SEPM/Linux Tasks Winter Term 2009

SE/Linux Team*

December 14, 2009

1 Task 3

1.1 Focus

This task focuses on building a team, designing an EHR provider node and planning the implementation.

1.2 Tasks

In this task you have to write a document and design the provider node component of the EHR system [5]. This document must include design decisions you made.

You will be arranged in groups. Arrangements will be online shortly after the deadline of task 2. After about one and a half weeks time you will meet with your tutor and explain your design in great detail. The design needs to be detailed enough to be the basis of an actual implementation. The implementation will be done by you and your group in task 4.

1.2.1 General notes

A design document almost always includes some drawing representing the architecture of the software. In this case we have a server and clients. Additionally you will use the Ice middleware. Your drawing for the architectural overview should make this clear. As you already know the SEPM/Linux EHR system is based on components. Identify them and mention the interfaces they have. There are always design decisions you have to make. Discuss and document the basis for your decisions.

Since we make use of the object oriented programming paradigm your server application will contain classes. Draw an UML class diagram representing the class structure of your server. Also think about use cases.

*`se-linux@inso.tuwien.ac.at`

1.2.2 Synchronisation

One very important aspect of the server is synchronisation. The server may be used by more than one client and methods may be called concurrently. Ice potentially handles each call, even on the same method of the same servant, in a different thread. It is your job to properly synchronise these threads if there is danger of data corruption. Document your thoughts and decisions.

1.2.3 Two releases

Based on the design document you will plan the implementation of the provider node. In task 4 you will have to release at least two working versions and present them to your tutor. As the releases must be built they need to include an Autotools-based build system. Implement a part of the functionality for the first release and the rest for the second. This means you will have to decide what functionality goes into which release.

1.2.4 Reading

- Read the Trac documentation at [6].
- Read *at least* the following chapters of the Ice manual [2] to attain deeper knowledge of Ice:
  - Server side slice to C++ mapping
  - Developing a file system server in C++
  - The Ice runtime in detail
    - Communicators
    - Object adapters
    - Object identity
    - Server implementation techniques
    - Proxies
    - Threading models

1.2.5 Requirements

A provider node hosts its local users, identified by account identifiers of the form `<userid>@<provider>`. For patients, a provider node hosts their EHR data. A provider node provides two interfaces:

- An interface for its local users (patients, pharmacists, physicians). This is the `Ehr::Provider` interface you should be familiar with by now.
- An interface for inter-provider communication. This interface is used by other provider nodes to fulfill requests that cannot be served locally. In this task you will design that interface.

The provider node has to use a relational database[1] for storage of the documents and related meta-data of its local users. Keep in mind that only patient accounts have associated documents.

To authenticate its legitimate users, a provider node needs an account database. For every account, at least the public key (or certificate) of that account needs to be stored to be able to authenticate requests made by that account. The easiest way to implement the account database is a directory at filesystem level, containing a certificate in PEM format for every valid account with a well defined naming scheme for the files. But you are free to chose a different storage mechanism, for example a relational database.

For inter-provider requests (a `Ehr::ThirdPartyRequest` where the requestor is hosted at provider *A* and the owner of the EHR is hosted at provider *B*), a provider directory is needed that establishes a mapping from the provider part of an `Ehr::AccountIdentifier` to an Ice endpoint that can be used for communication (*B* is mapped to something like `InterProvider:ssl -h lbzux -p 12345`). Inter-provider requests must be signed by a legitimate provider node and the signature has to be verified before serving the request. For this purpose, the provider directory needs to contain the certificate of every legitimate provider. The provider directory has to be stored in an XML file[2].

SSL has to be used as transport protocol with Ice for all communication and client certificates need to be verified to accomplish sender authentication.

The provider node has to run on at least two of our shell servers (you may choose which, but development on GNU/Linux und NetBSD is easier than on OpenSolaris).

1.2.6 Teamwork

- Get familiar with your new Trac instance.
- Use the Trac Wiki to write the design document of your provider node.
- Draw an ER diagram visualising the data model used for document storage.
- Draw a class diagram.
  - Isolate the database access code into a separate layer. Don't spread SQL everywhere in your code.
  - Keep in mind that some requests can be fulfilled in two ways: either serving them locally, or routing them to a different provider node.
  - In task 4 you will have to write unit tests for your code. Choose a class granularity such that every class can be automatically tested, i.e. it is possible to write code that tests whether the class works correctly.
- Design the Ice interface for inter-provider communication.
- Design the structure of the provider directory.

[1]You will use PostgreSQL [3] in task 4.

[2]You will use libxml++ [4] in task 4.

- Think about security considerations and document the security concept.
- Think about the scalability of the whole system.
- Think about configuration of your provider node.
- You will probably need a logging facility to inspect what your node is actually doing. The log4cxx [1] framework could be very useful in that regard.
- Think about error handling:
  - Make the assumption that clients and provider nodes are evil and cannot be trusted.
  - The node should stay robust even when malfunctioning clients connect.
  - How are you going to ensure this?
  - Every call on an Ice proxy can fail with an exception or can hang for a long time.
- As you already know Ice servers are multi-threaded. If the servants access shared data you have to protect it with mutexes.
  - Plan a locking granularity. One extreme is to use a single lock for all data in the provider node, another extreme is to use one lock for every variable. Both are suboptimal. With one Mutex you don't benefit of concurrency, but with a lot of Mutexes you consume too many resources.
  - The locking granularity determines your level of concurrency. With one global lock only one request can be fulfilled for only one client. How well will your server perform with 100 clients and 100 requests?
  - Read about deadlocks in [7] and plan a strategy on how you can avoid and prevent them.
- *If you have problems ask your tutor for help: email him, arrange an appointment for a consulting hour, post to the forum and join the IRC channel.*

There will be two milestones pre-created for you in your Trac instance. Each represents one release. Divide the implementation into fine-grained working packages that you assign to team members.

- Enter them as task-type tickets into Trac.
- Assign the tasks to team members.
- Team members accept the tasks.
- Assign the tasks to milestones.
- Examples of such tasks are:

- Build system
- pushfd/NetBSD port
- Storage backend for provider directory
- Implementation of `MyFooClass`
- ...

1.3 Deliverables

Use Trac to create and maintain your design document and to do the necessary project management and planning tasks. Your tutor should be able to browse through your tickets and Wiki and get a more or less clear picture on what you are trying to achieve.

Tag the IDL file containing the interface for inter-provider communication in your group's subversion repository as `/submission/task3_final`.

References

[1] Apache Software Foundation: *log4cxx - Short introduction to Apache log4cxx.* `http://logging.apache.org/log4cxx/`. Version: 2008. – [Online; accessed 2008-05-12]

[2] Henning, Michi ; Spruiell, Mark: *Distributed Programming with Ice.* `http://www.zeroc.com/doc/Ice-3.3.0-IceTouch/manual/`. Version: 2008. – [Online; accessed 2009-03-18]

[3] PostgreSQL Global Development Group: *PostgreSQL.* `http://www.postgresql.org/`. Version: 2008. – [Online; accessed 2008-05-11]

[4] The libxml++ Development Team: *libxml++.* `http://libxmlplusplus.sourceforge.net/`. Version: 2007. – [Online; accessed 2007-02-26]

[5] The SE/Linux Team: *The SEPM/Linux EHR System.* `http://se-linux.inso.tuwien.ac.at/se1/tasks.phtml`. Version: 2008. – [Online; accessed 2008-04-20]

[6] The TRAC Team: *The TRAC Project.* `http://trac.edgewall.org/`. Version: 2007. – [Online; accessed 2007-02-26]

[7] Wikipedia: *Deadlock — Wikipedia, The Free Encyclopedia.* `http://en.wikipedia.org/w/index.php?title=Deadlock&oldid=107741299`. Version: 2007. – [Online; accessed 2007-02-26]