with a lot of locality of reference might well be preferable. Not using a database for the data items would mean that transactional semantics on the operations on the store would have to be implemented manually. A proposed solution for this was the extension of the principles employed by the maildir standard, which essentially allows lock-free ACID access by relying on atomic rename operations on the filesystem.† For the first pass implementation, it was decided that the database would store both data and metadata, with the intent of optimizing this at a later point, when the requirements of searching in particular would be more well defined.

Given that the Akonadi store serves as the authoritative cache for the user's personal information, it can effectively offer and enforce advanced cache lifetime management. Through the concept of cache policies, very fine-grained control over which items are retained and for how long can be exposed to users to allow a wide variety of usage patterns. On one end of the spectrum—for example, on an embedded device with a bad link and very limited storage—it might make sense to develop a policy to never store anything but header information, which is pretty lightweight, and only download a full item when it needs to be displayed, but to keep already downloaded items in RAM until the connection is severed or power to the memory is shut down. A laptop, which often has no or only unreliable connectivity but a lot more disk space available, might proactively cache as much as possible on disk to enable working offline productively and only purge some folders at regular intervals of days, weeks, or even months. On the other hand, on a desktop workstation, with an always-on, broadband Internet connection or local area network access to the groupware server, fast online access to the data on the server can be assumed. This means that unless the user wants to keep a local copy for reasons of backup or to save network bandwidth, the caching can be more passive, perhaps only retaining already downloaded attachments for local indexing and referencing. These cache policies can be set on a per-folder, per-account, or backend basis and are enforced by a component running in its own thread in the server with lower priority, regularly inspecting the database for data that can be purged according to all policies applicable to it.

Among the major missing puzzle pieces that were identified in the architecture at the 2007 meeting was how to approach searching and semantic linking. The KDE 4 platform was gaining powerful solutions for pervasive indexing, rich metadata handling, and semantic webs with the *Strigi* and *Nepomuk* projects, which could yield very interesting possibilities when integrated with Akonadi. It was unclear whether a component feeding data into Strigi for full indexing could be implemented as an agent, a separate process operating on the notifications from the core, or would need to be integrated into the server application itself for performance reasons. Since at least the full text index information would be stored outside of Akonadi, a related question was how search queries would be split up, how and where results from Strigi and Akonadi itself would be aggregated, and how queries could be passed through to backend server systems capable of online searching, such as an LDAP server. Similarly, the strategy for how to divide responsibilities with Nepomuk—for example, whether tagging should be entirely

† *http://pim.kde.org/development/meetings/osnabrueck4/icaldir.php*