Complexity Research Initiative for Systemic InstabilitieS

FP7-ICT-2011-7-288501-CRISIS

Deliverable D1.2
Database Architecture and Software Design

Author(s)	Eric Beinhocker (UOXF), Artur Matos (UOXF),
Abstract	The project "Complexity Research Initiative for Systemic Instabilities" (CRISIS) is an ambitious three year project for modeling and understanding financial and macroeconomic risk and instability. Development of the CRISIS model will require extensive use of financial data. Currently this data is scattered among the different research institutions belonging to the project, which make it hard for researchers to access, as well as to share new data and intermediate results. This report describes a database for storing financial and economic data and designed specifically for the needs of the CRISIS project. It identifies requirements as described by the researchers belonging to the consortium and describes a potential implementation in detail. The database described here is organized into 3 layers: 1) a CouchDB instance, hosted in Oxford, that keeps data in CSV files, together with some additional attributes for searching it; 2) a small middleware layer that handles authentication and the most common operations; 3) a set of command line tools (or an optional web interface) that interfaces with the middleware to access the data.

Distribution level	Public	**Status**	Final	**Version**	01
Contractual delivery date	31/01/2013	**Actual delivery date**		29/01/2013	

The CRISIS project is funded by the European Commission under the Seventh Framework Programme (ICT)

Table of Contents

1 Introduction

The project “Complexity Research Initiative for Systemic Instabilities” (CRISIS) is an ambitious three year project for modeling and understanding financial and macroeconomic risk and instability. Development of the CRISIS model will require extensive use of financial and economic data. Currently this data is scattered among the different research institutions belonging to the project, which make it hard for researchers to access, as well as to share new data, ensure consistency in data usage and definitions, and share intermediate results. Therefore, it was considered essential, as part of this project, to develop a database to host shared project data.

This report builds on the previous deliverable (D1.1, “Data Needs Assessment and Inventory Report”, which identified the required data sources for this project), and describes in detail a database to hold all required data. It contains enough technical information to allow a programmer to implement the database as required.

The report is organized as follows: section 2 describes the database requirements, as described by the researchers involved in this project; section 3 converts the requirements in section 2 into a specification, describing the user interface to the database; section 4 describes in detail some of the possible technical options for the database, and why CouchDB was chosen from the alternatives; section 5 is the main part of the report and gives all the technical details required to implement the database; and lastly, section 6 breaks the implementation into tasks and gives some time estimates for building and populating the database.

2 Requirements

The requirements for the CRISIS database were determined through interviews with members of each consortium team designated by their teams as “data liaisons” and by group discussions during consortium meetings. We have also benefited from discussions with and reviewing the work of the EC funded FOC project which has designed and built a database with similarities to what is required by the CRISIS project (the FOC team have been generous with their time and agreed to share results of their work as appropriate).

2.1 Functional requirements

The CRISIS project is a collaborative effort across several teams in different research institutions, and therefore being able to share data effectively is important to the success of the project. As described in detail in D1.1 the project is drawing from a wide variety of data to parameterize and test its models, ranging from macroeconomic time series, to microeconomic bank, firm and household data, to financial data. In addition, as the project progresses, its researchers would also like to easily share intermediate results and summarized data output from models. Currently data sharing is done mostly by email or through sharing software like DropBox, which introduces delays in fetching the data, but more importantly, might lead to a proliferation of slightly divergent data sets among the groups. Therefore it is

deemed essential to have a centralized database, where researchers could easily download data and make their data available.

Another advantage of a centralized database is that it can work as a live catalogue of all available data across the groups. The previous deliverable (D1.1) already identified the major data sources maintained by each group, but that can easily become out of date as each group gets hold of new data or generates new intermediate results. If all key data is maintained through a centralized database there will always be an up to date index that researchers can refer to.

In terms of interactions with the database, 3 key operations were identified:

- Retrieving a listing of available data sets, either all of them or restricted by some criteria;
- Accessing data in a research environment (e.g. loading data into models or statistical packages);
- Sharing new data.

These will be described in detail in the following sections.

2.1.1 Retrieving a listing of available data sets

In order to retrieve data, users must be able first to list which data is available, either in its entirely or restricted by some criteria, especially as the number of data sets increase. Conversations with researchers in the different groups highlighted the following criteria:

- Description of the data set (e.g. key words);
- Originating institution;
- User who uploaded the data
- Version date.

These were deemed sufficient for doing efficient searches. More elaborated taxonomies and classifications were also explored, but they were considered too complex and not particularly useful — they would introduce too much administrative overhead for little practical advantage, and likely not to be kept up to date.

A key requirement mentioned by everyone is that the system should be simple to use, and therefore only the simplest of search interfaces was considered. No requests were mentioned for complex Boolean expressions (for example, an OR combination of two search terms), regular expression searches or negative match searches (for instance, searching for an element that doesn't contain the given search string). Researchers expressed a need only for the following:

- Doing a full text search that includes all criteria, e.g. searching for all data sets with "Milano" in at least one of the 4 criteria outlined above;
- Doing a full text search but on a specific criterion, e.g. searching for "Milano" in the originating institution only;
- Doing an AND combination of search terms, e.g. searching for data sets with "Milano" and "GDP".

Besides being able to do the simple searches above, it is considered important that the system lists the data sets with the criteria clearly labeled, so that the relevant data sets can be easily identified.

2.1.2 Accessing data in a research environment

Researchers use the data mostly from analysis and modeling environments like MATLAB or Python, as well as from MASON, a Java-based multi-agent simulation system (with some additional extensions built by AITIA). The database will need to interoperate with these environments.

Usually data is kept in CSV or Excel format and loaded into the environment with a script, which sometimes does some further processing or joining between files. In general, researchers are fairly experienced with loading CSV files, and expect the database to be able to handle them natively. In this regard, just having a simple command line program for downloading data from the database in CSV format is considered enough. Some users also mentioned that would be useful to have the ability to load data straight into the environment, i.e. a MATLAB command to automatically pull data from the database, or having a simple web interface to load data, but those weren't considered essential. However, at a later date as the MABM and FABM model modules are further developed (from Mark II to Mark III) the researchers may decide to create an interface within those model modules for pulling data from the database.

2.1.3 Sharing new data

As with accessing data, researchers use mostly CSV files for sharing data with their peers, either by email or by using specialized sharing software like DropBox. In a similar way to the previous section, having a command line program to upload a CSV file directly to the database is considered enough for most purposes; with some users mentioning having the ability to upload directly from the environment or a website as useful but not essential.

2.2 Data requirements

The data to be kept in the database is also fairly simple. The previous deliverable (D1.1) identifies the data needs in detail, which for the purposes of this report can be roughly classified under the following categories:

- Time series data, for example, LIBOR and EURIBOR rates or GBP M1 monetary aggregates;
- Balance sheet data, e.g. for European banks;
- Cross-sectional data, for instance, matrices with aggregated statistics or runs from multi-agent simulations.

These data types are in essence two dimensional matrices containing mostly numerical data, but sometimes also text (as in the balance sheet data) or dates (as in the time series data). Besides the data itself, time series and balance sheet data also will need to store column names. All of these matrices are fairly small — 1500 rows

per 1500 columns at most — and don't offer any particular challenge to any modern database, at least as far as storage is concerned.

A particular feature of this data is that it doesn't have a regular structure: the matrices have different number of rows and columns, different column names (when they are present), and different types of data for each column. This will be one the major features driving the selection of the database technology for this project, as it will be described later in this report.

2.3 Security requirements and licensing issues

Some data in the database may be proprietary, sensitive, or given to the consortium on a restricted basis (e.g. data from central banks). Likewise some commercial data may only be licensed to a few institutions. So there is a need for secure access controls in place. In general, researchers belonging to the same institution should have access to the same data sets. We discussed the following options with researchers:

- Host-based security, where access is restricted based on the computer accessing the data, i.e. only certain hosts can access the data, and access vary with each host. There is no need for a username or a password, and the user can download whatever data sets the host is entitled to, as long as all access is done through the host;

- User-based security, where access is granted on a per-user basis after authenticating himself, e.g. using a standard username/password combination.

User-based security was considered a better option in general, and that was the option chosen in the end for this database. Most researchers spend a considerable amount of time attending conferences and visiting other institutions, where they might need to use other computers besides their own; this introduces considerable overhead if host based security is used, as the admin user would need to add or remove host access controls rather frequently. Also, host based security was perceived as potentially less secure for researchers using laptops, as they might get stolen and all data sets could be accessible without further authentication. Finally, institutions providing access to sensitive data may feel more secure with logs of who is accessing the data.

As for authentication, a simple username and password combination for each user was deemed enough. Although some of the data is sensitive, it isn't considered sensitive enough to warrant the use of more complex authentication schemes, for instance, cryptographic signatures or two-factor authentication.

Besides authentication, users will need a way to specify who is able to see what data. After discussion with researchers, the general idea is that there is only a very limited number of security levels for the data: public, which is accessible by everyone, and a number of private levels, which would be restricted to specific research institutions. This leads naturally to a role-based security model, where there will be generic roles, e.g. "bank data", that users and data sets will be tagged with. Users can only access data for which they have the same set of roles.

2.4 Integration requirements

Besides being accessible by researchers, there might be a need for software to interface with the database as well. As an example, some of the software being developed by AITIA for this project, e.g. MEME, might need to store simulation results in the database. In practice this means the database will need to be available from Java (as all the software developed by AITIA is in Java) and allow easy access by programs needing the data.

3 User interface specification

In this section we will explore the requirements given above, and convert them into a high-level specification for a user interface. We will start with a possible command line interface, but we will give a specification for a web interface as well. The command line is the simplest interface and the easiest to implement, although the web interface shouldn't require much additional time. Depending on how much time is available, the project Steering Committee can choose between implementing one or the other, or possibly even both.

3.1 Command line interface

The requirements described above map naturally into 3 distinct command line tools: one for searching data sets, another for downloading data, and finally one for uploading.

3.1.1 Searching for data

Searching for data can be done by using a single command, say crisis-search. Passing a single word will default to search it across all attributes described in section 2.1.1, i.e., description, originating institution, and uploader. The following call would search any data sets with the word "index" in either one of these attributes:

```
> ./crisis-search index
Username: mdoughty
Password: *********
ID      Description         Uploader          Institution
5eb34   SP500 Index data    Matt Kingsbury    University of Oxford
53fd1   FTS100 Index data   Matt Kingsbury    University of Oxford
```

The user will need to type a username and password before accessing any data. ID will be a unique identifier for each data set. If the user is only interested in a searching for a specific attribute, it can be included in the command line as in the following example:

```
> ./crisis-search uploader:kingsbury
...
```

Multiple search terms can be given. The following example will return any data sets where the uploader contains the text “Kingsbury” and description contains “index”:

```
> ./crisis-search uploader:kingsbury description:index
...
```

Sentence fragments with spaces can be enclosed in double quotes:

```
> ./crisis-search uploader:"Matt Kingsbury"
...
```

3.1.2 Downloading CSV files

Downloading CSV files from the database follows a similar pattern. Assuming that the user knows the ID he is looking for, he can retrieve the data set as in the following example:

```
> ./download-crisis 53fb23
Username: mdoughty
Password: *********
Data set details:
  ID: 53fb23
  Description: S&P500 index data
  Uploader: Matt Kingsbury
  Institution: University of Oxford
Saving CSV file to "data.csv".
```

It would be quite impractical to have to refer to IDs all the time, so search terms can also be used, in a similar fashion as to crisis-search. The following call will retrieve the data set containing “S&P500" in any of its attributes:

```
> ./download-crisis S&P500
Username: mdoughty
Password: *********
Data set details:
  ID: 53fb23
  Description: S&P500 index data
  Uploader: Matt Kingsbury
  Institution: University of Oxford
Saving CSV file to "data.csv".
```

The command line program will exit with an error if more than one data set matches, to be sure that the wrong data set isn’t retrieved.

By default the file will be saved to “data.csv”, unless an option is given on the command line, as in this example:

```
> ./download-crisis S&P500 -o "sp500.csv"
...
Saving CSV file to "sp500.csv".
```

3.1.3 Uploading CSV files

Uploading CSV files work in a similar way. The file is given as the first argument, and the command prompts the user for details:

```
> ./upload-crisis sp500.csv
Username: mdoughty
Password: *********
Adding a new file.
Please input the details:
  Description: S&P500 index data
  Uploader: Matt Kingsbury
  Institution: University of Oxford
Uploading to http://host/crisis/6e1295.
```

If the user wants to update an existing data set he must give either an ID or a search term as the second argument:

```
> ./upload-crisis "S&P 500" sp500.csv
Username: mdoughty
Password: *********
Updating file in http://host/crisis/6e1295.
Please update the details, if they have changed:
  Description: S&P500 index data
  Uploader: Matt Kingsbury
  Institution: University of Oxford
Uploading to http://host/crisis/6e1295.
```

Similar to download-crisis, the software will exit and report an error if more than one data set matches, in order to avoid uploading to a wrong destination.

3.2 Web Interface

Figure 1 is a mockup for a very simple web interface. It defines exactly the same sort of operations as the command line interface, only in a more user-friendly way. The table in the middle lists data sets, following the same output as the crisis-search command, but each entry has a additional “Download” link in the rightmost column for retrieving the CSV files. When the page loads for the first time, the user will be requested to type in the username and password, and upon successful authentication, the table will display all available data sets for that user; the list of data sets can be narrowed down by defining criteria in the search box. Each criteria is made of a combo box to choose the field to search and a text box with the text to search for that field. There is also a button for uploading data, which when pressed, will display a window for the user to fill in the attributes and choose an CSV file to upload.

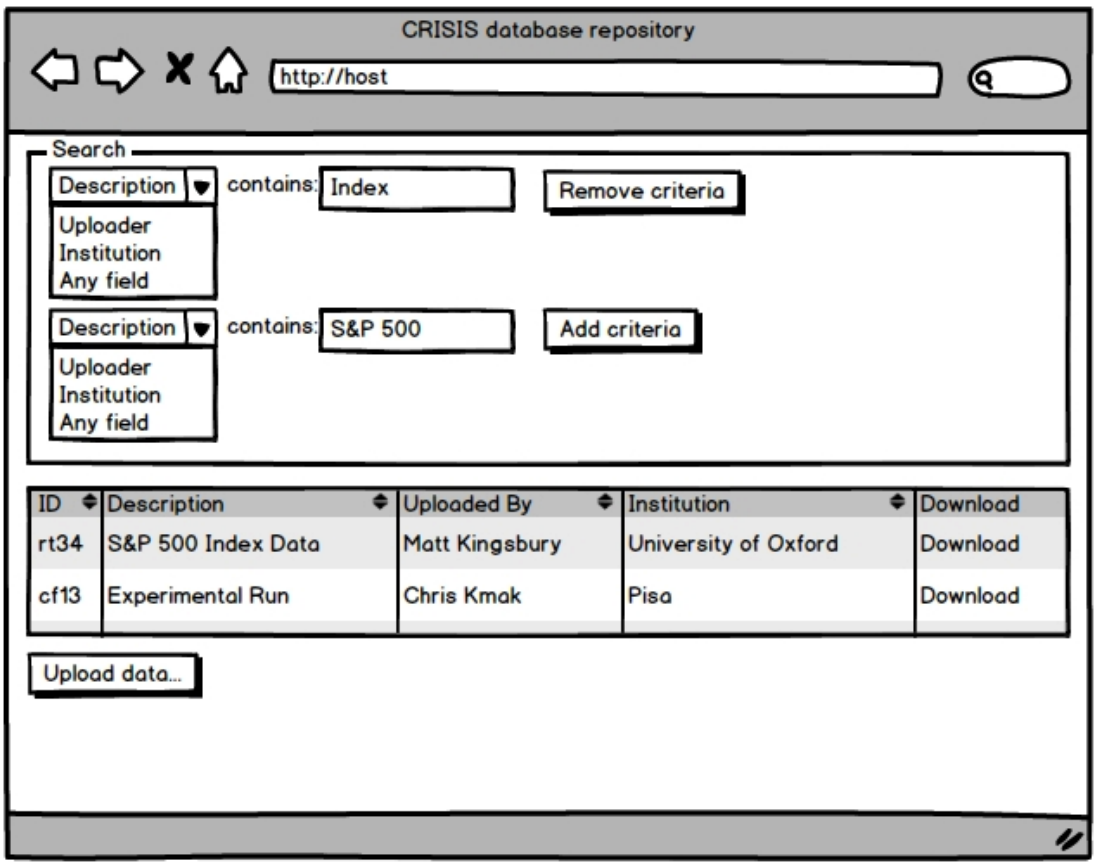

Figure 1: Mockup for a web interface

4 Technology selection and discussion

Given the requirements discussed earlier, there are several database technologies that could potentially be used for storing this data. In this section we will describe the most common alternatives, list their advantages and disadvantages, and decide on the most suitable one.

4.1 SQL databases

Traditionally, data that needs to be accessed in a centralized fashion is usually stored in a SQL database, for which there are several free as well as commercial implementations: MySQL, PostgreSQL, SQL Server, Oracle and Sybase, among others. These are widely used in industry from everything from small web companies to the largest financial services.

SQL databases work by grouping data into tables, which are basically two-dimensional tables with strictly typed columns. For instance, there could be a “Users” table with a “First Name” text column, “Last Name” text column, and an “Age” integer column. By restricting tables to follow a pre-defined structure, SQL databases offer a very nice abstraction for accessing data and joining data between disjoint data sets,

but that also means they aren't particularly suited for data without regular structure, as in this project.

Nevertheless, the data could still be stored in a SQL database if required, by using what is normally called a "key-value" table. These basically pivot the data and represent each value in a table as a separate row, together with its coordinates. For instance, the following matrix:

19-10-2011	1	"Text1"
20-10-2011	0.6	"Text2"

Could be represented in a SQL database like this:

Row Number	Column Number	Value
0	0	"19-10-2011"
0	1	"1"
0	2	"Text1"
1	0	"20-10-2011"
1	1	"0.6"
1	2	"Text2"

This, however, offers several disadvantages, as well as negating most benefits that a SQL database can bring. To start, the "value" column would have to be text in order to keep all the different value types, and additional programming would be needed to convert the values to appropriate types (something that a SQL database usually does automatically). Furthermore, it would also require code to convert back and forth between the usual matrix format and the key-value representation. Given all of this, SQL databases cannot be recommended for this project.

4.2 HDF5

HDF5 (http://www.hdfgroup.org/HDF5/) is a database technology developed by the National Center for Supercomputing Applications (NCSA) and optimized for storing high-volume numerical data. It is widely used for scientific as well as financial applications, e.g. for storing high-frequency tick data.

HDF5 works with single files, which might contain several different databases inside, with each database being a two-dimensional table. These tables can be retrieved as a whole, or accessed by specific rows or columns. They are required to have a pre-defined structure, as in a SQL database, but there is no overhead in adding or removing tables, which makes it much easier to handle data without regular structure.

In contrast to SQL databases, HDF5 doesn't currently offer a client/server mode, and all access must be done through files directly accessible to the data handling process. Data can be shared in a network to a certain extent by having HDF5 files in a network share, but that causes issues with concurrent updates. Another possibility is to implement a client and server on top of HDF5, which unfortunately requires extensive programming work. HDF5 is definitely a better option for this project than a SQL database, but it is not entirely suitable.

4.3 Document databases

A recent alternative to SQL databases are document databases, which offer centralized access like SQL databases but are better at managing semi-structured data. Instead of having tables with a predefined structure, these databases store documents, which are basically lists of keys with associated values. The type of values supported varies with each database, but usually they all support basic types (numeric and strings), lists, and even dictionaries (which allow for nested documents inside a document). Besides the restrictions implied by these basic types, there are no limitations on what type of documents can be stored in a collection — documents don't need to share any structure at all, and the keys don't need to be declared beforehand. They are therefore a perfect match for the type of data the CRISIS needs to support.

Currently, the most popular document databases are MongoDB (http://www.mongodb.org) and CouchDB (http://couchdb.apache.org). They are both open-source and fairly similar in the kind of features they support. We decided to use CouchDB for the following reasons:

- CouchDB has a REST interface built-in, while MongoDB requires a separate component for supplying a REST interface. The built-in REST interface supplied with CouchDB is very convenient and makes it easy to access data from a web browser, shell scripts or even analysis environments like MATLAB, if required;
- CouchDB has a more convenient interface for uploading and downloading files. MongoDB also supports files (via GridFS), but it requires instantiating a driver from a programming language. With CouchDB, uploading and downloading files can all be done via REST. Files uploaded to CouchDB have sensible URLs that can be easily downloaded via a web browser;
- CouchDB makes it very easy to implement simple web applications without additional software. For instance, the web interface described earlier could be easily implemented purely in CouchDB.

5 Technical specification

5.1 Architecture

The database will follow a standard architecture, organized into 3 layers:

Data layer This is the actual data stored in CouchDB. The database will be hosted in a server in Oxford and accessible to all researchers involved in the consortium;

Middleware layer This will be a very small layer built on top of CouchDB and hosted on the same server. It will be responsible for handling security and exposing the 3 basic operations of searching, downloading, and uploading data. All access to the data will be through this layer and no other component will be able to access CouchDB besides this one;

Presentation layer This will the user interface, either the set of command line tools or the web interface. This layer is responsible for translating user interactions into commands to the middleware.
These will be described in turn.

5.2 Data layer

5.2.1 Data set representation

The most natural way to keep data sets in CouchDB is to have one document for each data set and grouped in a single database, e.g. crisis. Arguably, the data could be kept in multiple databases, for instance, a database for each participating institution, but given that the data sets are small, this would only make retrieval more complex without any significant advantage.
Each document will contain the data as well as metadata for searching the documents, as discussed in section 2.1.1. JSON doesn't have a native matrix format but there is no need for one, as CouchDB supports file attachments. Therefore each document will have 2 parts:

1. A data set's metadata and role information is kept in a document and in JSON as key/value pairs, as in the following example:

```
{
"description": "FTSE100 index data",
"institution": "University of Oxford",
"uploader": "Mark Kingsbury",
"data-roles": ["public"] }
```

2. The data itself is kept as a file attached to each document in CSV format. For instance, if the metadata for the particular document above is available at http://host/crisis/6e1295, the CSV file can be retrieved by appending /data.csv to the URL, i.e., http://host/crisis/6e1295/data.csv

5.2.2 Security model

Unfortunately, CouchDB doesn't have a very well developed security model, and therefore security will be partially handled by the middleware layer. All access to CouchDB will be done exclusively by the middleware, that acts as a general interface for any software (including the command line tools) that need to access the data.
Each researcher will have his own CouchDB user, and these will be stored in the usual users database. The middleware layer will be responsible for passing credentials to CouchDB and storing the passwords in hashed format. Besides the user ID and password, the users table will also need to store the data roles each user can access.

5.2.3 REST interface

CouchDB supplies the basic building blocks we need via a REST interface. Following our previous example, and assuming we have a CouchDB server running on http://host and that data is kept in a database called crisis, we can retrieve the list of all data sets by retrieving the URL http://host/crisis/_all_docs/. It returns JSON data like this:

```
{
"total_rows": 3, "offset": 0, "rows": [ {"id": "6e1295",
"key": "6e1295", "value": {"rev": "4
324BB"}}, {"id": "23fa23", "key": "23fa23", "value":
{"rev":"24
41HF"}}, {"id": "4512f1", "key": "4512f1", "value":
{"rev":"74
EC24"}} ]
}
```

Each CouchDB document needs to have an identifier that is unique across its database. It can be any text, as long as it is unique, but here we will follow the usual CouchDB convention of using universally unique identifiers (UUIDs) for all documents. These can be generated from CouchDB itself by retrieving the URL http://host/_uuids.

Assuming we know a data set ID (as for instance, returned by http://host/crisis/_all_docs/), we can retrieve the metadata associated with it by appending the database name and ID to the base URL. For example, if we want to retrieve a data set with ID 6e1295, we would use http://host/crisis/6e1295/, which returns JSON as defined in the previous section:

```
{
"description": "FTSE100 index data",
"institution": "University of Oxford",
 "uploader": "Mark Kingsbury",
 "data-roles": ["public"]
}
```

For updating metadata, we issue a PUT request on the same URL and with the entity being the new values in JSON format, as above. Managing data files follow a similar pattern, except that the name of the file is appended to the URL, e.g. a get request to http://host/crisis/6e1295/data.csv retrieves the data file, and a PUT request with the file contents updates the file.

This covers all basic requirements as described in section 2.1, except for querying. Unfortunately, CouchDB doesn't have full text search capabilities built in, but these can be easily added by using an additional component called CouchDB-Lucene (https://github.com/rnewson/couchdb-lucene). Configuration for this component is fairly straightforward. We will require query URLs for each metadata attribute, as defined in Table 1. More complex queries can also be done by using the full Lucene syntax on those URLs.

Table 1: Some example URLs as exposed by CouchDB-Lucene and required for querying the data sets. Attribute contains the metadata fields, while URL gives an example on how to query on that field.

Attribute	URL
Description	http://host/crisis/_fti/_design/search/by_name?q=index
Institution	http://host/crisis/_fti/_design/search/by_name?q=oxford
Uploader	http://host/crisis/_fti/_design/search/by_name?q=kingsbury
Any field	http://host/crisis/_fti/_design/search/by_any?q=index

5.3 Middleware layer

CouchDB's stock REST interface already does most of the required work, and the middleware will only be a very thin layer on top of CouchDB. It can be implemented in any language, either in a scripting language like Python or even possibly just by using CouchDB or a reverse proxy like Apache. It will handle security and expose more user friendly URLs for common database operations. Stating briefly, these are:

- http://host/list, for querying data sets. It accepts arguments for each of the attributes as defined earlier, e.g. http://host/list?bydescription=index would return all data sets that contain "index" in their description. Multiple attributes can be added as arguments.

- http://host/metadata/ allows to download or update metadata for a data item. Argument can either be a document ID, for instance, http://host/metadata?byid=6e4f or a search term as in the previous command. The command should fail if more than one data set matches. Following CouchDB conventions, a GET request will download data, while a PUT request will update or create a new item.

- http://host/data/ is the converse of the previous operation and allows to download or update CSV files. It follows the same exact interface as the previous command.

All of the above URLs return data in JSON format, and authentication is done by using HTTP basic authentication, in a similar way to CouchDB. In order to make operations safer, all access should be via HTTPS.

The following sections give an outline on how to implement each command:

5.3.1 list

1. Parse the search arguments and convert them to a CouchDB URL;
2. Do a GET operation on the URL and obtain the list of data sets in JSON;

3. Filter any data sets for which the user doesn't have the necessary roles;
4. Return the filtered JSON

5.3.2 metadata

1. If doing a GET request:
(a) Convert the search terms into an URL;
(b) Do a GET operation on the URL and retrieve the list of data sets;
(c) Filter the list on the data sets that the user can access;
(d) If more than one data set is returned, return an error;
(e) Otherwise, return the metadata in JSON format.
2. If doing a PUT request:
(a) Convert the search terms into an URL;
(b) Do a GET operation on the URL and retrieve the list of data sets;
(c) Filter the list on the data sets that the user can access;
(d) If more than one data set is returned, return an error;
(e) Otherwise, update the metadata.

5.3.3 data

1. If doing a GET request:
(a) Convert the search terms into an URL;
(b) Do a GET operation on the URL and retrieve the list of data sets;
(c) Filter the list on the data sets that the user can access;
(d) If more than one data set is returned, return an error;
(e) Otherwise, return the file in CSV format.
2. If doing a PUT request:
(a) Convert the search terms into an URL;
(b) Do a GET operation on the URL and retrieve the list of data sets;
(c) Filter the list on the data sets that the user can access;
(d) If more than one data set is returned, return an error;
(e) Otherwise, update the CSV file.

5.4 Presentation layer

The command line tools and the web interface are just very simple interfaces to the middleware layer, and therefore don't require further implementation details. As in the middleware layer, they can be implemented in any language. The command line tools are very simple and can be just simple shell scripts. The web interface can either be implemented in CouchDB itself, or possibly integrated with the middleware and implemented in the same language.

6 Implementation tasks

Table 2 has a basic breakdown of the tasks required to implement the database, as well as estimates in working days. We assume the project will be done by someone proficient with the technologies discussed here, and it will involve only setup and

programming work, i.e. the programmer will not be responsible for cleaning and converting the data, only to load it in the database.

Table 2: Breakdown of implementation tasks and estimates.

Task	Estimate (Days)
Installing and configuring CouchDB	4
Installing CouchDB-lucene	1
Loading initial data and CSV files	3
Implementing the middleware	2
Implementing the command line tools	2
Implementing the web interface	3

7 Conclusion

In this report we describe a database, together with a set of command line tools and a web interface, to give researchers working on the CRISIS project easier access to data, and to allow them to share it more effectively.

Data needs for this project are rather simple. Users need to be able to search existing data sets based on a few attributes, download CSV files, and upload CSV files to the database. One salient feature of the data is that it doesn't follow a regular structure, and therefore traditional SQL databases aren't particularly suitable for this project; after considering several technologies, we decided to use CouchDB, which is an open source document database with several convenient features.

In the proposed architecture, CouchDB is just one of the layers of the overall database. On top of CouchDB there is a middleware layer responsible for managing security and exposing database operations, followed by an interface layer — this being a set of command line tools or a web interface, depending on how much time is available for implementation.

Based on our analysis, discussion with researchers, and review of work done by other related modeling efforts, the database described here covers all the major requirements of the CRISIS project and will support the project's modeling and policy analysis efforts..

Our next step will be to begin programming, populating, and testing the database in preparation of deliverables D1.3-D1.5.