- There would be only a very limited amount of data. After all, how many contacts would users typically have to manage, and how many appointments or tasks? The assumption was that this would be in the order of magnitude of "a few hundred."
- That access would be required only by C++ and Qt libraries and KDE applications.
- That the various backend implementations would work "online," accessing the data stored on a server without duplicating a lot of that data locally.
- That read and write access to the data would be synchronous and fast enough not to block the caller for a noticeable length of time in the user interface.

There was little disagreement among those present at the 2005 meeting that the requirements imposed by real-world usage scenarios of the current user base, and even more so the probable requirements of future use cases, could not be met by the current design of the three major subsystems. The ever-increasing amounts of data, the need for concurrent access of multiple clients, and more complicated error scenarios leading to a more pressing need for robust, reliable, transactional storage layers cleanly separated from the user interface were clearly apparent. The use of the KDEPIM libraries on mobile devices, transferring data over low-bandwidth, high-latency, and often unreliable links, and the ability to access the user's data not just from within the applications traditionally dealing with such data, but pervasively throughout the desktop were identified as desirable. This would include the need to provide access via other mechanisms than C++ and Qt, such as scripting languages, interprocess communication, and possibly using web and grid service technologies.

Although those high-level issues and goals were largely undisputed, the pain of the individual teams was much more concrete and had a different intensity for everyone, which led to disagreement about how to solve the immediate challenges. One such issue was the fact that in order to retrieve information about a contact in the address book, the whole address book needed to be loaded into memory, which could be slow and take up a lot of memory if the address book is big and contains many photos and other attachments. Since access was by way of a library, with a singleton address book instance per initialization of the library and thus per process, a normal KDE desktop running the email, address book, and calendaring applications along with helpers such as the appointment reminder daemon could easily have the address book in memory four or more times.

To remedy the immediate problem of multiple in-memory instances of the address book, the maintainer of that application proposed a client/server-based approach. In a nutshell, there would be only one process holding the actual data in memory. After loading it from disk once, all access to the data would be via IPC mechanisms, notably DCOP, KDE's remote procedure call infrastructure at the time. This would also isolate the communication with contact data backends (such as groupware servers) in one place, another concern with the old architecture. Lengthy discussion revealed, though, that while the memory footprint issue might be solved by this approach, several major problems would remain. Most notably, locking, conflict resolution, and change notification would still need to be implemented on top of the server