Evan Phelps Response to Scott Sherman RFP: U.S. Patent to Expert Witness Match Evaluator

Overview

The request included a brief problem description, a use case, sample data, task flows, and a proposed algorithm. In this response, I will distill the information into my understanding of the business requirements and suggest a technology solution consisting of a relational database and a simple model-view-controller web application with a well-isolated data access layer to provide flexibility in database implementation details. While the high-level design is straightforward, the hosting and deployment options are more complicated due to considerations of cost, reliability, scalability, and complexity of deployment and maintenance.

In addition to the the material provided to clarify the project, the RFP listed several items to be addressed explicitly. Rather than intertwining the answers within a full technical proposal, I include first a brief summary, and then a direct reply to each of the items (last section, ***Item-by-item response to RFP***).

High-level Problem Description

The high cost of defending against patent infringement suits leads some companies to offer settlement as a less expensive alternative to successfully defending. This is one factor that has encouraged the proliferation of non-practicing entities (NPEs) who position themselves to profit from licensing and litigation of patents rather than from the manufacturing or use of patented inventions. Developing tools that streamline resource-intensive patent defense activities is a necessary component in reducing costs and lowering the frequency of strategic settlements and, in turn, discouraging non-productive patent purchases and “patent trolling.” The proposed database application for finding and ranking expert witnesses for particular patent numbers represents one such streamlining tool.

Business Domain Entities

The current process of mapping a U.S. patent number to a suitable expert witness involves the tedious manual process of referencing case data (“records”), U.S. patent codes (“USPC”), Cooperative Patent Classifications (“CPC”), and associated data. These data entities are related according to Figure 1, which illustrates the following:

- each ***record*** has **multiple** ***cases*** and **1** ***witness***,
- each ***case*** has **1** ***USPC*** and **multiple** ***CPCs*** (call these ***record CPCs***),
- each ***USPC*** has **1** ***CPC*** code equivalent, and
- each ***CPC*** encodes *section*, *scheme*, *class*, *group*, and *level*, which are used in the query-to-record-CPC ranking (previous project, see https://github.com/evan-phelps/patent-tools/blob/master/score-uspto-requirements.pdf).

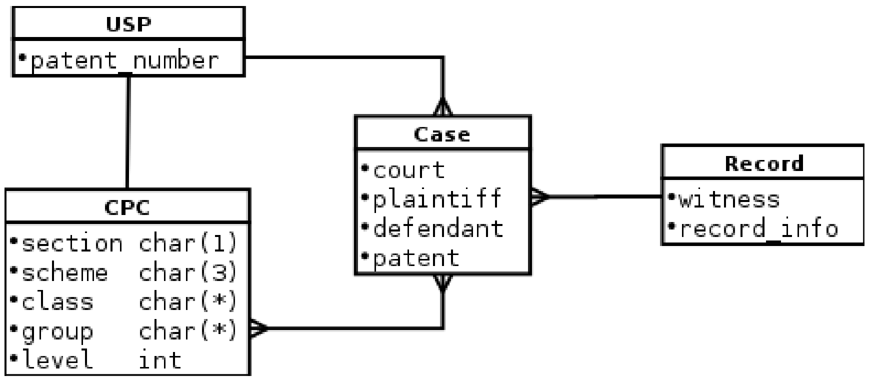

Figure 1: Data entity relations. The diagram may look similar to a database schema, but it is meant only to communicate the relationships between data, not the physical representation of the data. However, some data type information is included, especially if it is encoded within a high-level identifier, such as the CPC and relevant to at the business domain level.

Use Case

I. User enters USPC (***query USPC***).

II. System checks USPC.

  A. USPC is invalid

    i. system returns error and USPC form for re-entry

  B. USPC is valid

    i. system converts USPC to CPC (***query CPC***)

    ii. system scores each ***record*** as follows

      a) system pulls score for each ***record CPC*** (database has score for each combination of two CPCs; recall previous scoring project)

      b) ***record score*** = highest score of all ***record CPCs***

      c) system returns score-sorted (highest to lowest) list of records to user

System Message Sequence

The sequence diagram of Figure 2 illustrates the successful path of the primary use case. Note that there is a separate data access layer, which prevents a database change from propagating through the entire system. If the number of individual records is very large, the loop of database calls can be aggregated similarly (sql “group by”).

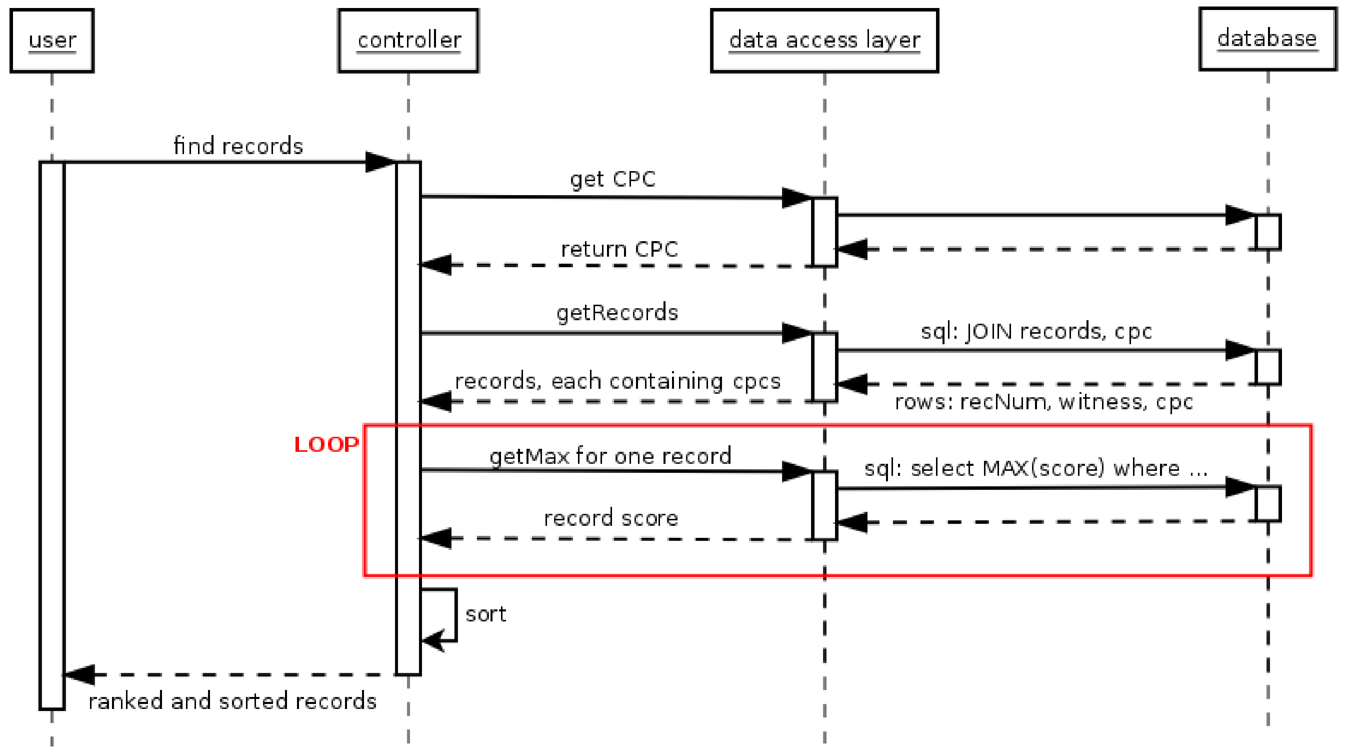

Figure 2: Sequence diagram for successful path of primary use case. It does not describe the implementation-level object, but rather the layers into which each object will belong.

Item-by-item response to RFP

The RFP explicitly listed several items that I will address before continuing with my summary and proposal.

I. "Explain why the spec looks like an efficient solution (or why it is not)"

A. The task flow seems reasonable and efficient, but the flat csv score file poses a performance problem. Large memory resources, a binary random-access score file, or a 50-GB database would remedy the problem, but each has its own drawback. I found no hosting services under the Enterprise hosting solutions threshold that allow for high enough memory allocation; implementing a binary random-access file and corresponding data access layer would add significant project overhead (separate problem analysis would be required to estimate); the most appropriate solution, a 50-GB database, is uncommon on cloud services under the Enterprise price-points. The database option, however, is provided (sometimes by special arrangement) by non-cloud providers that still offer high reliability – I received quotes for between $12 and $50.

II. "Explain tools you want to use (LAMP stack, hadoop, etc) "

A. Based on speed of development, hosting options, and existing libraries/frameworks, I suggest a Python/MySQL-based solution on a Linux hosting solution. The web2py framework provides impressive documentation and out-of-thebox tools to facilitate model-to-database (and vice-versa) mapping and service-oriented approaches (think AJAX). This would allow for more graceful transition to a different data layer or a full service-oriented architecture, in the case it is deployed on the cloud later. The framework is compatible with prominent cloud application providers, like Google AppEngine and Amazon Web Services.

III. "Explain the hosting options you suggest (AWE/ec2, heroku)"

A. This is a difficult question that ultimately requires a full analysis to determine (a) database size, (b) static file storage, (c) number of database-application interactions, (d) traffic, (e) service level requirements (uptime, redundancy, support, etc.), and so on. With respect to the proposed application, most of the factors are non-issues, but the items (a) and (c) tightly constrain the options and lead me to suggest high-quality non-cloud providers. Two that have offered competitive quotes and inspire a level of confidence with my interactions are *Python Anywhere* (https://www.pythonanywhere.com/) and *Web Faction* (http://www.webfaction.com/).

IV. "Estimate how long a project like this would take to build."

A. In reality this would probably be an iterative process, but for the sake of identifying tasks, lets break the estimate into subtasks:

i. (2 hrs) DB design/implementation

ii. (5 hrs) data migration (flat files → RDB) and verification [prereq: i]

iii. (6 hrs) user interface, model, and controller skeleton (this will allow you to step through the use case with fake data)

iv. (8 hrs) integrate model with database (Data Access Layer) [prereq: i-iii]

v. (8 hrs) "sandbox" testing [prereq: iii,iv separately]

vi. (8 hrs) deployment to production environment (low effort after first deployment and environment configuration) [prereq: i-iv]

vii. (2 hrs) acceptance testing in production environment [prereq: i-vi]

B. Total: 39 hours. I think the time allows for correcting issues based on run-of-the-mill unit testing findings.

C. Unless I enlisted help, I would only be able to invest about 6 hours per week on this until the beginning of June when I return from a conference in Spain.