An Architectural Overview of SPK

Alan Westhagen
Software Architect
RFPK Software Team Leader

The System for Population Kinetics (SPK) is software being developed by the RFPK Laboratory, in the Bioengineering Department of the University of Washington, under grant *P41 EB-001975* from the National Institutes of Health (NIH) of the U.S. Department of Health and Human Services. This whitepaper describes its architecture.

Table of Contents

What the Architecture Supports

This section describes the requirements that the SPK architecture is designed to satisfy.

The Problem to be Solved

To better understand and to predict the operation of biological systems, researchers design mathematical models which they believe will closely conform to observations. The equations that comprise these models contain parameters whose values must be determined by "fitting" the equations to real-world data. In the biomedical field of population kinetics (PK), a methodology known as *nonlinear mixed-effects modeling* has been very successful. This technique, when properly applied, can provide models which accurately project the behavior of an entire population from a small number of observations collected from a small sample of individuals. This makes this technology very attractive in medicine and pharmaceutical development, because collecting data from living subjects is a difficult, time-consuming, sometimes risky and, inevitably, expensive procedure.

The Computational Challenge

From a computational stand-point, the process of developing nonlinear mixed-effects models presents some interesting challenges.

- Computing the parameters is highly processor intensive. In developing a model, it may be necessary to perform the computation many times. On a very fast workstation, each recalculation of the model can take a week. On a typical single processor workstation, only one such calculation can be performed at a time. Clearly, it would be desirable to offload the calculation to a specialized computational engine. Going even further, it would be highly desirable to distribute the calculation over a cluster of processors, for parallel computation.
- Although a high performance cluster is the ideal platform for model computation, it is unacceptable, by itself, as a platform for model design and results analysis.

Model design is a creative process, and the user needs appropriate tools to enhance this process. The preferred platform to support these tools is the same one the user employs for other creative work, such as writing papers, preparing presentations, communicating with colleagues, etc. In nearly every case, this platform will be a personal computer workstation or laptop running the user's preferred operating system, whether Windows, Macintosh ,or Linux.

- Internet connectivity is just as important as her workstation or laptop to the creative designer of PK models. While a model's parameters are being computed, the user will be working on other models or unrelated work, but will need to monitor the progress of her PK jobs as they are processed to completion. The user should be able, for example, to submit a job for computation at her office, obtain results and resubmit a modified model at an airport en route to a conference, and receive the completed work upon arrival at her destination.

SPK is a Web Service

SPK is a service delivered to researchers via the world wide web, which enables them to develop, compute and validate PK models quickly and efficiently. The service is offered to a designated set of users by an SPK service provider. At the present time, the only SPK service provider is RFPK itself, but in the future, many additional service providers, located at both academic and commercial research laboratories, are envisioned.

Although SPK can run on a single machine, its real power can only be realized when a high performance computational cluster is involved. Computer resources cost money, hence the SPK service will not be free. We believe that competition between service providers will, however, insure that pricing will be fair. In part to encourage the establishment of as many SPK installations as possible, SPK software will be distributed free-of-charge under an open source license.

The User Perspective

The user's access to SPK is provided by two graphical user interface (GUI) programs. One of these is simply the web browser of the user's choice, such as Internet Explorer, Mozilla, or Opera. The other is a component of SPK called the Model Design Agent (MDA).

The MDA is written in the Java language, so that it works under nearly any operating system, including Windows, Linux and Macintosh. The first time the user accesses an SPK installation with her browser, she is lead through the downloading and installation of the MDA and, if not already present, the Java runtime environment. This only takes a few minutes. At the beginning of subsequent sessions, she does not need to download any software unless updated versions are available, in which case, the download and installation occur automatically.

The following is an abbreviated list of functions that a user can perform with the assistance of the MDA:

- Specify a new model and save it in a central database maintained by the service provider.
- Update an existing model to create a new version. The new version does not destroy its predecessor. Instead, all versions are preserved by means of a revision control system.
- Import or create a dataset and save it in the central database.
- Update a dataset, creating a new version. Datasets are also maintained by the revision control system.

- Combine a model, dataset or dataset simulation parameters. initial estimates of model parameters and some additional information to create a job and submit it for transformation into an executable program and subsequent running on the computational engine.
- Check the status of all jobs that the user has submitted. At a given moment in time, a job can be
  - In a queue, waiting to be transformed into a C++ program by the SPK compiler.
  - Being processed by the SPK compiler.
  - In queue, waiting to be transformed from C++ into executable binary code and then run on the computational engine.
  - Being processed by the computational engine.
  - At end. Report available.
- Provide a simple and convenient presentation of results.
- Export results in forms required by other analysis and visualization tools.
- Create a new job by cloning a previous job, changing initial arguments, run parameters, method or dataset. For example, SPK currently supports several methods of calculation, including *first order* , *expected Hessian* , *Laplace approximation* and *Monte Carlo likelihood* , which have been listed in order of increasing computational cost. The user might want to start with a quick computation using the first order method, then follow with one of the more expensive methods, using as initial estimates the first order results. The MDA makes this very convenient for users.

The Service Provider Perspective

The MDA, described in the section above, is the *client side* of SPK. Everything else is on the *server side*, which is provided by an *SPK service provider*. A service provider might simply be a function within an academic department. It might be an operation of a a campus-wide department of academic computing. The service provider might be part of the IT department of a commercial lab or pharmaceutical company. It might be a service provided by a consulting organization to its clients.

There are, inevitably, costs involved with providing SPK service. In some cases, these costs are simply absorbed in departmental budgets or research grants. In other cases, costs will be billed back to users. Commercial operations may want to build a profit into the fees that they charge. At present, SPK software does not contain any functions that directly support billing. Information that could provide the basis for billing or charge-back is, however, present in the database.

SPK does not require any predetermined set of computers. It is, in fact, possible to run the entire system on a single machine. Other than for demonstration purposes, however, using a single machine does not make sense, because this brings insufficient computational power to bear. At RFPK, the server side consists of two server machines, plus a computational cluster.

In discussing Internet software, one is just as likely to apply the designation of *server* to a program as to a machine. For example, the Apache Web Server is software, not hardware. It is common for more than one software server to run on a single machine. It is also common for a software server which requires a lot of resources to be replicated on multiple machines in order to share the load.

From a software point of view, the server side of SPK consists of an application server, computational server, a web server and a database server. At RFPK, the web server is a separate machine for security reasons, the application and database server share a machine, and the computational server runs on a computational cluster.

All SPK server-side software is developed using the GNU software development tools to be portable to any Linux or Unix environment. Unlike the client-side MDA, the server side cannot be run on Windows and Macintosh systems.

Installing and operating the server side is non-trivial. In the development of SPK, a strategic decision was made to move as much of the inherent complexity of the system onto the server side as possible, in order to maximize the convenience to the end user. The server side is no more complex, however, than comparable web-based applications, and well within the competence of system administrators typically found in university departments or commercial research facilities.

SPK is distributed free-of-charge under an open source license. This distribution strategy reduces the risk to service providers, because they can modify or continue the development of SPK independent of RFPK, if necessary. On the other hand, RFPK eagerly encourages collaborative development by all other interested parties.

The Software Developer Perspective

RFPK uses the GNU GCC compiler and many related tools to develop most of the SPK server-side software, and the Standard Edition of Sun Java-2 for the MDA. The MySQL relational database management system is a key component. Programming languages include C, C++, Perl, and Java. Source code is managed by the CVS revision control system. All of the computers used in software development at RFPK run RedHat Enterprise Linux, version 3.

One of the reasons for distributing SPK under an open source license is to encourage cooperative development of the software. Any organization or individual has access to the source code and the same open source development tools used by RFPK. A development and test environment can be set up very inexpensively on a few PCs. RFPK is eager to work with any serious co-developers.

Current Status and Future Directions

At the present time (July, 2004), SPK would have to be classed as a *beta* product. The only service provider is RFPK itself, and the only users are scientists associated with the lab. It will be at least nine months before source code will be well enough organized for general release. The system is being tested in end-to-end mode and the initial results look good.

In the coming months, these are some of the capabilities that will be added:

- Support for model functions expressed as differential equations. (At present, only algebraic functions are supported).
- Support for parallel computation.
- Open source distribution of the software, via the Internet.

Architectural Strategy

The design of SPK is guided by certain architectural principals:

1. *Emphasize user effectiveness.* The user's time is very valuable. Don't give the user a product that is hard to install or hard to use. Trade increased complexity on the server side for simplicity on the client side.

2. *Build security into every stage*. SPK employs public key authentication and the secure socket layer (SSL) security for connections across the public Internet, password security and session security for the MDA, and firewall security for the server network.
3. *Incorporate only open source software*. This is, of course, a requirement for open source distribution. Equally important, however, is the access that this policy affords to a wealth of open source components which can be freely incorporated into SPK without the need to expend resources negotiating commercial licensing and pricing issues.
4. *Minimize dependencies between major modules*. It must be possible to modify a module with little or no impact on other modules. In practical terms, this means that all communication between modules must be tightly controlled, fully documented, and narrow-band.
5. *Employ existing technologies whenever possible*. Among the technologies employed are the MySQL relational database management system as the channel for all inter-server communication; the Gnu versions of Lex and Yacc for building the SPK compiler; the Xerces XML parser; Sun Java-2 and the Java Runtime Engine; the NetBeans IDE for building the graphical user interface; Java WebStart for rich-client java technology; the secure socket layer (SSL) to encrypt Internet communications; RCS for version control.

Software Architecture

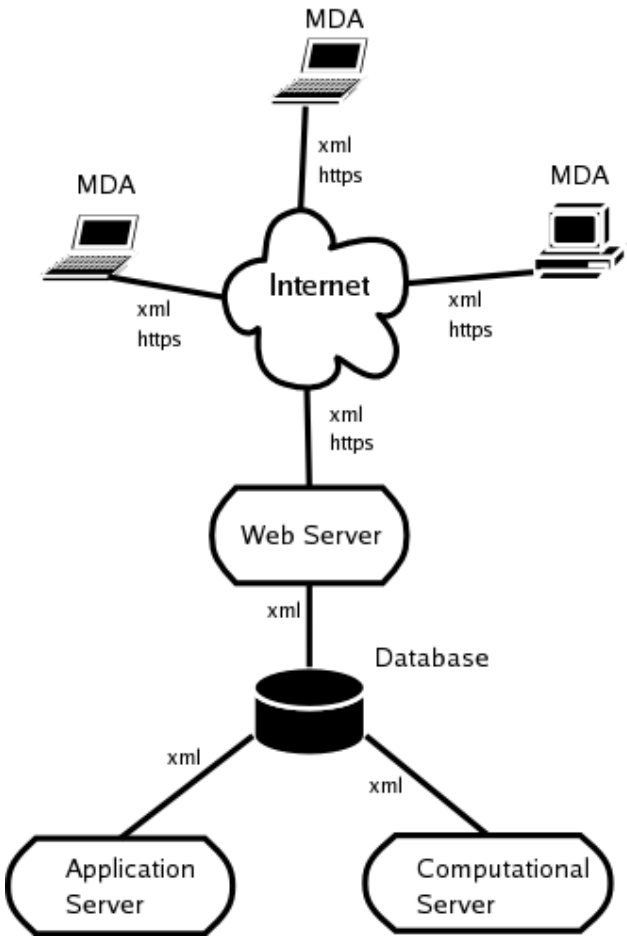

Figure 1. Software Architecture

Centrality of the Database

Architecturally, SPK software is built around a relational database. Every significant change in the state of the system is recorded there. For instance, a user's models and datasets are stored in the database. An implication of this is that the user can employ one workstation at the office, another at home and a third on the road without ever having to "synchronize" them. Because models and datasets are stored in a version control format, all versions of a model or dataset, from the first to the most recent, are available.

To have SPK compute model parameters, a user *submits* a job. Behind the scenes, the MDA passes a request to the web server which, in turn, creates a new row in the *job table* of the database. From that point onward, the application server and the computational server access the database for all of their inputs and outputs concerning that job. Finally, the computational results are retrieved from the database by the MDA via the web server.

The use of a database as the sole medium of communication between the web server, the application server and the computational server is a powerful expression of the concept that inter-module communication paths should be very narrow and well defined, so that the modules themselves can be as independent as possible.

The most important database fields concerning a job are formatted in XML, and conform precisely to specifications that SPK has developed and which are part of the open source release. This is an example of the leverage obtained from the use of widely accepted techniques and standards. Because SPK employs XML, it was not necessary to develop a special-purpose parser. Instead, the widely used Xerces parser is employed.

The Lifecycle of a Job

Another pillar of the architecture of SPK is the concept of job lifecycle. Milestones in the life of a job are implemented as events in a finite state automaton, implemented by the three servers and the database.

As shown in the following diagram, the web server submits a job that it has received from the MDA by adding a new row to the *job* table of the database and entering the value *q2c* (queued to compile) in the the *state_code* field of that row.

Meanwhile the *compiler daemon*, which is a part of the application server, will have noticed the appearance of a new job with *queued to compile* status. When sufficient resources become available to the application server so that it can compile another job, the daemon will start an instance of the SPK compiler to transform the job specifications in the database into C++ code. It will change the *state_code* field to *cmp* (compiling). If the compilation completes without error, the daemon stores the resulting code in the job's database row and changes the *state_code* field to *q2r* (queued to run).

As soon as the entry is changed in the database, another daemon which is part of the computational server will notice it. In due time, when resources become available and when the job is the highest priority job waiting with *queued to run* status, the *runtime daemon* will change the *state_code* field to *run*, will have the C++ code compiled and linked to form an executable binary and will start it running on the computational cluster.

When the job's run is complete, the runtime daemon will store the results into the database and will change the *state_code* field to *end*.

The next time the user checks the status of his jobs, either on his personal SPK web page or with the MDA, he will notice that a job has results available. He will use the MDA to transfer a copy of the results to his workstation for analysis, presentation, or export to other tools.

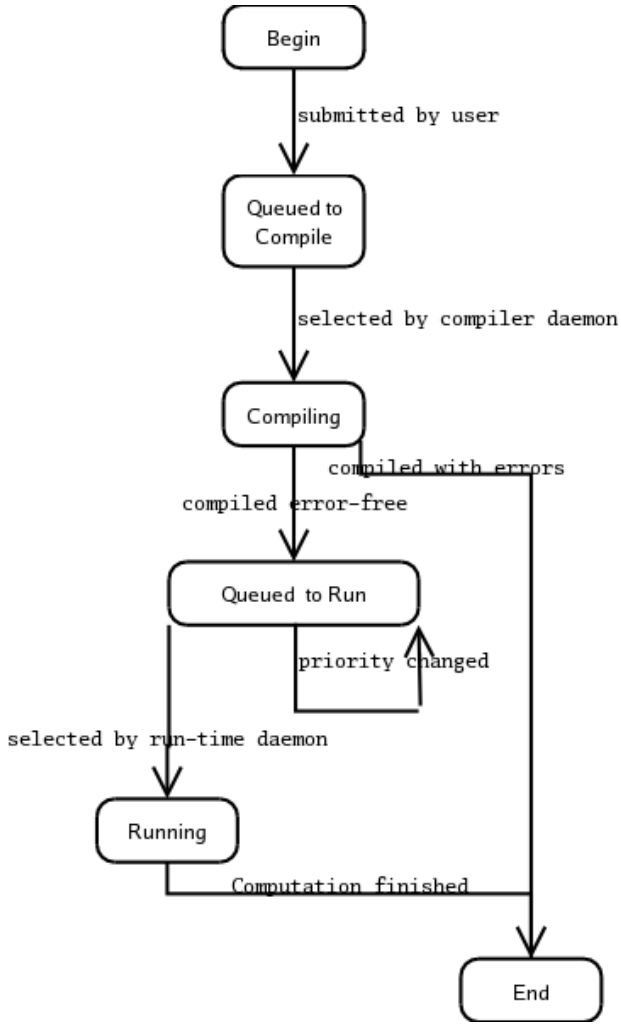

Figure 2. Job State Diagram

Design of the Database

SPK employs the MySQL relational database management system to provide the database which is central to its architecture. MySQL is very widely used on the world wide web. It bears an open source license when distributed as part of a system such as SPK which is also distributed as open source. Recent versions of MySQL support atomic transactions, which is required by the lifecycle automaton because changes in job state require simultaneous changes to multiple tables.

As explained above, the database is the central communications path between servers. From the user perspective, it provides persistent storage for models and data, as well as a complete history of his jobs. From the point of view of the service provider, the database maintains a history of services and the users to whom they were provided.

In the diagram that follows, the rectangles represent the most important tables in the database, while the lines that connect them represent relationships. The arrow head at one end of a connecting line indicates the direction of the relationship. For

instance, the fact that the arrow head on the line between job and user points toward user implies that the user *has* a job, rather than the other way around. The crow's foot symbol at the other end of the line indicates that a user may have zero or more jobs. If a line has no crow's foot, a row of the table at the head of the arrow *has* exactly one row of the table at the foot of the arrow.

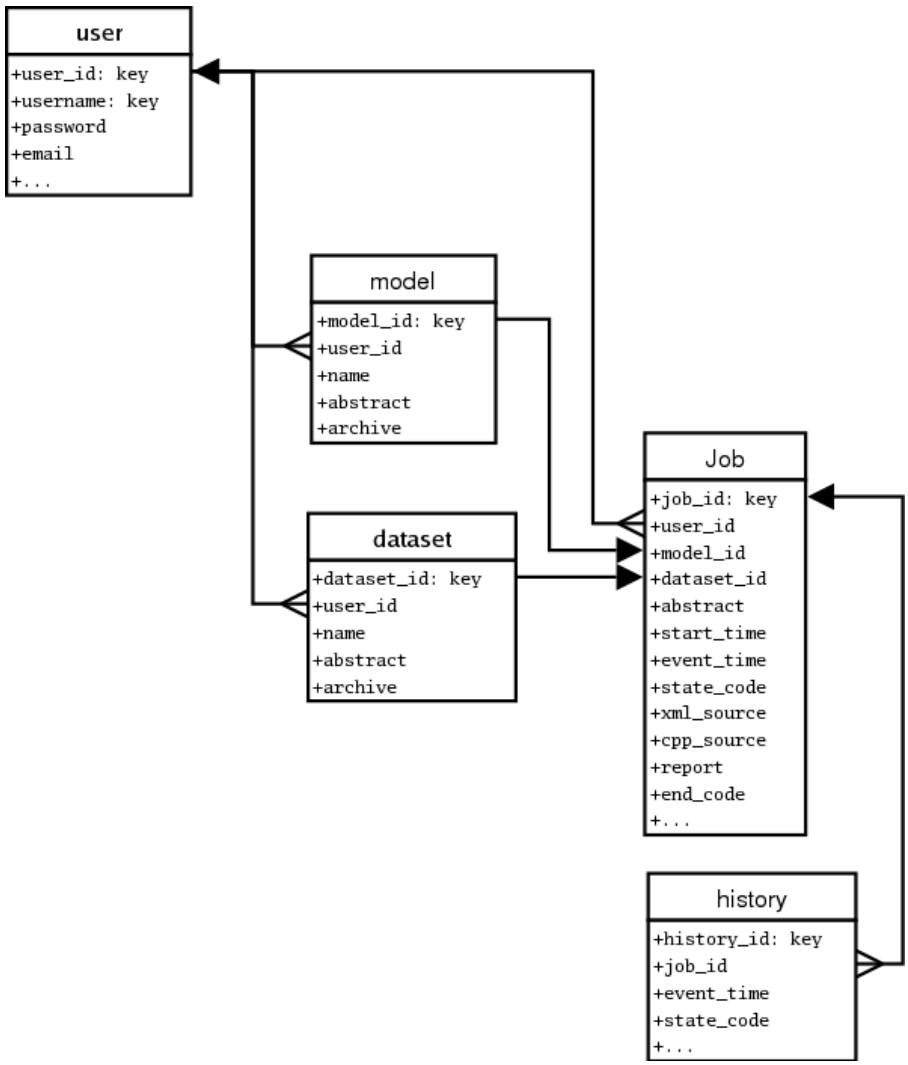

Figure 3. Database Diagram

The Application Server

The Application Server for Population Kinetics (ASPK) is one of the three logical servers that comprise the server side of SPK. Its main components are the SPK compiler and the compiler daemon.

SPK Compiler

If it were not for the SPK compiler, users would have to express their models as C++ programs. The compiler frees them from this very time consuming and error-prone activity by automatically generating the C++ program from specifications they input into the MDA. The resulting program can subsequently be compiled by the GNU GCC compiler and linked to several libraries to create a highly efficient binary program for run on the computational engine.

Users can include Fortran-77 statements in their model specifications. The portion of the SPK compiler that translates Fortran into C++ was, itself, automatically generated by the GNU versions of two classic compiler building tools: the *Lex* lexical analyzer and the *Yacc* compiler-compiler.

The SPK compiler does much more than translate Fortran into C++, however. It creates an entire driver program which incorporates the users initial estimates, runtime parameters, and output specifications.

The compiler is very efficient. On average, it takes only a few seconds to compile a job.

The Compiler Daemon

A Daemon is a program which runs continuously to provide round-the-clock services. The SPK compiler daemon constantly monitors ASPK resources. When sufficient resources become available to compile another job, the daemon scans the database for jobs with *queued to compile* status, and picks the job with highest priority. If several jobs were queued with the same priority, it picks the one which has been waiting the longest. In practice, the compiler queue is rarely congested, because the compiler takes only a few seconds to process a job.

When the daemon selects a job for compilation, it creates a working directory for the job, copies the *xml_source* field from the database to a new file called `source.xml` and extracts the latest version of the dataset referenced by the *dataset_id* field to create a file called `data.xml`. It then runs the compiler as a child process, passing `source.xml` and `data.xml` as arguments.

Because the daemon is the parent of the compiler, it is notified when the compiler terminates for any reason, whether successfully or unsuccessfully. If the compilation was successful, the daemon creates an archive (using the Unix **tar** command) of the entire working directory, and stores it in the *cpp_source* field of the database. It changes the state_code to *q2r* (queued to run). If, on the other hand, the compilation terminated with errors, the daemon places an error message, plus any output that the compiler wrote to a file called `software error`, in the *report* field of the database and changes the state_code to *end*.

The Computational Server

The Computational Server for Population Kinetics (CSPK) is where the really heavy work gets done. This logical server creates an efficient binary program by employing the GNU GCC compiler to translate the C++ output of the SPK compiler into object code and to link it to the GNU libraries, the SPK library, ATLAS, and CppAd. It then runs this binary on the the computational engine.

GNU GCC Compiler and Libraries

GNU[1] is the organization that invented open source. The development tools that they distribute are among the very best. They are available for essentially all Unix and Linux platforms, which means that SPK can run on all such platforms. The wide

availability of these tools provides strong encouragement for cooperative development.

The SPK Library

The SPK library is the result of years of cooperative development at RFPK between a scientific team, a mathematics team and a software team. This C++ class library encapsulates most of the logic required to do the PK calculation. It was originally released by itself, but was found to be impractical for most scientists to use because of the difficulty of writing the required driver program. Now the driver is automatically generated by the SPK compiler.

Atlas

The Automatically Tuned Linear Algebra Software (ATLAS)[2] package provides a C language implementation of Basic Linear Algebra Subprograms (BLAS), and a small subset of a C implementation of Linear Algebra Package (LAPACK), along with an installation program that automatically tunes the software for the particular processor that it will be running on. This software is open source.

CppAd

Bradley M. Bell[3], member of the RFPK mathematics team, has developed an automatic differentiation package on his own time. This excellent package is a strong demonstration of the power of open source. CppAd relieves the user from having to provide derivatives of model functions.

The Runtime Daemon

The runtime daemon continuously monitors resources available on the cluster and compares them with needs of jobs waiting in the run queue. When sufficient resources become available, and balancing the need to efficiently allocate cluster resources with the priority at which jobs were queued and how long they have been waiting, the daemon selects a job to run and changes the *state_code* field to *run* (running).

From the *cpp_source* field in the database, the daemon recreates on the cluster the job's working directory, as it was at the end of processing by the ASPK. It then calls upon the GNU **make** command to compile the ASPK-generated driver and link it to GNU libraries, the SPK library, ATLAS, and CppAd, to create a binary tuned to the processors of the cluster. The daemon then starts the binary running on the cluster as a child process.

When either the *make* process or the *run* process terminates, the daemon is immediately notified because it is the parent of these processes. It creates a file called `report.xml` from the output, then writes the contents of the file to the *report* field in the database. It changes the *state_code* field to *end*.

The Web Server

The web server provides the interface between the user with her browser and MDA, and the rest of SPK.

Java-Based Dynamic Content

The SPK web server provides both static and dynamic content. Static content consists of web pages which do not change in response to what the user or her jobs are doing. Dynamic content is created on the fly, depending on the state of the system and the course of the interaction with the user.

Static content is created by the service provider. As a service provider, RFPK presents information about the organization, its directions, and its products. Other service providers present similar sorts of information.

The dynamic content is part of the SPK software In particular, the web server acts as a server-side proxy for the MDA when it accesses the database. Dynamic content takes the form of Java servlets and Java Server Pages (JSP).

Web Security

The web server always uses the *https* protocol, which is also known as SSL, the secure socket layer. All transmissions between the user's workstation and the server are encrypted.

In addition to encryption, the web server employs certificate based public key server authentication. This allows a user to log into an SPK service with confidence, knowing that the system he is connecting to is not an impostor.

Java WebStart

Unlike browser applications such as Java applets or JavaScript, the MDA is a full computer program. This allows it to be both more responsive and more reliable than applets and JavaScript are capable of being. At the same time, it is very easily downloaded via Java WebStart from the SPK web site the first time a user accesses the system, and it is automatically updated from that point onward. The downloading and updating are encrypted by SSL for security. Java WebStart also protects the web site from MDA programs that may have been maliciously modified, because it will refuse to run any MDA modified in the field.

Tomcat

SPK uses the open source Apache Tomcat web server platform. Tomcat supports static content, Java servlets, JSP, SSL and Java WebStart.

The Model Design Agent (MDA)

The MDA is a complete editor for models, datasets and jobs. It submits jobs and presents results. In many ways, it is quite similar to a class of modern software development tools known generically as interactive development environments (IDEs). This is no accident. The MDA is built from an open source Java IDE called NetBeans. The MDA is, in effect, an IDE for developing programs which embody PK models.

In the interest of security and simplicity of configuration, the MDA does not access the database directly. Instead, it accesses the database via Java servlets resident in the web server. The user does not have to install any MySQL software on his workstation.

The Internet connection between the MDA and the web server is via the usual IP ports assigned for web browsing; namely ports 80 and 443. This means that users who work from behind firewalls will usually not have to make special arrangements with their system administrators in order to access SPK.

Hardware Architecture

The diagram below depicts the hardware architecture of the SPK service provided by RFPK, except that at RFPK the database server and ASPK server are the same machine.

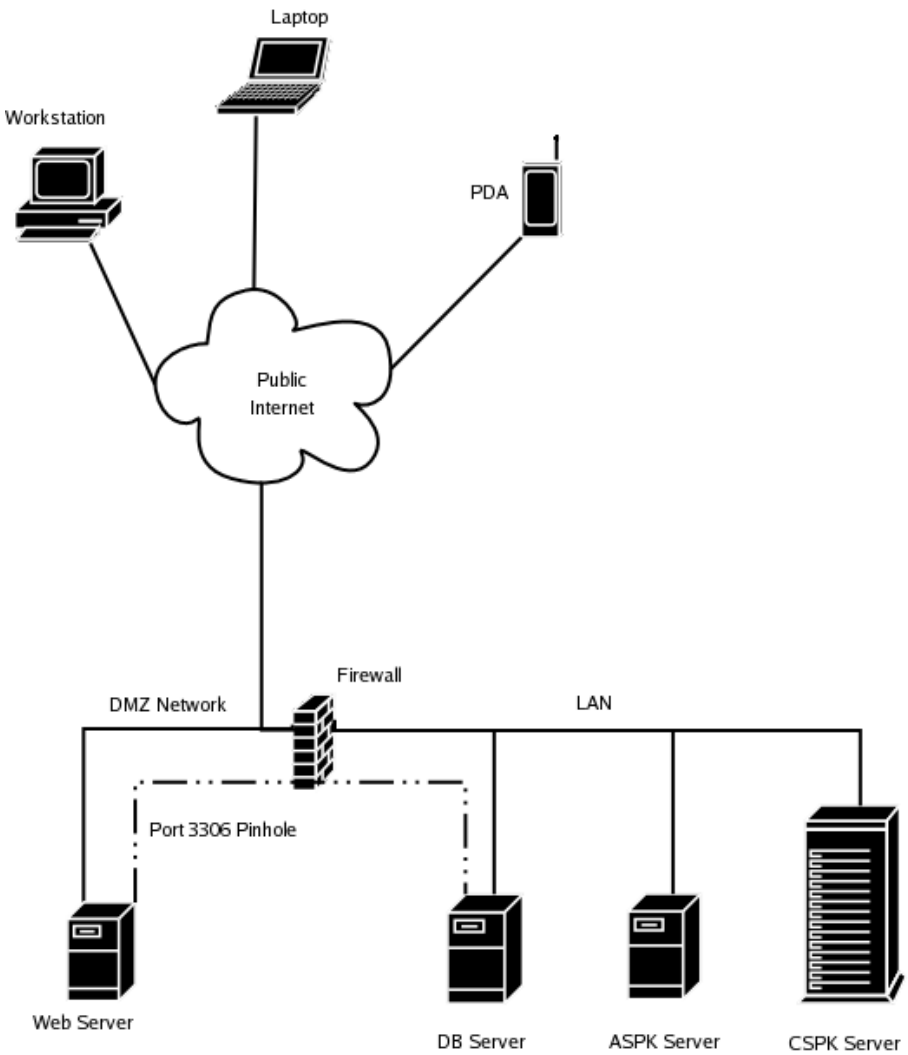

Figure 4. Hardware Architecture

The diagram shows a triple interface firewall. Such a firewall or something equivalent is highly recommended. In fact, the security of SPK depends on it. The firewall and network topology illustrated accomplishes the following:

- Shields the servers on the LAN from all TCP/IP requests originating on the Internet. Servers on the LAN can, however, send requests to the Internet and receive responses.

- Shields the servers on the LAN from requests originating on the DMZ network. In particular, the LAN is protected even if the web server is compromised.
- Passes MySQL database requests from the web server to the database server on IP port 3306. This is the only exception to the previous statement.
- Shields the web server on the DMZ network from all Internet requests except http and https requests. These requests are forwarded by the firewall to the web server and nowhere else.
- Uses unroutable IP addresses on both the DMZ network and the LAN. This increases security by making the servers virtually invisible to the Internet.
- The servers on the LAN can freely pass TCP/IP communications between them. In particular, the ASPK and CSPK have free access to the database server and, due to the protection afforded by the firewall, can do this without need for encryption or certificated authentication.

Notes

1. http://www.gnu.org/
2. http://math-atlas.sourceforge.net/
3. http://www.seanet.com/~bradbell/home.htm