1. Define types using RDF-Schema

The application will let the users define their own data types using RDF-Schema. For example:

Project { people involved, files, calendar events, notes }

Trip { pictures, locations, people }

Types can be complex (containing subelements) or simple (string, date).

Instances of types can contain more or less elements than defined in a type. Some elements may be marked as required, other may be marked as optional.

2. Associate presentation logic with types

Types may have (different kinds of) presentation logic associated with them. For example, an *image* type might have some associated code that renders the image on the screen. A *soundfile* type might have associated logic to play that file. These are low-level presentation functions that will be implemented by a programmer/power user.

The low-level presentation logic can be combined to form *views*. A view is a form which onto which the user can drag type elements. Each type element may have several different presentation options (low-level or other views) – the user can choose the way that an element is going to be presented.

Questions:

- Filtering associated information – how to let the user define filters in a simple and intuitive way?

3. Create and annotate type instances

The user will be able to create a new instance of a particular type and fill it with data directly in the view. Text fields will be modifiable. One will be also able to drag files from a folder into an image list box to add them (this will create a link between the container and an image – see *Linking information* below).

Any information contained in the system can be annotated. For example, one can enter the birthdate of a friend and annotate it with a comment like “This date may be wrong.” Furthermore, the comment itself could be annotated as a TODO event: “Verify the birthdate with someone.”

4. Linking information

Information can be linked together. For example, one may drag an object of type Person onto an object of type Picture, which will establish a default relation between these two objects (could be just “is-related-to” or something like “is-depicted-in-picture”). One could also drag a Person onto an object of type Meeting – this could popup a box asking for the type of relation between these two objects (“meeting chairman”, “required attendee”, etc.)

5. Equivalence

An equivalence relation can be established between two objects (owl:sameAs). This can be used to link objects that represent the same thing. For example, a user might have created a Person object describing John Doe. However, John Doe also has a Facebook account, which is represented by an object of another type. Establishing an equivalence relationship will basically merge these two objects together for presentation purposes.

Note:

- Maybe it should also be possible to establish an equivalence relationship between classes (owl:equivalentClass). This could be useful to state that Person::name is the same as FacebookAccount::name.

6. Sharing

The user will be able to publish selected information via a webpage for others to see. The information will be presented using the user's defined views.