiple versions and the need for inter-county data access. In order to solve these problems and efficiently expand into other jurisdictions, the code from differing versions must be unified into a single application serving diverse needs. This paper describes the process and challenge of unifying the code-base of Sword to produce a single application that can best serve the needs of the juvenile justice system in southern Mississippi.

Keywords: Unifying; Justice; C++; Database; VB.NET.

INTRODUCTION

Sword was developed to aid Forrest county Detention centers and Youth courts in keeping track of juveniles trickling through their system. It helped them maintain information about juveniles and their charges, in addition to producing automated reports and court documents. As Sword emerged as a functional and efficient system, other agencies showed interest in using the application. Before the system could be deployed in the other agencies, it was customized to meet their needs by adding new features and culling out unnecessary ones. As this trend of customized clients continued, the applications differed substantially from one agency to another. This caused a lot of problem in maintenance and development such as: (1) bugs common to all agencies' applications had to be fixed in each client separately, (2) new features that multiple counties were interested in had to be added repeatedly in different clients, (3) users had to access and log into different versions of the client installed on the same machine to access inter county data.

The solution to these problems was to unify the different versions and create one single application that would allow cross county access and reduce maintenance issues. The merged application would facilitate faster and more efficient growth into more counties. Unification of different versions of applications is a significant task undertaken by many in the past.

---

[1] Graduate Student, School of Computing, 118 College Dr., Hattiesburg, MS 39406-8312, Vinitha.Muraleedharan@usm.edu

[2] Assistant Professor, School of Computing, 118 College Dr., Hattiesburg, MS 39406, strelz@orca.st.usm.edu

[3] Associate Professor, School of Computing, 118 College Dr., Hattiesburg, MS 39406, Tim.Rehner@usm.edu

[4] Associate Professor, School of Computing, 118 College Dr., Hattiesburg, MS 39406, Ray.Seyfarth@usm.edu

Traditional approach to unification is multi-staged. At each stage, a different component of the system is merged, and steps taken at one stage overflows into another.

For example, when CSC (Computer Sciences Corporation) faced the task of merging applications used by three New York City hospitals into one health care management system, they undertook a multistage approach. The different stages being- (1) consolidating the data centers, so all data was at one server, with a single help-desk, (2) upgrading the systems used in all the hospitals to conform to the same standards, and (3) merging the applications of each system. [2]

Later sections describe the multi-stage approach adopted for unification of Sword.

COMPONENTS OF SWORD

Before getting into the details of the unification process, this section presents the various components of Sword and their interconnection. This will make it easier to understand the unification process and the reason behind it. Sword can be visualized as having 3 components: (1) the front end or the client, which is the interface between the system and the user, (2) the server, which interacts with the client, accepts requests from the user, and returns relevant data, and (3) the actual database, which stores all the information about users, counties, and most importantly the youths being tracked. The following figure illustrates this division of components and communication between them, which will be explained in later sections.

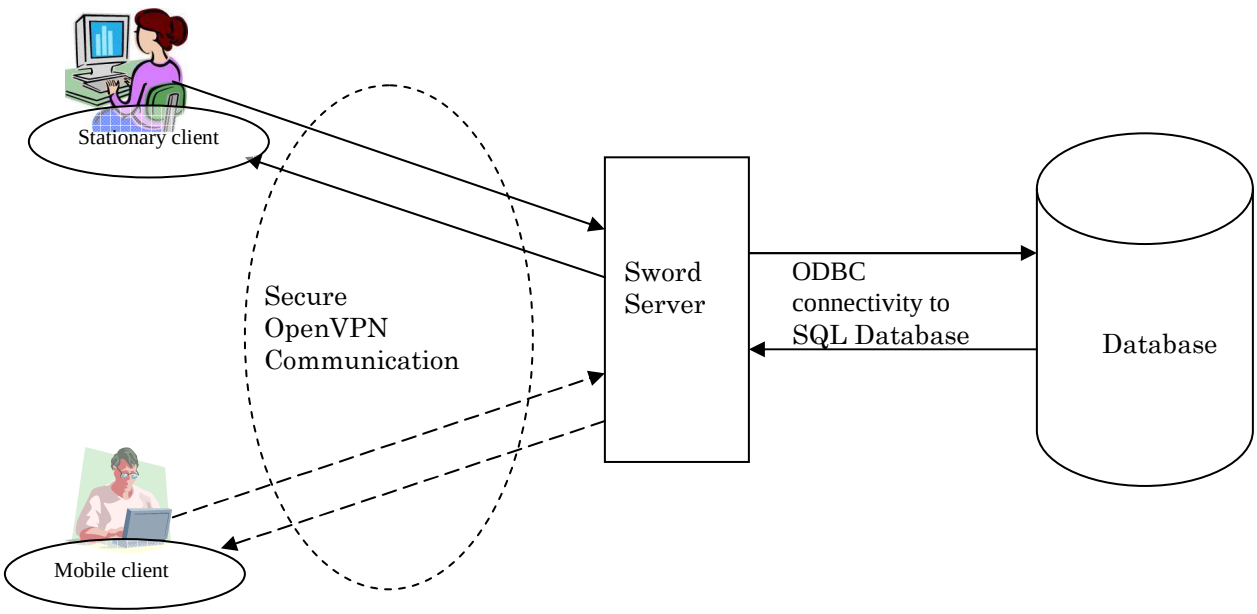

Fig 1: Components of the SWORD system

Front end/ Client

The client side of Sword is a user friendly application designed in VB.NET (Visual Basic .NET). The client does very little of the actual data processing. It accepts data from the user converts it into a format suitable for transmission and sends the data to the server. For some counties, the server is remote with respect to the client. Hence, to protect the data being transmitted, the client communicates with the server using a secure OpenVPN (Virtual Private Network) connection. Certificates are used to authenticate communication between the server and client machine.

The client provides various services such as maintaining personal and case related information about a youth, generating reports, keeping track of expenditure, etc. Each agency had its own set of requirements, be it the type of data stored for a youth, the format of reports generated or the amount of data stored. And as the system grew, the application at each agency started to differ drastically from others. The different versions became a drawback since we had to keep track of each version, and when some services in the clients overlapped, there was redundancy in maintenance and development.

Another hitch in the initial design was the fact that a client could connect only to one database through the server it was customized for. This prevented cross-county data access through the same interface. Users had to access multiple versions of the client application installed on their system to get data from other counties. This complicated matters for youth court counselors working for multiple counties.

Server

The server, written in C++, is the interface between the client and the database. Just as in any client-server architecture, the server accepts and processes requests from the client and replies with relevant data. In our system, we have a set of requests that perform certain operations. Each request is associated with a function of the system, like adding a user (request: ADD_USER), adding a case to a youth record (request: ADD_CASE), editing the docket number assigned to a youth (EDIT_DOCKET_NUMBER) and so on.

As the needs of each county diversified the clients, servers that provided database access to these clients had to undergo changes to meet the new requirements. This led to a scenario where multiple versions of the server were running on different ports out of the same server machine. This led to the same maintenance issues as with the clients.

EDIT_DOCKET_NUMBER Protocol

This request lets the user edit the docket number of a youth's case if they have the capability to do so. The user can either specify the new docket number explicitly or just choose the new year, in which case the server calculates the new docket number based on the year.

Parameters

String: Current docket number
String: New docket year
String: New docket number

Reply

String: New docket number

Related

EDIT_CHILD_NUMBER

Table 1: An example protocol of the communication between client and server

Database

The pivotal component of the system is of course, the database, where all the data we process is stored. We use a MySQL database, which is a reliable, easy-to-use and high performance relational database [1]. The server communicates with the database via ODBC (Open DataBase Connectivity) connections. The database resides in the same machine as the server code, and hence, there are no security issues in communication (since it does not go over a network).

Since the initial database was designed with one particular agency in mind, the tables in the database did not have county associated information to distinguish data from different counties. So as we spread to other agencies with different needs, whole new databases were created to store their data. This solved the quandary of mixing up data from different counties but created new problems. On the same server machine, we had to maintain multiple databases associated with different agencies. With time, the structure of each database evolved to meet the growing needs of differing counties, resulting in database with schemas at different levels of complexity.

UNIFICATION PROCESS

This section describes the multi-stage unification process of Sword. Starting with the most generic of the different versions, the structure of the server, client and the database schema was changed to facilitate unification. Unification of the databases is discussed first, since it forms the basis for the unification of client and server.

Unifying the databases

Before unifying the server or client, the database schema had to be changed to support data from multiple counties. As already mentioned, the initial database design did not store county related information in each table. The following image is a snapshot of a portion of the initial database schema.

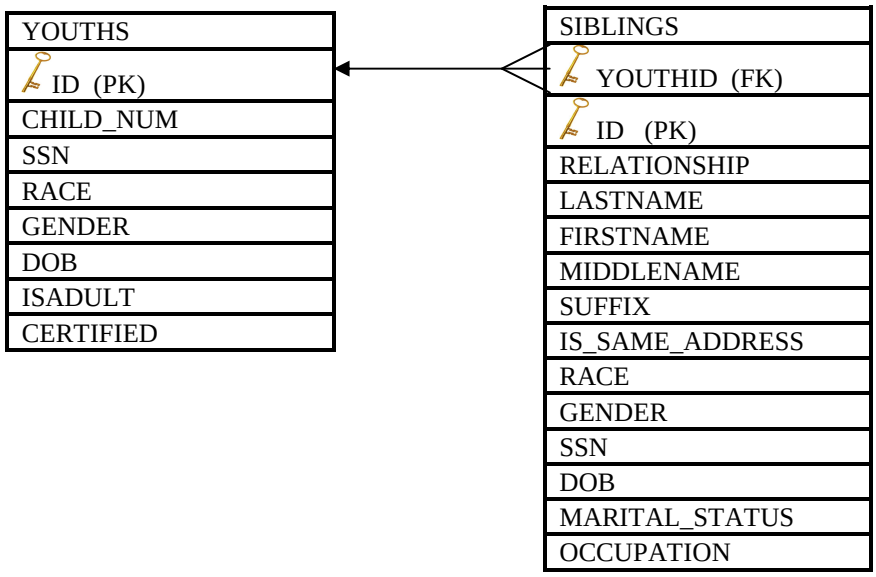

Fig 2: Database schema before unification

None of the tables stored information regarding the county the data belonged to. Fields like Youth ID formed the primary keys. The problem with this design was that youth records from different counties could not be stored in the same database. Two youth records from different counties could have the same ID since the ID needs to be unique only within the county, but when they are stored in the same database, it would result in an error since there would be two records with the same primary key.

To solve this problem, county associated data was stored in each table. A new field called the "DID" (Department ID) was introduced. Each county had a DID, and the primary key of each table was converted to a complex key (DID, <old_primary_key>). Now, records from different counties could be stored in the same database without conflicting primary keys.

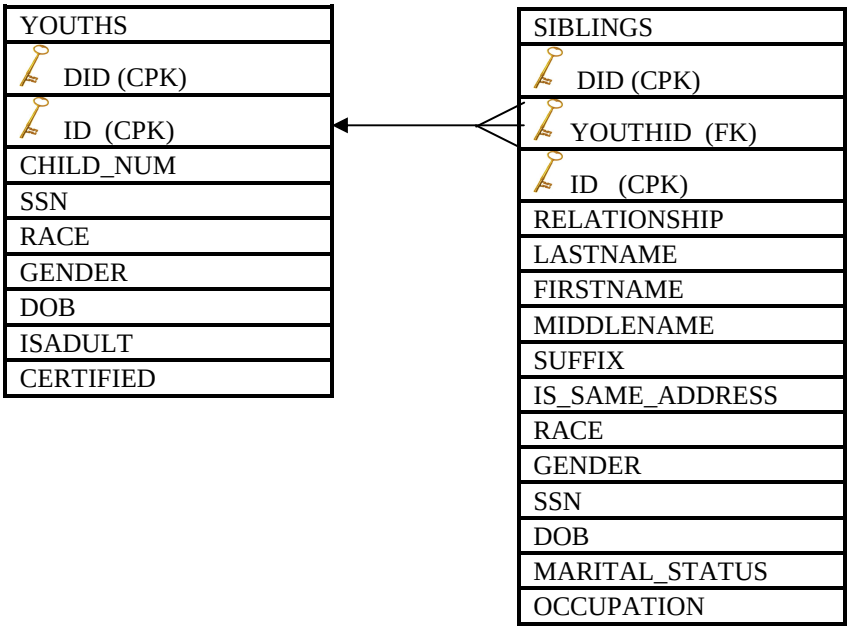

Fig 3: Database schema after unification

This new design also enabled cross-county data access. Unlike the earlier design a single database connection allowed inter county data access, which simplified matters by precluding multiple ODBC connections. To avoid unrestricted inter-county data access, a list of other 'departments' or counties an agency had access to was stored. For example, the Forrest Youth Court users can access data from Forrest Detention Center, but other Youth Courts cannot.
Now that data from different counties could be stored in the same database and inter-county access was enabled, focus was shifted to unifying the server and client. Unifying the server and client meant merging all the features and services of differing versions into one single application. Before that was done, the system needed a way to make sure that each user could access only those services relevant to his/her agency that they had permission for. Introducing the concept of 'roles' and 'capabilities' into the system facilitated this.

As already described, services provided by the server are associated with 'requests'. These requests were initially stored as part of the server code, and the server referred to it to process requests from the client. This concept was extended by storing these requests in the database as a list of 'capabilities' in a table in the database. This list formed

the super-set of all the services provided by the server. Certain 'roles', such as admin, counselor (for Youth Courts), jailers (Detention centers) and nurses were defined. Each role had associated with it set of capabilities which formed a proper subset of the super-set table. So when a user is added to the system, he/she is assigned a role and they have access to services based on their list of capabilities.

The database has already been modified to handle more than one county, now they had to be populated with data that was stored previously on separate databases. The main hurdle is to match youth records across counties. Over the years a youth might have been booked in several agencies, and each agency would have maintained a different record for the youth. Records that belonging to the same youth had to be identified based on his SSN (Social Security Number), or a combination of other details such as name and address, and merge these records.

Unifying the servers

Unifying the servers involved merging the codes for different versions of the server, and developing one server that would process data and perform services for multiple counties.

One of the first steps in unifying the servers was to change the server code to accommodate the changes in the database. When a user logs in to the system, the server retrieves and stores the 'DID' of the county the user belongs to. So for that particular session, the server knows which county the user belongs to and this information is used while inserting and retrieving information from the database. This step involved changing the SQL statements used to communicate with the database, to include the new fields wherever necessary.

The next step was to process the information provided by the roles and capabilities tables. Each time a user initiates communication with the server, the server communicates with the database and retrieves the subset of capabilities the user has based on his/her role. The next section explains how this subset of capabilities helps in providing agency specific access.

To unify the services provided by different servers into one, code that serviced requests from different counties was merged into one project. Defining services using requests made it easier to distinguish agency specific services. After confirming that none of the requests from different counties were in conflict, the server functioned as before, servicing requests from the client.

Unifying the clients

During the phase of unifying the various versions of the clients, the end goal was, to the client should look the same after unification as it did before. Users shouldn't be expected to take care of new entries, and they should have access to the same features as before. In the merged client, the user had to have access to the features he/she has the capability/permission for and the other features were hidden from their view.

Since each user had a 'role' that defined their level of access, this concept was used to determine what is visible to the user and what is not. As explained above, when a user logs in, the server stores the sub-set of capabilities the user has. To enable proper level of access, the server sends this list to the client. Based on this list of capabilities, the client determines which features are required for the current session, and only those are displayed. Hence the user can access only a sub-set of all features incorporated in the client.

This concept of associating the services of a client with user's capabilities allows a user to access various features of the client without strict demarcation between agencies. For example, if 'counselors' need to access some report that was initially designed for 'jailers', the capability to access the report had to be to the counselors' list. This way, changes are required only in the database and the server and client remain unchanged. Next time a 'counselor' logs in he/she has the capability to access the report, and the client displays that option to the user.

Inter county data access is enabled in 2 ways in the merged client. A user can have different user names associated with each agency they have access to (each user name will have a different DID), and using the same client, the user can now log into the system and access data from different counties.

Also, if all users of a particular agency need to access data from another agency, this information was included in the database's inter-county data access table, and the server automatically accesses that data from the database, enabling a user to access inter-county data in a single session.

CONCLUSION

This paper introduced Sword, a client-server E-government application that aids Youth courts and Detention centers in keeping track of juveniles. As the system developed, it branched out to several customized versions for each agency. For continued and effective growth of the system, the different versions had to be merged. Most of the steps for Unification mentioned here have been executed successfully. The merged version is still being tested and will hopefully be deployed soon. The merged version will be valuable to the agencies since it will allow inter county data access from a single application and it will give them the option of easily accessing services provided for other agencies. It will also reduce a lot of time and redundant work on maintenance for us, the developers.

REFERENCES

[1] "Inside MySQL 5.0, A DBA's perspective", A MySQL Business White Paper, October 2005.

[2] CSC Health Services, "St. Vincent's and CSC Merge Diverse Systems into One", A case study. 2003. http://www.csc.com/industries/healthservices/casestudies/2550.shtml (November 2006)

[3] Middaugh, Kris., "SWORD of Justice", Government Technology (Crime and the Tech Effect), July 2004. http://www.govtech.net/magazine/sup_story.php?id=90807&story_pg=2 (November 2006)

[4] Rehner, Tim., Seyfarth, Ray., Zhou, Hong., and Forster, Michael., "SWORD: An integrated information management system for practice in juvenile justice settings", Council on Social Work Education Conference, February 2004.

Vinitha Muraleedharan

Vinitha Muraleedharan is a Graduate student pursuing her Masters degree in Computer Science at School of Computing, The University of Southern Mississippi. She earned her Bachelor of Engineering Degree in Information Science and Engineering from Visvesvaraya Technological University, Bangalore, India. She started working on the SWORD project in May 2006.

Dr. Andrew Strelzoff

Andrew Strelzoff is currently an Assistant Professor in School of Computing, University of Southern Mississippi. He earned his Ph.D. in Computer Science from University of California, Santa Barbara in 2004. His research interests are: FPGA computing, Radio Frequency Identification (RFID).

Dr. Tim Rehner

Tim Rehner is a Professor and the Assistant Director of School of Social Work at University of Southern Mississippi. He earned his Ph.D. from the School of Social Work, The University of Alabama, Tuscaloosa, in 1994. He is the Co-Founder & Director of Family Network Partnership - a university-affiliated community based delinquency prevention agency that initiated the SWORD project.

Dr. Ray Seyfarth

Ray Seyfarth is currently an Associate Professor of Computer Science in the School of Computing at the University of Southern Mississippi. Prior to his current appointment, he completed a Ph.D. in Computer Science at the University of Florida in 1989. His teaching interests include UNIX, algorithms, graphics, compilers, programming languages, operating systems, formal languages and databases. His research has spanned many disciplines through collaboration with polymer scientists, physicists, oceanographers, biologists and mathematicians. His current funding is in juvenile justice software and image mosaicking.