ADDIS 1.x Retrospective

ADDIS 1.x Overview

The Aggregate Data Drug Information System (ADDIS) was developed as a prototype to show the feasibility of structured, evidence based, drug benefit-risk decision making through the application of appropriate decision support software. The target audience consists of drug regulators, especially the European Medicines Agency (EMA). The following specific functionality was implemented, driven by a case study in anti-depressants:

- Evidence synthesis using network meta-analysis in a Bayesian (Markov chain Monte Carlo) framework. Our contribution included automated methods for model generation, an implementation in Java, and a graphical user interface that presents the model outputs. Because of the MCMC nature of the models, they are computationally intensive and require convergence assessment. Task scheduling infrastructure was developed to support this. In addition to network meta-analysis, a standard frequentist pair-wise meta-analysis was also implemented, including forest plots.

- Multiple Criteria Decision Analysis (MCDA) methods to assess trade-offs between multiple outcomes. The Stochastic Multi-criteria Acceptability Analysis (SMAA) family of methods were our main focus since they support uncertainty in the measurements as well as incomplete preference information. We also implemented a simpler method derived from the Cost Effectiveness Acceptability Curve (CEAC) approach and support visualization of the benefit and risk criteria based on the qualitative BRAT framework. Benefit-risk analyses can be based on a single trial, or on meta-analyses (one for each outcome).

- A structured database of clinical trials, in which especially the structure of trials is well captured. Measurements can be coupled to specific arms, outcomes, and time points in the trial. Between-trial analysis is enabled by "mapping" the outcomes and interventions in each trial to globally defined entities. The data is sufficiently structured to enable the EMA D80 summary of efficacy table to be filled in automatically, although a few elements are missing.

- Semi-automated import of protocol and results from ClinicalTrials.gov are available. Most data and some of the structure of the trial can be automatically imported. Outcomes and interventions have to be mapped to entities in the ADDIS database by the user. Because ClinicalTrials.gov does not model the time structure of trials, this information has to be created manually.

- With an appropriately filled database, wizards guide the creation of meta-analyses and benefit-risk analyses. Especially the process of creating meta-analyses has become increasingly refined over the development of ADDIS 1.x, allowing either different drugs or different doses of the same drug to be compared to each other, or a combination thereof. Effectively, we found that the definition of a "treatment" for the purposes of meta-analysis needed to be decoupled from the notion of "drug".

ADDIS 1.x is a cross platform desktop application developed in Java. The database is implemented as an XML file with a well-defined XML Schema. Compatibility of XML files between ADDIS releases is supported using XSLT. The graphical user interface has been developed using the Swing framework and jgoodies libraries and is based on the Presentation Model variant of the Model-View-Controller pattern. The network meta-analysis functionality is implemented in a separate component (GeMTC) which also has an independent GUI. The SMAA components were taken from the JSMAA software.

ADDIS 1.x Design Decisions

- A cross-platform Java desktop application.
    - (+) Enabled development on Linux while minimizing worries about deployment on Windows or Mac OS X. Incompatible implementations of the Java standard library did mean that we needed to test on each platform.
    - (+) Having a standalone desktop application meant that we did not need to worry about deployment, infrastructure, or authentication and authorization.
    - (+) It is easy to find developers with experience in Java.
    - (+) A large community exists around Java, and many open source libraries were available that were relevant and of high quality.
    - (+) Tool support for Java is excellent. For example unit testing and continuous integration tools as well as the refactoring and code navigation tools in Eclipse are great for productivity.
    - (-) To maintain cross-platform compatibility, all components needed to run on the JVM. For example, standard software for MCMC could not be reused.
    - (-) Java is not always the best tool for the job. For example, it is often easier to implement statistical analyses in R. The constraints of bundling a desktop application make it difficult to choose the best available technology, leading to re-implementation of existing software.
    - (-) Java development involves a lot of boiler plate and can be more time consuming than using a more dynamic and/or high level language.
    - (-) No control over the execution environment means that the application has to be aimed at the lowest common denominator.
- Database in local XML file.
    - (+) Having the database in a local file means that data ownership, concurrent access, and other multi-user problems did not need to be solved.
    - (+) Data recovery and version control of ADDIS data files is helped by the plain-text nature of XML.
    - (+) Migration from one version of the ADDIS data model to the next is made easy by XSLT.
    - (+) XML files are easy to copy, backup, and share.
    - (+) XML appears to be a good fit for this type of data modeling.
    - (-) Binding the Schema to Java objects is not straightforward. The mismatch between XML's hierarchical structure and Java's object oriented model ensure that all automated tools for this have (big) disadvantages. We ended up with a "converter" between our own Java entities and those generated by the binding library. This converter needed to be updated for every change to the data model. A source of frustration and bugs.
    - (-) Because the data are stored in local XML files, collaborating on a data file is difficult.

There is no straightforward way of merging changes made by several users.

- Presentation Model and test driven development.
    - (+) Our wizards had a lot of logic that was not directly related to a single model. Creating presentation models for these wizards made them much more testable.
    - (-) Fully implementing presentation model is difficult. Often some of the logic ended up in GUI code, either because implementing it purely in the presentation model was much more difficult, or because the GUI framework forced us to do so. This made the code more difficult to understand, although a few big refactoring efforts made a cleaner separation between presentation and GUI.
- Decoupled GeMTC and JSMAA components.
    - (+) The GeMTC and JSMAA components are also successful as stand alone software.
    - (-) Fixing a bug in ADDIS often involved changes in both ADDIS and GeMTC or JSMAA, thus complicating release management. The components were not completely decoupled, also due to a desire to reuse the full stack rather than just interfacing with a single layer.
- The ADDIS data model aims to model what truly happened in the trials, yet forces the user to map certain things to globally defined entities, such as endpoints, indications, drugs.
    - (+) By not tying the data extraction to a specific analysis there is greater flexibility in (re-)analyzing the data.
    - (+) The mapping process is relatively straightforward.
    - (-) The more accurate the data model, the more nuances need to be dealt with in the process of creating an analysis. For example, we allow modeling of the time structure, but it currently can not be taken into account automatically.
    - (-) Mapping is often arbitrary, and different analyses may require a different mapping. For example, one analysis might want to view the IR and ER versions of a drug as the same, while another might want to look at differences between the two. Especially outcomes suffer from this problem. Essentially there are different levels of description that may all be appropriate for some purpose or other, but we force the selection of a single description.
    - (-) There may not always be a clear candidate for mapping. One may end up with a very large collection of entities that are each used in only one study. This is especially relevant for import from ClinicalTrials.gov as it is undesirable to throw away data just because it does not map to anything of (current) interest.
- ADDIS helps the user identify studies in the database and use them in analyses by going through a number of fixed steps in a wizard. These depend greatly on the studies having accurate mapping to the globally defined entities.
    - (+) The process of creating an analysis is relatively straightforward, consisting of just a fixed number of choices.
    - (+) Wizards are relatively easy to implement.
    - (-) Flexibility to go outside the standard process is very limited, consisting only of the ability

to manually exclude certain interventions or studies.

- (-) Any additional flexibility (like categorizing by dose) introduces additional complexity in the wizard. We partially solved this by providing good default values, but still there are additional steps to go through which may not seem to make a lot of sense to the average user.

- (-) The wizards hide some important logic, for example how we determine when a study matches a given treatment definition.

- (-) Important choices made during the creation of an analysis are not always stored, nor can a rationale be provided. The wizards' design was driven more by technical considerations than user requirements.

ADDIS 2.x Directions

- ADDIS needs to become more community-oriented. It should be easy (or even the default) to share data extractions between users. ADDIS should enable the research community to collaboratively build a large database of structured clinical trials data for flexible (re-)use.

- ADDIS needs to become more service oriented and components should be decoupled further. This will enable us to use the right tool for each sub-problem without being tied to the Java platform. It should also allow third parties to integrate ADDIS or any of the subsystems in their applications.

- Users should be able to clearly define their projects, their motivations and their choices in building up analyses. The system should keep a log of user actions and choices so that the work flow can be retraced.

- ADDIS should support multiple levels of description for key concepts such as outcomes and interventions. Otherwise identifying data for analysis can not be made efficient in general.

- It should be easier to define different work flows for defining analyses. Different target audiences will perform different kinds of analyses. A single wizard will not suit all of their needs.

- We need to develop ways of dealing with the fuzzy reality of this type of data. We can no longer assume that everything has been mapped, or has been mapped accurately. Moreover, mappings can change over time.

- We need to make better use of existing ontologies and terminologies.