board drawing from that discussion in Figure 12-1 shows, there would be a storage (persistence) layer at the bottom, with logic to access it on top, a transport and access protocol layer above that, on up to application space.

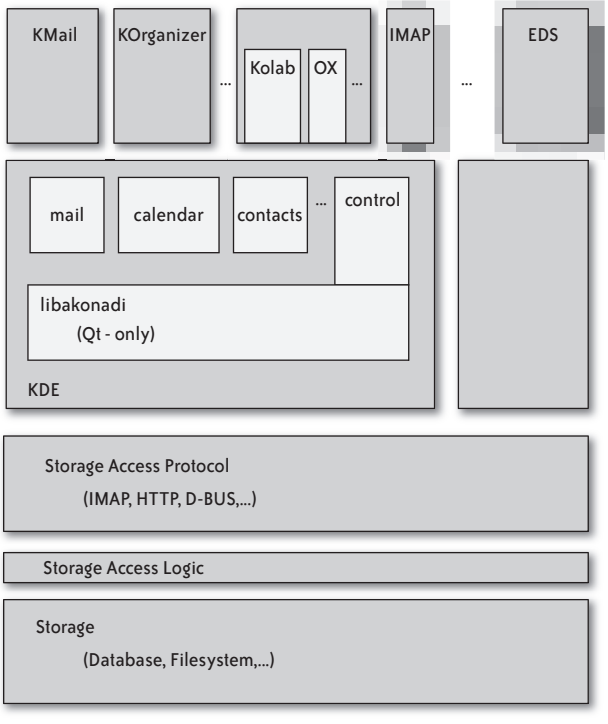

FIGURE 12-1. Initial draft of the layers in the Akonadi framework

While debating what the API for the applications' access to the store would look like (as opposed to that used by agent or resources delivering into the system), some were suspicious that there would need to be only one access API, used by all entities interacting with the store, whether their primary focus was providing data or working with the data from a user's point of view. It quickly emerged that that option is indeed preferable, as it makes things simpler and more symmetric. Any operation by an agent on the data triggers notifications of the changes, which are picked up by all other agents monitoring that part of the system. Whatever the resources need in addition to the needs of the applications—for example, the ability to deliver a lot of data into the store without generating a storm of notifications—is generic enough and useful enough to be worthwhile in one unified API. Keeping the API simple for both application access and resource needs is required anyhow, in order to make it realistically possible for third parties to provide additional groupware server backends and to get application developers to embrace Akonadi. Any necessary special cases for performance or error recovery