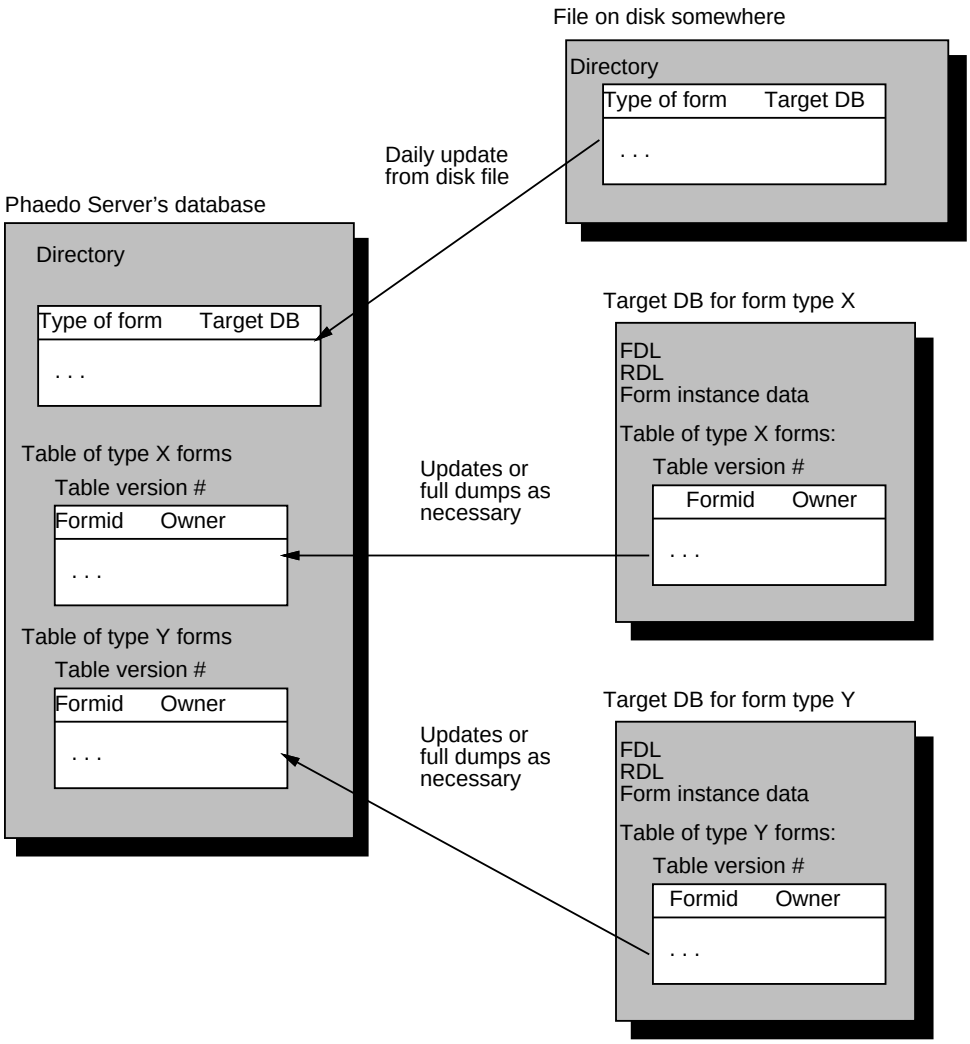
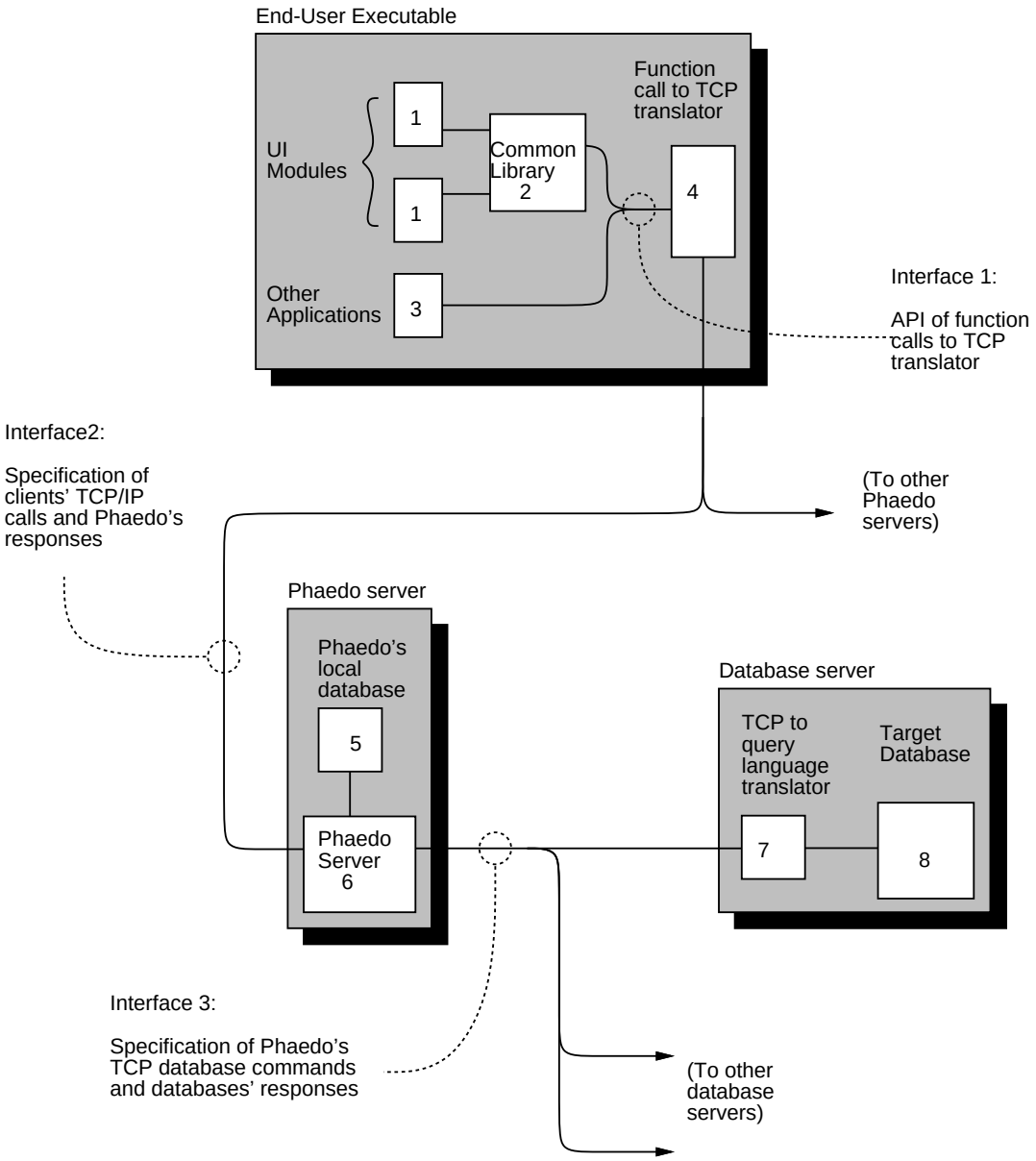
Appendix B
Distributed Database for Phaedo



Appendix A
Major Components of Phaedo



Distribution of Information

There are two types of information which must be known to the Phaedo servers. First, the slowly-changing table of form types and their locations. Second, the much more rapidly-changing table of form id's and the owners and types of these forms. (Appendix B diagrams these tables.) Each Phaedo server will have a local copy of the information, and a well-specified method of ensuring that its own information is correct, when it wishes to do so. All servers will be peers, and they will not attempt to enforce a globally consistent database amongst themselves.

Table of form types

This table contains a list of all the types of forms available on Phaedo, along with a pointer to the server which is responsible for each type of form. Each Phaedo server will have its own copy of this table, and will request updates periodically from a central location. Since this information changes infrequently, disconnection of the central location for hours or even a day or two will not be catastrophic, since the servers can continue to run sanely while it is down. The only consequence will be that newly-created types of form will not be available.

Tables of forms and owners

The second type of information is a collection of tables, one for each type of form, listing individual forms and their owners. This information will be changing rapidly, and all Phaedo servers must be informed of any changes that another server makes.

Each database server will maintain the authoritative version of the tables for the type or types of forms it is responsible for. The version number of each of these tables will also be stored on the database server. Whenever a Phaedo server makes a request which changes one of these tables, such as creating a new form or reassigning an existing one, the database server will update the authoritative table accordingly and increment the version number.

All Phaedo servers will maintain their own copies of these tables, along with the version numbers for their copies. Before a server takes any action which depends on the values in these tables, it will check its own table version number against the target database's version number, and request an update if its copy is obsolete. If a target database goes down, only the information for the types of forms stored on that database will be unavailable; other types of forms can still be used normally.

Conclusion

The Phaedo system architecture is designed with two chief goals: extendibility and reliability. Extendibility is provided by designing the system as a set of functional modules, with well-defined interfaces between the modules. Reliability is provided by decentralizing the system's state information, so that the failure of a single module will not cripple other, unrelated, modules. Furthermore, the ability to support multiple servers providing the same functionality, so that one can function when another dies, adds to the system's reliability.

We believe that this design will provide the skeleton for a enduring and usable system.

user, and not a mere pretender. Similarly, the Phaedo server authenticates itself to the database servers it is speaking with, so the databases know that they can trust the information and requests coming in from Phaedo.

There are tradeoffs between doing security at the Phaedo level and at the individual database level. For example, some security decisions are made at a level which an individual database may not be aware of, e.g., only the current owner of a form may edit it, or a field may not be filled in unless a prerequisite field is also filled in. These decisions must be made at the Phaedo level. On the other hand, the individual database may insist on making some authorization decisions itself, such as determining who is allowed to delete completed forms from the system. For this reason, a method must be provided for the database and Phaedo to negotiate who is responsible for security on the level of individual fields on a form.

Phaedo Server Architecture

The Phaedo server is designed to be replicated. The primary reason for this is to avoid a single point of failure; if there are multiple Phaedo servers running, and one goes down, the surviving server[s] will continue to run. Also, the interactive nature of electronic forms demands speedy response time. A single server could act as a bottleneck, making use of Phaedo's forms very frustrating.

Connection from end-user application to Phaedo server

Whenever a user application starts up, it will locate a Phaedo server and establish a connection to it. As with the current Zephyr implementation, this may be done either through querying Moira, or searching a flat file.

A Phaedo server will maintain a TCP connection to a end-user application for as long as the applications wishes to maintain the connection. The server's connection to the target database may be transient, because the communications between the application and the Phaedo server are likely to be interactive, while the communications betwen the Phaedo server and the target database are much more batch-like. Note, though, that this does not preclude an application from operating in a batch-like fashion by opening a connection to the Phaedo server, getting the desired data, and then closing the connection while the user does work, with the intention of reestablishing the connection once the user is done.

System Robustness

The single most important demand on the distribution of the Phaedo servers and the information stored within the Phaedo system is that there be no single point of failure, and that the system must continue to work, although probably with reduced functionality, if any single component dies. This means that there can be no single repository for Phaedo state information.

Data will be distributed throughout the system. If a one customer's target database goes down, only forms using that database are disabled, but other portions of Phaedo continue to function normally. This is very similar to Discuss meetings, where the failure of one host only disables the meetings maintained on that host.

Application Programmers' Interface

The first is the Application Programmers' Interface, or API. This allows an independent programmer to write applications using Phaedo. It is a library of routines with which the programmer can link his programs, and which trigger network requests to the Phaedo server.

A programmer will be able to build new applications which use Phaedo simply by compiling with this library. These applications may be new user interfaces, autonomous processes which convert incoming mail into new forms, or any of a number of other applications. So long as they use the Phaedo library properly, there will be no difference between these applications' effects on the Phaedo system and the effects of the pre-supplied interfaces.

Typical requests include:

- Authenticate user to server
- Ask for a new blank form
- Check out an existing form
- Check back in a form you previously checked out

Bringing Phaedo to a new platform (Macintosh, DOS, etc) will begin with porting this library. The various platform-specific end-user applications can be compiled with this library. The user's application will send messages to the Phaedo server by calling these routines, and the Phaedo server sends information to the application by filling in structures or triggering pre-registered callbacks.

Application-to-Phaedo Interface

This interface definition specifies the actual format of the packets which travel on the network between the end-user application and the Phaedo server. It is not expected that these will be directly human-generated; instead, they will be generated by the library described above. In general, there will be a one-to-one correspondence between a call to a library function and an outgoing TCP message to the server. Library routines will wait for a TCP reply from the server, if appropriate, and return the value to the calling application.

Phaedo-to-Database Interface

The Phaedo-to-Database interface is the protocol by which a Phaedo server communicates with the various database servers. Like the second interface, it is a definition of the format of TCP packet going over the network. When a database server receives a packet, it translates the generic request into the corrosponding database-specific language (e.g., SQL), runs the request on its database, and returns the result, if any. This level is more specific and low-level than the other interface levels. Examples are setting a field of a given record to a specified value, or simply querying the value of a field.

System Security

Communications are authenticated at various points within the system. A user at an application authenticates him/herself to the Phaedo server via Kerberos; thus Phaedo knows that it is talking to the real

Overview

This document gives an overview of a proposed system architecture for Phaedo.[1] This system architecture was designed with two key requirements: First, that the architecture be extendable, so that new functionality can be added to the system without disrupting day-to-day use. Second, that there be no single point of failure which can cripple the entire system. To meet these requirements, the Phaedo system is a collection of intercommunicating modules, where new modules can be added as desired, and the failure of any single module only removes the functionality provided by that module. Modules may be replicated, in order to provide more robustness.

System Components

There are three major component types in the Phaedo system, as diagrammed in Appendix A. These are end-user applications, Phaedo servers, and database servers.

It is expected that there will be more than one of each of these components. For example, different database servers would maintain data for different kinds of forms, and different end-user applications would run on different platforms, such as Macintoshes and Athena workstations.

- End-user application. An end-user application is a program which the end-user uses to interact with the system, e.g. popping up blank forms, filling them out, and forwarding them to the next owner of the form. Multiple end-user clients can provide different mechanisms for interacting with Phaedo's forms. Some clients may simply be different user interfaces providing identical functionality (for example, character-based and X-based interfaces) while other clients may provide unique services, such as a process which automatically converts Discuss transactions into Phaedo forms without explicitly being run by a human.
- Database Server. A database server provides access to the actual information contained on the forms, and stores the information. Each database server provides access to a different database. All forms of a single type will live on the same database, and a single database may contain many different types of forms. The various databases may be implemented completely differently, e.g., some may be Ingres databases, others simple flat files, and others Discuss meetings. All, however, will have the same functionality as far as Phaedo is concerned. Phaedo will maintain records of which database servers are responsible for which types of forms.
- Phaedo Server. A Phaedo server acts as a clearinghouse for forwarding the users' requests to the appropriate database servers, and returning the responses. There may be any number of Phaedo servers, all of which are peers and clones of each other. It is expected that, if there is more that one Phaedo server, the different servers will run on different machines, to help reduce bottlenecks in the system, and provide better reliability in case one of the server machines fails.

Interface Definitions

There are three interface definitions within Phaedo. These interfaces specify the way in which different modules communicate, and make it possible to indefinitely expand the Phaedo system by adding new modules which use the appropriate interface.

[1] More information on Phaedo can be found in the document "Requirements of Phaedo, an electronic form routing system," by Andy Oakland, sao@athena.mit.edu

Draft of a System Architecture for Phaedo

Andy Oakland

September 15, 1992