WHITE PAPER

Open End™ Technical Overview

December 2006

1. Open End overviews and goals

Open End™ product is a framework for workflow-intensive applications. Open End eases development of applications to manage projects, resources, or relationships with suppliers, customers or employees. Another field in which Open End enhances developer productivity are applications that help track and handle bugs or helpdesk-cases.

Application frameworks enable rapid development of custom applications. A framework, just like a library, supplies a substantial body of helpful code. A framework, in comparison to a library, enhances productivity by letting application programmers exploit the design and architectural expertise embodied in the framework's structure.

The Open End framework emphasizes collaboration through the storage and sharing of information. Open End functionality includes real-time updates and event handling. Open End is also designed for flexibility, to integrate with other systems already deployed in the enterprise.

Open End embodies principles of Library Science and Information Science, and also principles derived from experience in management and entrepreneurship. As a result, Open End empowers and encourages "best of breed" approaches to collaboration in knowledge-intensive workplaces.

2. Open End layered architecture

To deliver these benefits, Open End relies on a flexible multi-layer architecture, implemented in the Python programming language and running portably across platforms. Linux, Solaris and Windows are supported at this time, but adding other platforms is quite feasible, thanks to the cross-platform nature of the architecture and technologies Open End uses.

Open End' multi-layer architecture can be summarized as follows:

0. "database": any enterprise-level RDBMS (Relational Database Management System)
1. "IFC™": a special OODB (Object-Oriented Database) built on top of the layer-0 RDBMS
2. "BL": the middleware that interfaces to the layer-1 IFC
3. "BLM"s: modules loaded by the layer-2 BL to define business objects and implement business rules
4. "front-end" or "clients": GUIs, report-writers, and other such tools that interface to the layer-2 and -3 BL and BLM

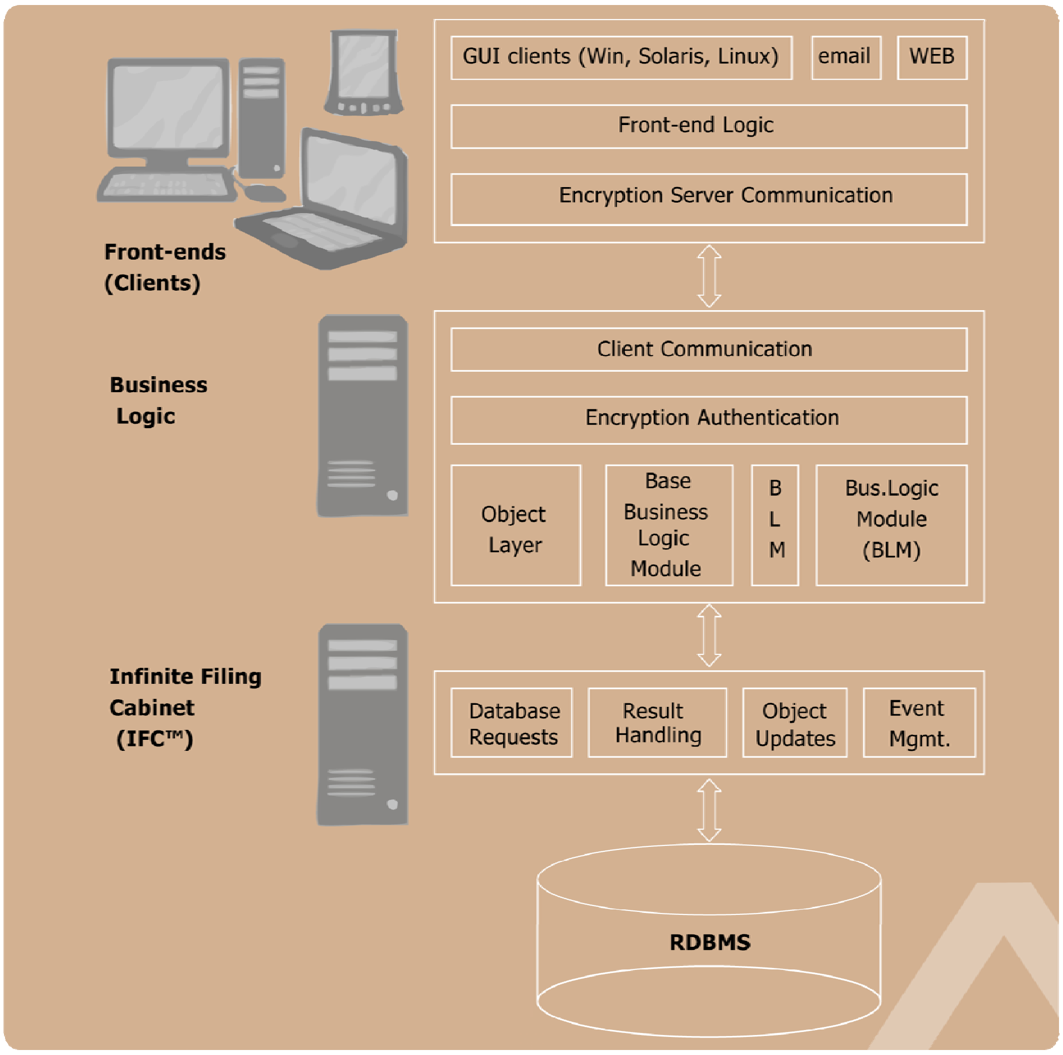

Figure 1: Open End layered architecture

3. Open End communication infrastructure

Open End' layers 1, 2 and 4 communicate securely among themselves with the SSL (Secure Sockets Layer) protocol, as implemented by the OpenSSL library. Python's interface to OpenSSL, known as PyOpenSSL, was developed by Strakt and released under the GNU Lesser General Public License (LGPL). [Layer 3 runs within the same process as Layer 2, and therefore intrinsically communicates with it securely and rapidly by typical intra-process mechanisms such as function calls and callbacks].

SSL runs on top of TCP/IP. Therefore, the processes that run the programs composing Open End' layers 1, 2 and 4 can (and, in fact, generally do) run on multiple hosts in the same LAN. This "naturally distributed" architecture eases the deployment of scalable systems, with optional load sharing and redundancy.

Secure communication among Open End layers 1, 2 and 4 is also quite feasible over WANs, and indeed even over the Internet. However, performance issues related to network traffic (bandwidth and delay) need to be carefully assessed in order to make such deployments practical. As a rule of thumb, most front-end (layer 4) programs may often be successfully deployed at locations remote from upstream layers. Deployment of server-layer programs at locations remote from each other, on the other hand, normally requires reliable, high-speed connections.

Open End' communication with other systems employs various appropriate protocols and formats. Open End uses SQL (and modules compliant with Python's DB/API specifications) to interoperate with relational database systems (including the "layer 0" database). SMTP and RFC 2822 formats (as implemented by Python libraries and extensions) let Open End interoperate with email systems. Other standard formats and protocols, such as HTTP, HTML, and XML, similarly allow Open End to interoperate with Web servers and browsers, Web services, and other applications yet.

All of the communications-related components within Open End use a highly scalable and responsive asynchronous architecture. This architecture employs both event-driven processing and multi-threading, and is implemented on top of the "Twisted Matrix" Python communications framework.

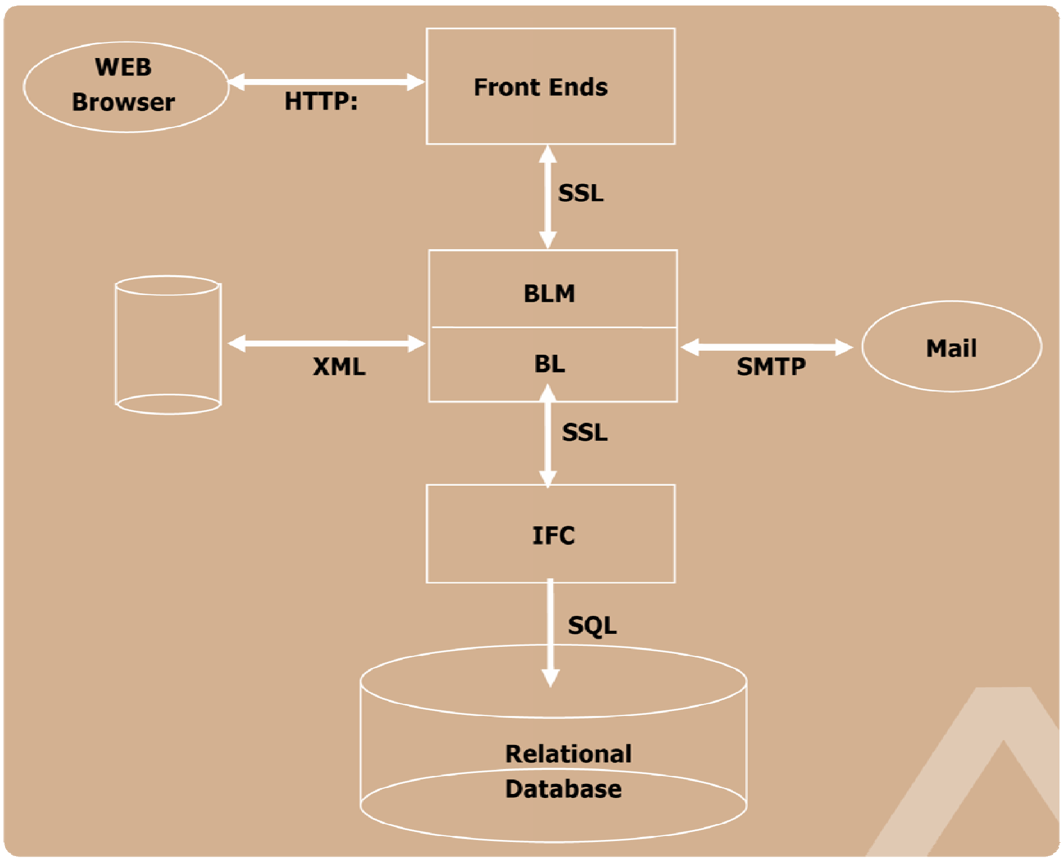

Figure 2: Open End communications

4. Open End layer 0: the database

Open End "Layer 0", outside of Open End proper, can be any enterprise-level RDBMS (Relational Database Management System). The RDBMS's functionality must include such features as transactions, views, outer joins, and the like. Layer 0 is also informally known as the "database layer".

The development of Open End has taken place using PostgreSQL (with some specific attention also given to Oracle), without exploiting special, non-portable features found in

PostgreSQL only. Special features and quirks specific to each given RDBMS can be enOpen Endulated into a thin, well specified "portability layer". As a consequence, future addition of other RDBMS engines of comparable functionality, if and when AB Strakt's partners and customers require it, will be feasible.

However, please note that Open End defines its own DB schema. Integrating Open End with other existing applications that use the same DB engine, but with their own schemas, is a task similar to that of other integrations between Open End and existing applications. There is no way to make Open End just use a pre-existing relational DB schema. Therefore, normally, no special advantage is gained by having Open End Layer 0 share a particular DB engine with other applications. Many of the beneficial system characteristics we shall examine in the following depend on the crucial design choice of having Open End define and use its own DB schema.

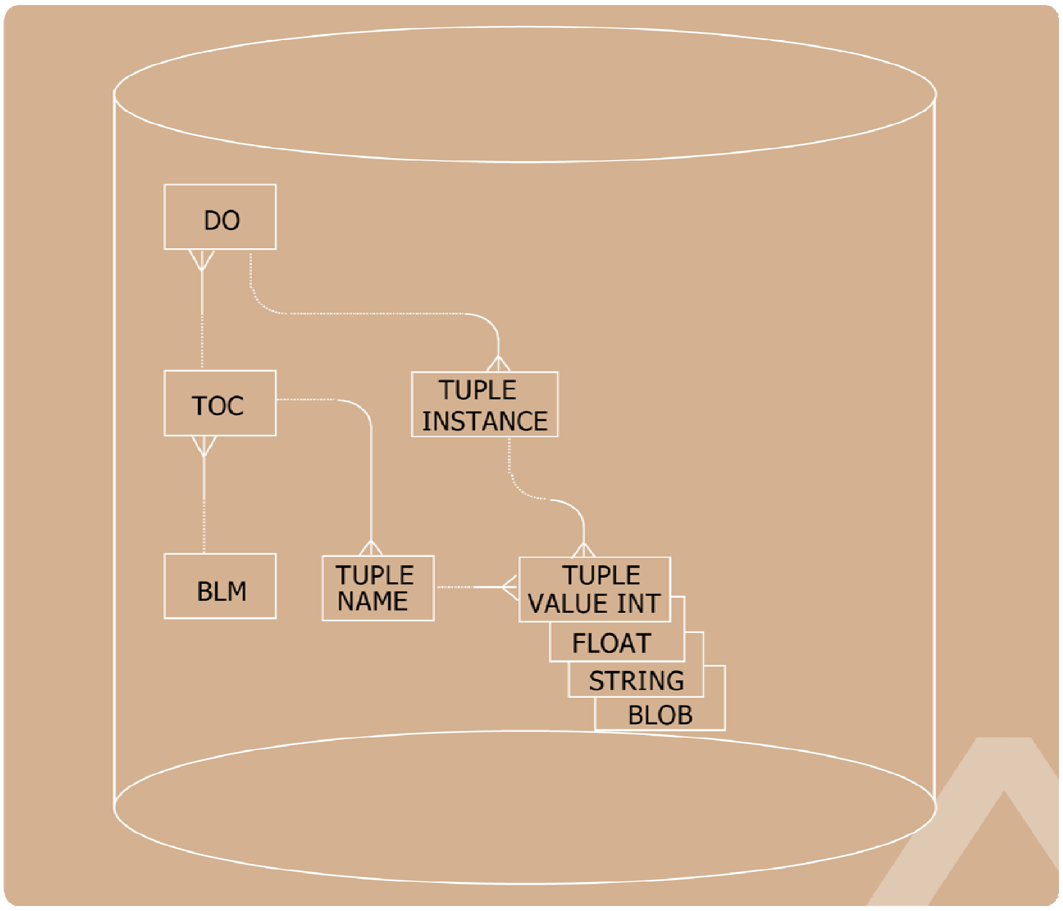

Figure 3: Open End Database schema

5. Open End layer 1: the Infinite Filing Cabinet (IFC)

Open End Layer 1 builds, on top of the relational DB in Layer 0, a real-time, transactional Object-Oriented Database (OODB).

The "Infinite" adjective in Layer 1's name refers to the fact that, by design, the IFC _never removes information_. Data that is logically "deleted" or "updated" is in fact simply marked up as "deprecated" so that by default it does not show up in normal searches.

This approach intrinsically provides a _perfect audit trail_ for all transactions performed in the IFC. Furthermore, the approach ensures that a wealth of data remains permanently available for any future "data mining" requirements. Applications built with the Open End framework need no special precautions to ensure auditability, nor to support "data mining" operations.

The ability to physically delete and obliterate data can be provided separately, through a dedicated "administrative console" interface that is not normally accessible to applications. This functionality is necessary in the rare case of applications for which legal requirements mandate physical, irrecoverable deletion of some data. The same interface, in theory, could also be used to recover storage space by forgetting some history. However, that is not the design intent of Open End. With the predicted rate of cost decrease and capacity increase of storage solutions, we judge it most likely that the business value provided by keeping all historical data around will far outstrip the modest per-bit costs of storage subsystems.

The OO architecture of the IFC separates _queries_, which yield the sets of objects meeting certain conditions, from _requests_, which yield the values of specified attributes of a given set of objects. Queries and requests can be employed in the same way as queries are used in traditional database operations. Such "traditional" queries and requests (known in Open End as _transient_ ones) obtain the sets of objects, or attribute values, "captured" at the time the query or request is performed. In addition, as befits the real-time enterprise capabilities of the IFC, queries and requests can be flagged as _subscriptions_. In this case, the IFC sends information about updates to the relevant sets (additions, removals, changes in attribute values), each time the database changes in ways that modify the sets.

This crucial "subscription" ability ensures that all subsystems downstream of the IFC can, if they wish, display constantly-updated information, without any need for continuously "polling" the IFC with repeated requests and queries.

The real-time enterprise abilities of the IFC also include "IFC events". Events are similar to the "triggers" and "stored procedures" popular in RDBMSs. However, IFC events can be triggered or masked by the passage of time, as well as by conditions related to the data held in the database. Each IFC event alerts Open End layer 2, the BL, so that the BL can perform appropriate actions specific to the event.

IFC events may be "normal", "timed", or "rule-based". Each kind of IFC event is represented by a corresponding kind of event-object. A _normal_ event-object sends an alert when certain atributes change to given values, or away from given values, or from given "entry" values to given "exit" values. A _timed_ event-object sends the alert if, and only if, specified attributes have certain given values at a given instant of time. A _rule-based_ event-object is similar to a normal event-object, but, rather than applying to specific existing objects, it applies to all newly created objects that satisfy specified rules.

6. Open End Layer 2: the Business Logic (BL)

Open End Layer 2 works on top of the OODB capabilities provided by Layer 1. Layer 2 is known as the "Business Logic" layer (BL) because it builds full-fledged "business objects" that reflect the needs of specific applications and respect the "business rules" of those applications.

All "front-end" components (Layer 4) interact with the BL, never directly with the IFC. Therefore, front-end components never have to deal with lower-level, implementation-related concepts: front-ends can always deal with application-oriented, higher-level concepts and terms.

Vice versa, the IFC does not have to deal with security issues, besides the obvious one that only a properly validated BL is allowed to connect. The BL handles all issues related to granting various access privileges to different users. If an application's needs require it, such privileges may be defined with very high (detailed) "granularity". In the future, such granularity will get down to specifying, for example, which users or groups of users can even _see_ that a certain attribute exists at all in a given class of objects (apart from being able to get the _value_ of that attribute for certain objects).

The BL has real-time enterprise capabilities comparable to that of the IFC.

In the current version of Open End, only one BL process connects to the IFC. However, Open End is designed to allow multiple BLs to connect to the same IFC. This functionality will enhance scalability in future versions of Open End, and the system architecture allows for such BL multiplicity.

When multiple BL connect to the same IFC, they will be able to exploit the IFC's real-time enterprise capabilities to enhance cooperation. For example, a BL could cache queries and requests that are often executed by components further downstream. In such a case, the BL will be able to efficiently assure that the cache is always up-to-date, by exploiting the IFC's "subscription" mechanisms. Subscription-update messages will then be automatically triggered by all relevant changes to the database, including updates performed on the DB by other BLs simultaneously connected. Such messages will in turn allow the BL to update or flush the cache as and when appropriate.

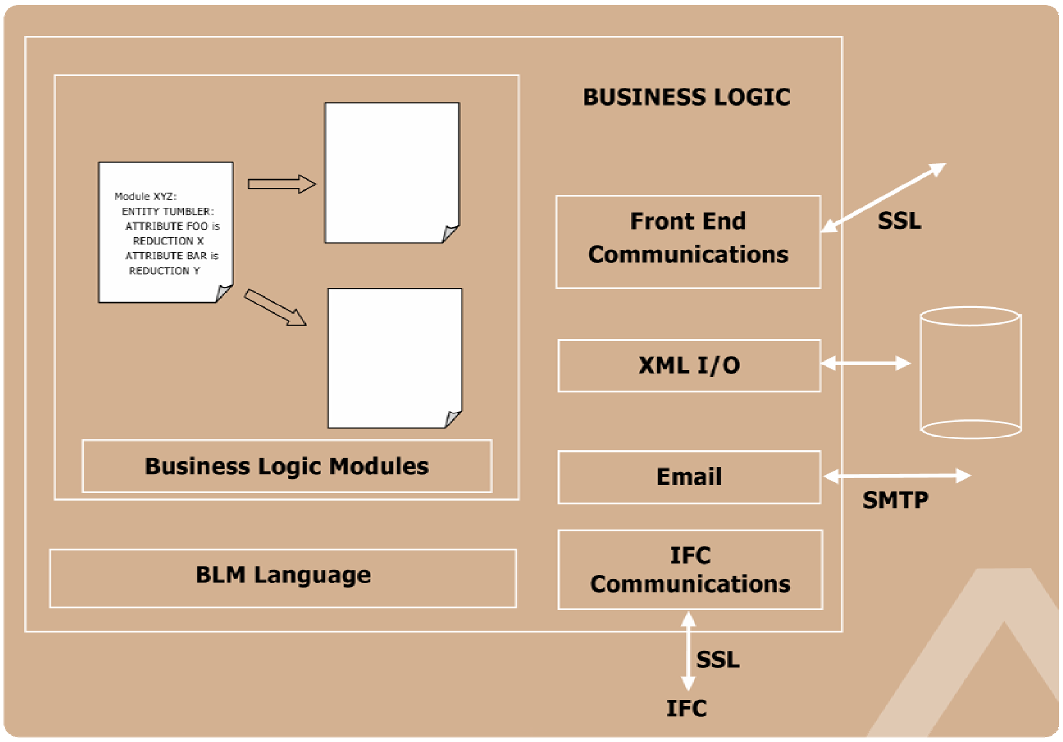

Figure 4: Business Logic Layer

7. Open End Layer 3: Business Logic Modules (BLMs)

The operation of the BL subsystem is highly modular. On top of the BL infrastructure proper, which constitutes Layer 2, a further logical layer runs (as part of the same process) any number of application specific business-logic modules. The BL loads (from the database, via the IFC) the set of "Business Logic Modules" (BLMs) that define the business objects and business rules of one or more applications.

BLMs are coded in a simple but powerful "declarative" language, syntactically similar to Python. The "BLM language" lets the application programmer express all of the fundamental properties of the application, resulting from Entity-Relationship analysis. The BLM language's first-class concepts include the Entities involved in an application, the Attributes of each Entity, and the (binary) Relations that hold between Entities.

The BLM language lets the application programmer declare the most important characteristics of these high-level objects. For example, in the case of a Relation, the programmer declares the "N-Arity" of each "end" of the Relation, that is, whether it is one-to-one, one-to-many, or even "optional" for that end (an N-arity of "zero or one" on that end of the Relation). The BLM language itself allows direct declaration of many typical results of analysis, such as the Types of Attributes, and a variety of Constraints that Attributes must respect (one simple kind of Business Rules). Entities can use (single) inheritance, and can be organized across multiple Modules and code Libraries, with multiple-leveled structured ("package-like") names.

The "procedural" aspects of business objects and rules are expressed by embedding Python source code in the BLM sources. Such embedded code is executed either when requested by downstream subsystems, or when certain conditions occur. For example, some attributes of an entity can be specified as being "computed". The value of such an attribute is not actually stored in the database; rather, embedded Python code runs, and computes the attribute's value on the fly, on the basis of other data, each time the value is requested.

Open End operations include full transactional "ACID" guarantees (Atomicity, Consistency, Isolation, Durability), and internally operate in highly-asynchronous, multi-programmed ways. However, the BLM Language supplies a library that application programmers can use to structure their code in ordinary, sequential ways, while transparently supporting such multiprogramming and ACID guarantees.

Each BLM can be associated with sets of descriptive data, "decorating" each Entity, Relation, Attribute, and so on, with presentation-related meta-information. The Open End framework comes with general-purpose front-ends ("clients"). The "presentation guidelines" associated with each BLM let the application programmer easily customize these clients to properly display the data relevant to each given Open End-based application. Since several such customizable clients exist, each BLM can be associated with several parallel sets of "presentation data", each tagged with a different structured name.

8. Open End Layer 4: the front-end (AKA "clients")

Programs in Layer 4 of Open End interact with the Business Logic (and, indirectly, with the BLMs running in the BL). Such front-end programs also interact with users, directly or through yet other "layers" located even further downstream. For example, typical examples of Layer 4 programs would be Web servers, which could feed "level 5" Web browsers which present data to users. Layer 4 Web servers could also supply Web services (for example through the XML-RPC protocol) which in turn are consumed by further middleware-layer programs yet.

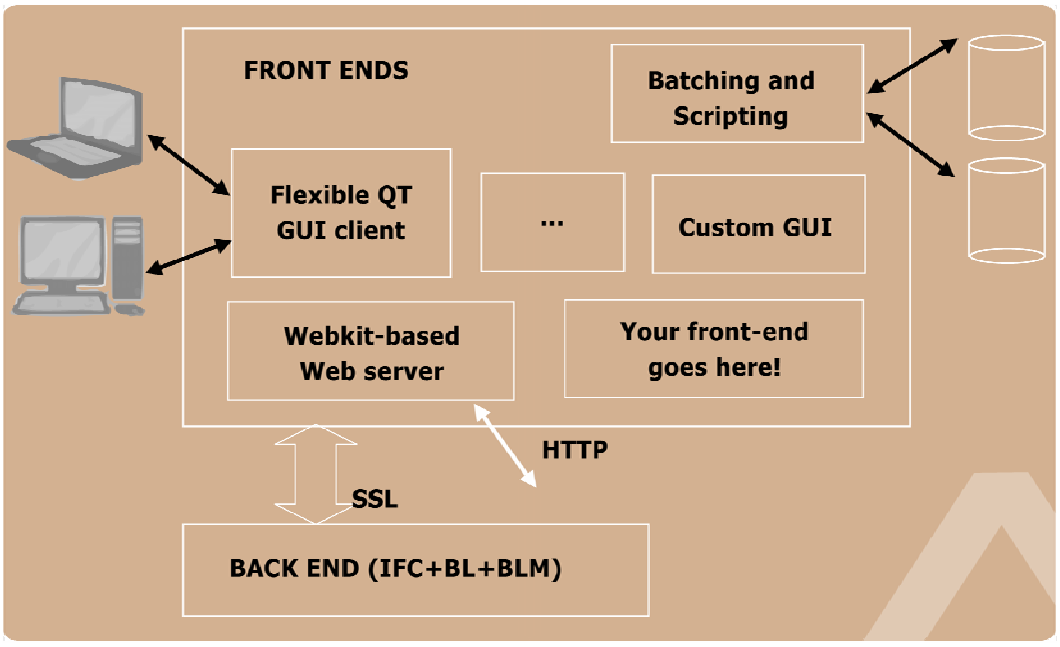

Figure 5: Front ends

8.1 The Open End General GUI Client

Most needs for direct GUI interaction with Open End-based applications are met by one "General GUI Client" program, customized through the presentation data it obtains from the BL. The General GUI Client is coded in Python and uses the popular Qt cross-platform GUI library. This client can run on all platforms supporting these two highly portable technologies, including Windows, Linux and Solaris.

As the primary User Interface to all Open End-based applications, the General GUI Client offers a rich slate of functionality. Such functionality includes the ability to customize presentation choices for a specific user, for a certain set of workstations, or for a whole site (i.e., all Open End usage in an installation). Other examples of advanced General GUI Client functionality include macros, local printing, and real-time operation.

Not yet supported in the General GUI Client, but slated for future implementation, is the capability for "detached" operation.

Detached operation will let machines with only occasional network access, such as laptop computers, keep running Open End-based applications locally during times without network access. Data edits and creations performed during such time periods will be "batched" locally, and sent to the back-end as a "burst" when connection is re-established. Layers 2 and 3 (the BL and BLMs) will ensure that such asynchronous updates do not violate the "ACID" properties, by pointing out any conflicts that need to be resolved before updates can take place.

8.2 The Open End Web-server front-end

The Open End General Webserver is another Open End front-end. It runs on top of the popular, crossplatform Apache Web Server (and potentially on other brands of Web Servers), using the "Webware for Python" toolkit. The General Webserver is customized with presentation data similar to that meant for the General GUI Client. The end-user can interact with the General Webserver by using any standard Web browser as the User Interface, since the General Webserver generates standard HTML for output and processes standard HTML Forms for input.

Such a web server is a very practical way to allow generalized remote access to some of the information held in the database of a Open End-based application. This remote access can optionally include some ability to update the database, both by creating new objects and by editing existing ones.

However, the end-user functionality that can be made available through the Web is only a subset of that which Open End' General GUI Client can supply. In particular, since no widespread "server-side push" technology has yet taken hold, at this time the General-Webserver does not support real-time updates of the information end users are shown. Since the General Webserver is meant to run on the "server side" of Open End, it will not directly support detached operation and batched creations and edits with burst updates, either.

8.3 Other Open End front-ends

Not all user interactions with the Open End back-end are optimally handled in the highly interactive ways typical of GUIs and web access. The BL directly supports the receiving and sending of e-mail (with the standard SMTP protocol). This support can potentially turn any "mail user agent" (MUA) into a (limited and purely textual) "front-end" to some Open End-based applications. Similar "pseudo front-ends" can be designed, along the same guidelines, for other purely textual information interchange protocols, such as the Netnews Transfer Protcol (NNTP). Even deeper integration with arbitrary MUAs is planned for the future through the use of the IMAP protocol: by operating as a dedicated IMAP server, the BL or a specialized front-end can allow more advanced use through mail user-agent programs.

For programmatic updates and reports, Open End supplies "scripting front-ends". Such front-ends enable simple Python scripts to run queries, requests, and updates. Through this scripting interface, and Python's vast power and extensibility, specialized front-end scripts can easily be coded. Such front-end scripts may include the ability to produce and consume XML files, or other specialized formats such as PDF. Non-interactive "clients" can run as daemons (services, in Windows) and are able to mediate between Open End databases and other database formats (such as LDAP). All such non-interactive front-ends rely on just the same fundamental architecture as the scripting front-ends.

A specific "reporting front-end framework", able to perform data mining and produce easily customized, elegantly formatted summary reports, is planned. However, implementation of this reporting front-end has not yet begun. A key enabling technology for this forthcoming front-end framework is the ability to "snapshot" the database as it existed at any time instant, or specific set of instants, in the past. Such "snap-shots" will allow lengthy analysis (for data mining and/or auditing purposes), independent of the continuing editing and updating activity on the "live" database. The ability to snapshot at more than one instant allows specialized front-ends to study, summarize and report on developments in the database between any two given points in time.

9. Conclusion

The Open End framework employs a scalable and flexible layered architecture, based on solid and proven technologies, yet open to the addition of further new ones. This combination of factors makes Open End a high-productivity platform on which to develop and deploy customized application programs, for internal use or for further resale. Consider Open End's advantages when you need to develop workflow-intensive applications. Applications that require extensive storage, retrieval and exchange of information, to coordinate and facilitate the collaboration of multiple knowledge-workers, will be particularly helped by Open End' unique strengths.

Open End AB
Norra Ågatan 10
SE-416 64 Göteborg
Sweden

tel: +46-31-749.08.80
fax: +46-31-749.08.81
email: info@openend.se
web: www.openend.se