LON-CAPA Next Generation Code Manual

LON-CAPA Consortium

March 2, 2015

Contents

Chapter 1

License

The LearningOnline Network with CAPA - LON-CAPA

Copyright (C) 2014 Michigan State University Board of Trustees

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
    GNU General Public License for more details.

Chapter 2

Requirements

2.1 Interface

2.1.1 Fully accessible

All aspects of the interface need to be accessible to screen readers. They should not only function, but also explain themselves. For example, if a number of form fields are optically bundled on screen, they need a fieldset and hidden labels that screen readers can pick up. All interaction with the system needs to be possible by using only the keyboard, no functionality shall depend on only the mouse; if using mice input is convenient or desirable for seeing users, for example in a canvas element, at least an alternative viable textual input method must be provided.

2.1.2 Fully internationalized

The interface and all input elements need to be fully internationalized. This includes menus in helper applications, text direction (right to left), full unicode transparency, and number formats (e.g., comma versus decimal point).

2.1.3 System reacts in less than one second

Any user interaction needs to be acknowledged by the system in less than one second, and non-active interface elements disabled. If a user interaction requires more than one second of processing, progress indicators need to be provided.

2.1.4 Mobile support

Mobile devices, most notably iOS and Android, must be supported. Both instructors and students must be able to conduct everyday course business on mobile devices including smartphones. Some interface functions like MouseOver are not supported in some mobile operating systems, so alternatives must be provided.

2.1.5 No Java or Flash

The interface must not depend on Java or Flash. Instead, JavaScript/HTML5/CSS need to be used.

2.1.6 Cross-browser compatibility

The system needs to support Firefox, Safari, Opera, Chrome, and Internet Explorer. However, the minimum version of Internet Explorer to be considered is Version 11 (keeping in mind that development of the system will take some time, so IE will move on in the meantime).

2.1.7 No complete screen rebuilds

The screen should never completely rebuild. Instead the URL of the screen should always be the server itself and interactivity achieved through AJAX or iframes. Deep-linking and single signon should be enabled but bounce the session back to the normal full screen setup. We are not subservient to other systems.

2.1.8 Easy things easy, hard things possible

The product should be able to directly compete with commercial course management solutions, which means that the 80%-functionality must be as easy to accomplish as in commercial systems. Our heritage demands that we still make the 20% possible.

2.1.9 Preventing users from shooting themselves into the foot

A powerful system will have ample opportunity to configure things in non-sensical ways (for example, having an opening date after the due date, etc). The system must warn students and instructions about potential problems and offer direct ways to remedy the conflict - however, we need to be careful with "auto-correct" or being obnoxiously helpful ("Are you writing a letter?").

2.2 Assets

2.2.1 Backward compatibility

The new LON-CAPA must be backward-compatible to old LON-CAPA using conversion and clean-up processes. Given the size of the LON-CAPA resource pool, more than 99% of the conversion must be fully automatic. The remaining 1% must be identified by the conversion process, so we do not have surprises, and must be fixable. Dynamic metadata from the last decade must be converted to the new system, so we do not lose this usage information.

2.2.2 Printability

Assets have to be printable at different granularity (single, chapter, course) in high quality and compactly. This is essential for exam functionality.

2.3 Data

2.3.1 All changes take effect after at most 10 minutes

Any changes to user roles, course tables of contents, due dates, portfolios, etc, must take effect all across the network in at most 10 minutes, and this includes running sessions.

2.3.2 Separation

No content or interface handlers must directly touch local disks. All network functionality must be abstracted away beyond the "entity" level.

2.3.3 Times

All stored times are UTC. Conversion should only happen on input/output. We need to make sure we are not running into the Y2038-bug.

2.4 Network functionality

2.4.1 SSL

All communication between servers is at least 4096-bit encrypted. LON-CAPA issues server and client certificates. Servers need full reverse DNS.

2.4.2 Predictable storage

To observe privacy and export rules, any permanent user data (beyond temporary session information) needs to be stored in a predictable location. The domain concept of LON-CAPA facilitates this.

2.4.3 Authentication

It is the responsibility of the homeserver of a user to authenticate the user.

2.4.4 Authorization

Authorization in the system is based on roles. A user can have several (time-limited) roles, each of which provides a certain set of privileges within a particular realm.

2.4.5 Run everywhere

All system functionality must be available on all servers, so that content, courses and users can be served from any machine in the cluster (subject to freely configurable limitations).

2.4.6 Servers

Servers need to be dedicated to LON-CAPA for security and privacy reasons. In the age of virtual machines, that should not be a problem.

2.5 Coding environment

2.5.1 Linux Distributions

We are developing on CentOS/RedHat Enterprise. We do not have the time to support other distributions ourselves, and will only accept them if some individual guarantees to be their longterm curator. We will not accept short-life-cycle distributions, as only long-life-cycle distibution guarantee the required security and stability. In the age of virtual machines, it should not be a problem to run a VM with a particular distribution, particularly since the machines are completely dedicated anyway and if ready-to-run images are provided.

2.5.2 Open source

The software is licensed under the GNU General Public License Version 3 or higher, copyright Michigan State University Board of Trustees. Only software components that are compatible with this license shall be used.

2.5.3 Porting from LON-CAPA 2.x

The project will not succeed in a timely fashion if we do not port code from LON-CAPA 2.x. Eventually, program code will need to be cleaned up and moved over in order to keep timelines. This will mean that sometimes we need to sacrifice using the "latest-greatest" technology in order to facilitate portability of code or even just algorithms.

2.6 Scalability testing

During development, all course management functionality needs to be tested with

- Courses of 5000 students or more
- Courses with 2000 assets or more
- Courses with 1500 assessment items or more

The systems needs to remain responsive. MOOCs may reach 20,000 users or more.

Chapter 3

Installation

This is the description of the installation for *code developers*, it is not yet how the system will be installed eventually in production.

3.1 Linux

LON-CAPA runs on Linux. Since the target eventually is a server, enterprise editions of Linux are recommended, and in particular CentOS. For CentOS, a "minimal desktop" installation is recommended. The 64 bit version of CentOS 6 should be used. MongoDB recommends and the included repository files assume 64 bit. CentOS 7 currently has a bug which interferes with the installation of Perl modules.

For developers, a virtual machine is good enough: something like two cores, four GB RAM, and 40 GB disk will do fine.

3.2 Certificate IP/DNS considerations

LON-CAPA is a networked system, and all inter-server communication is secured via certificates. It is important that the machine has a stable IP address and DNS. This works fine for simple virtual machines, where this address is `localhost:localdomain`, and the download includes test certificates for this basic configuration.

When using a real server setup building an actual cluster or using a real server, a customized certificate from the LON-CAPA certificate authority is required. Gerd Kortemeyer (for now) can make those certificates, corresponding to `LONCAPA.crt`.

3.3 Updating

It is recommended to update your operating system as well as the Perl package manager (CPAN). All commands in this chapter should be run as root.

```
$ yum update
```

The following CPAN configuration will automatically install prerequisite modules and greatly reduce the number of prompts during installation.

```
$ yum install perl-CPAN
$ cpan
cpan> o conf prerequisites_policy 'follow'
cpan> o conf build_requires_install_policy yes
cpan> o conf commit
cpan> install Bundle::CPAN
cpan> exit
```

3.4 Disable SELinux

For now, SELinux must be disabled. Edit the file `/etc/sysconfig/selinux` and set the parameter

```
SELINUX=disabled
```

For this to take effect, you will need to reboot your machine.

3.5 Downloading

1. get a username at Github, notify Gerd Kortemeyer (for now) to be listed as collaborator
2. after that, clone the repository

```
$ yum install git
$ git clone https://github.com/gerdkortemeyer/loncapa
```

3.6 Install LON-CAPA

- in subdirectory `install`, use `install_packages.sh` (do this on a fast connection, there are a ton of libraries, etc)
- in subdirectory `testcerts`, use `install_test_certs.sh`
- if not `localhost:localdomain`, now copy customized server certificates into `/home/loncapa/certs` — please do not overwrite the sandbox certificates in the repository
- if not `localhost:localdomain`, the cluster table and manager (see Section 5.3.1 on page 13), as well as (possibly) the server name in the Apache configuration `httpd.conf` need adjusting — please do not overwrite the sandbox configurations in the repository
- back in `install`, use `install.sh`
- see if the whole thing starts. Then call `http://localhost/test` - that should make an initial user "zaphod" with password "zaphodB" (for now)

3.7 If things don't work

If things don't work, the following log-files might be helpful:

- Apache server error logs `/etc/httpd/logs/error_log` and `/etc/httpd/logs/ssl_error_log` (or equivalent in distributions other than CentOS)
- LON-CAPA log files `/home/loncapa/logs/errors.log` and `/home/loncapa/logs/warnings.log`

Chapter 4

URL Translation

4.1 Accessing handlers

Handlers are defined in lc.conf using Location and LocationMatch statements. For example

```
<Location /menu>
SetHandler perl-script
PerlAccessHandler Apache::lc_auth_optional
PerlHandler Apache::lc_ui_menu
</Location>
```

defines how the URL /menu is handled. During the Apache request cycle, first the PerlAccessHandler is called. There are two versions,

- lc_auth_acc, which requires a session and loads the session environment - otherwise, an error handler usually points to the login handler
- lc_auth_optional, which only loads a session environment if a session is active

The statement then defines lc_ui_menu as the response handler. Additional error handlers can be called, for example to route back to the login screen.

Some handlers also have cleanup handlers, which may do the actual work, for example

```
<Location /async>
SetHandler perl-script
PerlAccessHandler Apache::lc_auth_acc
PerlHandler Apache::lc_ui_async
PerlCleanupHandler Apache::lc_ui_async::main_actions
</Location>
```

This handler can reply to the browser immediately and then do the actual work in the cleanup phase. This is most useful in connection with asynchronous AJAX requests.

4.2 Accessing assets

Access to assets is more complicated. They are caught in the global translation handler

```
PerlTransHandler Apache::lc_trans
```

which (at some point) looks/looked like this:

```
sub handler {
    my $r = shift;
# We care about assets
    if ($r->uri=~/^\/asset\//) {
# First check if we can even find this
       my $filepath=&Apache::lc_entity_urls::url_to_filepath($r->uri);
       unless ($filepath) {
```

```
# Nope, this does not exist anywhere
          return HTTP_NOT_FOUND;
       }
# Is this locally present?
       unless (-e $filepath) {
# Nope, we don't have it yet, let's try to get it
          unless (&Apache::lc_entity_urls::replicate($r->uri)) {
# Wow, something went wrong, not sure why we can't get it
             &logwarning("Failed to replicate ".$r->uri);
             return HTTP_SERVICE_UNAVAILABLE;
          }
       }
# Bend the filepath to point to the asset entity
       $r->filename($filepath);
       return OK;
    } elsif ($r->uri=~/^\/raw\//) {
       $r->filename(&Apache::lc_entity_urls::raw_to_filepath($r->uri));
       return OK;
    }
# None of our business, no need to translate URL
    return DECLINED;
}
```

This deals with URLs of the type /asset and /raw, and does the translation of URL-paths to the asset entity:domain, as well as versioning and replication. By the time this handler is done, the asset is either present or we throw a NOT_FOUND.

4.3 System Pages

System pages such as the dashboard or the preferences are actually HTML-pages, which include LON-CAPA XML. They use the same XML parser as for example homework pages. An example is lc_preferences.html, the user preferences screen:

```
<html>
<head>
<title>Preferences</title>
<script src="/scripts/lc_preferences.js"></script>
</head>
<body>
<h1><localize>Preferences</localize></h1>
<p>
<span class="lcstandard"><localize>You can modify your user preferences below.</localize></span>
<span class="lcerror"><localize>A problem occured, please try again later.</localize></span>
<span class="lcproblem"><localize>A problem occurred while saving your preferences.</localize></span>
<span class="lcsuccess"><localize>Your preferences were saved.</localize></span>
 
</p>
<lcform id="preferencesform">
<lcformtable>
<lcformtableinput type="language" description="Language" id="language" default="user" />
<lcformtableinput type="timezone" description="Timezone" id="timezone" default="user" />
</lcformtable>
<lcformtrigger id="storebutton" description="Store" />
</lcform>
</body>
</html>
```

4.4 Deep Links

Deep linking works via the `lc_ui_direct` handler. It utilizes the token-mechanism (see Section 8.5 on page 28).

The URLs for deep linking start with `/direct`.

Chapter 5

Network

5.1 Overview

LON-CAPA has logical layers, see Fig. 5.1:

- Clients talk to particular servers in the network
- Requests from clients are handled by handlers (at the "ui" level)
- Handlers talk to the web service layer (the "entity" level) — the webservices layer abstracts away the fact that the network is distributed
- Databases and file systems

5.2 Nodes

LON-CAPA is built as a network of servers, which can host sessions and store data. LON-CAPA can have several independent clusters of such servers, but it is expected that in real operation, there is essentially one production cluster. In principle, any server in a cluster can host sessions for any user in the cluster, and any server in the network should be able serve any asset in the system. In practice, limitations may be put on this to preserve privacy and copyright.

There are two classes of servers: access and library. The difference is that library servers permanently store data, and access servers do not. This means that library servers need to remain stable and have backup, which access servers can be added or removed on demand to absorb session loads.

5.3 Cluster Table

5.3.1 Configuration

Clusters are established by cluster tables, see below for an example

```
{ domains   : { 'msu'      :    { name    : 'Michigan State University',
                                  class   : 'university',
                                  locale  : 'en-us',
                                  timezone: 'America/Detroit' },
                'sfu'      :    { name    : 'Simon Fraser University',
                                  class   : 'university',
                                  locale  : 'en-ca',
                                  timezone: 'America/Vancouver' },
                'ostfalia' :    { name    : 'Ostfalia University of Applied Sciences',
                                  class   : 'university',
                                  locale  : 'de',
                                  timezone: 'Europe/Berlin' },
                'elps'     :    { name    : 'East Lansing Public Schools',
                                  class   : 'k12',
                                  locale  : 'en-us',
```

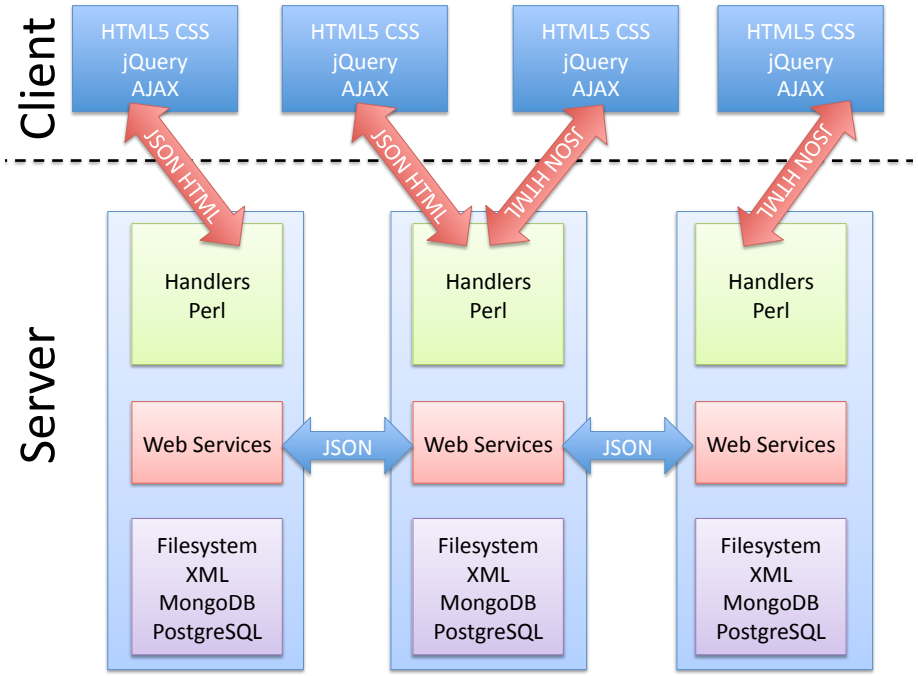

Figure 5.1: General overview of LON-CAPA Next Generation.

```
                                timezone: 'America/Detroit' }
            },
  hosts     : { 'zaphod'   :    { address : 'zaphod.localdomain',
                                  default : 'msu',
                                  domains : { 'msu'      : { function : 'library' },
                                              'elps'     : { function : 'library' },
                                              'sfu'      : { function : 'access'  }
                                            }
                                },
              'marvin'   :    { address : 'marvin.localdomain',
                                  default : 'msu',
                                  domains : { 'msu'      : { function : 'library' },
                                              'elps'     : { function : 'access'  },
                                              'sfu'      : { function : 'access'  }
                                            }
                                },
              'arthur'   :    { address : 'arthur.localdomain',
                                  default : 'sfu',
                                  domains : { 'msu'      : { function : 'access'  },
                                              'elps'     : { function : 'access'  },
                                              'sfu'      : { function : 'library' }
                                            }
                                },
              'slarti'   :    { address : 'slartibartfast.localdomain',
                                  default : 'ostfalia',
                                  domains : { 'ostfalia' : { function : 'library' },
                                              'elps'     : { function : 'access'  },
                                              'sfu'      : { function : 'access'  }
                                            }
                                }
            }
}
```

Thus a particular server can serve more than one domain in different functions. Internally, servers are identified by the internal ID, e.g. "slarti" — this allows migrating nodes between hardware.

5.3.2 Homeservers

Each entity in the system has a so-called homeserver within its domain which holds the authoritative and permanent copy of their data.

5.3.3 Cluster manager

One server in any cluster is the cluster manager. This machine holds the authoritative copy of the cluster configuration table. Who is cluster manager is configured in

```
/home/loncapa/cluster/cluster_manager.conf
```

This needs to be the full DNS name of the cluster manager, and essentially configures which cluster a server is a member of.

The cluster manager can trigger updating this table on other servers via

```
# On non-cluster manager, this triggers fetching the cluster table
#
<Location /fetch_cluster_table>
SetHandler perl-script
PerlHandler Apache::lc_init_cluster_table
SSLRequireSSL
SSLVerifyClient require
SSLVerifyDepth 2
</Location>
```

The other server will then fetch the cluster table via the cluster manager's

```
# Cluster manager serves up authoritative cluster table
#
<Location /cluster_table>
SetHandler perl-script
PerlHandler Apache::lc_cluster_table
SSLRequireSSL
SSLVerifyClient require
SSLVerifyDepth 2
</Location>
```

5.4 Connections

Connections between the servers are mediated via https.

5.4.1 Connection authentication

Connections are authenticated via a configuration in lc.conf:

```
<Location /connection_handle>
SetHandler perl-script
PerlHandler Apache::lc_connection_handle
SSLRequireSSL
SSLVerifyClient require
SSLVerifyDepth 2
</Location>
```

The certificates need to be located in /home/loncapa/certs/.

5.4.2 Connection handling

Connections are handled by the module lc_connection_handle. The following examples are associated with namespaces (see Section 8.6 on page 28), but the mechanism applies to almost all network communication and storage needs. Any routines that should be externally accessible need to be registered with the module, like so:

```
BEGIN {
    &Apache::lc_connection_handle::register('modify_namespace',undef,undef,undef,
                                            \&local_modify_namespace,'entity','domain','name','data');
    &Apache::lc_connection_handle::register('dump_namespace',undef,undef,undef,
                                            \&local_json_dump_namespace,'entity','domain','name');
}
```

The first argument is the command name, the following three arguments will be used for additional security (not yet implemented), the next argument is the pointer to the subroutine that deals with this, and the remaining arguments are the named arguments of the function.

5.4.3 Local versus remote

Routines need to determine if the data is local or remote, depending on the homeserver of the entity, like so:

```
#
# Dump namespace from local data source
#
sub local_dump_namespace {
   return &Apache::lc_mongodb::dump_namespace(@_);
}

sub local_json_dump_namespace {
   return &Apache::lc_json_utils::perl_to_json(&local_dump_namespace(@_));
}

#
# Get the namespace from elsewhere
#
sub remote_dump_namespace {
   my ($host,$entity,$domain,$name)=@_;
   my ($code,$response)=&Apache::lc_dispatcher::command_dispatch($host,'dump_namespace',
                                "{ entity : '$entity', domain : '$domain', name : '$name' }");
   if ($code eq HTTP_OK) {
      return &Apache::lc_json_utils::json_to_perl($response);
   } else {
      return undef;
   }
}


# Dump current namespace for an entity
# Call this one
#
sub dump_namespace {
   my ($entity,$domain,$name)=@_;
   if (&Apache::lc_entity_utils::we_are_homeserver($entity,$domain)) {
      return &local_dump_namespace($entity,$domain,$name);
   } else {
      return &remote_dump_namespace(
             &Apache::lc_entity_utils::homeserver($entity,$domain),$entity,$domain,$name);
   }
}
```

The named arguments are sent as JSON.

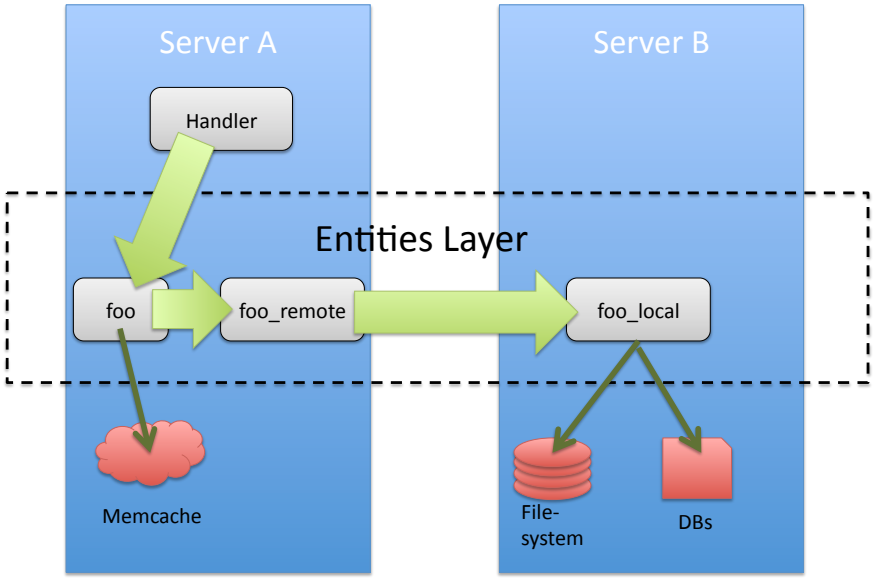

Figure 5.2: Scheme for storing and retrieving data.

Most routines that register services are in `/entities` and contained in modules called `lc_entity_...` — other handlers should use the routines provided in the entity-handlers and *never* directly access the disk, the databases, or the network. The routines to be called are the ones without "local" or "remote" (see Fig. 5.2).

5.5 Replication

Assets are replicated between servers. Any server in the network can serve any asset in the server under the same URL path. This is transparent, i.e., the same URL path can be requested from any server, regardless of whether or not that asset is already locally present. The server needs to locate the asset in the network and replicate it if needed.

Assets are versioned. Replication works by copying a certain version of an asset from its homeserver using

```
<LocationMatch "^/raw/">
SetHandler perl-script
PerlAccessHandler Apache::lc_raw_acc
SSLRequireSSL
SSLVerifyClient require
SSLVerifyDepth 2
</LocationMatch>
```

This separate address (as opposed to the normal "/res" address) is important, since we need the raw XML, not a rendered version.

The system independently caches the most recent version of a resource. Once this cache expires, a server again asks the homeserver of the asset what the most recent version is via the "current_version" command that is registered by lc_entity_urls. If this is not the version currently stored, a new version is fetched.

Chapter 6

Entities

6.1 Identification

Users, courses, and assets are entities in the system. Since the network has domains, an entity is identified by two components:

- An entity code (in the system internally often referred to as just "entity", e.g., "qLLTNbdEhQaxQ8AZyYp"
- The domain (e.g., "msu")

Thus, entities should *always* be refered to by both, qLLTNbdEhQaxQ8AZyYp:msu. Just specifying the entity code is not enough!

Entity codes are guaranteed to be unique within a domain, but not between domains.

Entities do not change. An individual, course, or assets always keeps its entity:domain.

6.2 Storage and Interface

Information in the system is always stored by entity:domain, not by usernames or course codes. However, entity:domain is not exposed on the interface level, here we talk in terms of usernames, personal ID numbers, or course codes.

6.3 Users

Usernames and personal ID numbers (PIDs) are properties of the user entity. They need to be unique within a domain, thus to fully specify a user entity in this fashion, one needs to specify the username *and* domain. Just specifying the username is not enough!

Usernames and PIDs can change. For example, username changes can happen due to life events or because the user wants a vanity ID. At most universities, usernames once given out are never recycled. Also, PIDs can change if a person changes from student to faculty/staff or vice versa. Once again, PIDs at most places are never recycled.

The system should thus assume:

- At any given point in time, users have one primary username and PID.
- The system needs to still support the old usernames and PIDs, as particularly during mid-semester changes, uploadable grading lists or other spreadsheets may still include the old username/PID

Each username or PID points to one and only one user entity. However, a user entity can have more than one supported username or PID.

6.4 Courses

Courses are treated very similarly to users. They have course IDs such as "phy231fs14", which might have. These are treated the same way as usernames or PIDs.

Communities are special kinds of courses which do not have grade book, and which have different associated roles. Internally, they are distinguished by the "type" in the course profile being "community" instead of "regular."

Courses can have sections and groups.

- Sections usually correspond to educational venues such as laboratories or recitations. A student can only be in one section at a time.

- Groups are more like teams, working together. A student can be in more than one group at a time

Internally the two are distinguished by "type" in the course/community profile.

6.5 Assets

6.5.1 URLs

Assets are HTML pages, problems, images, movies, etc, which are stored in user portfolios. Instead of usernames, PIDs, or couse IDs, assets have URLs. One asset can have more than one URL, however, the URL of published assets looks like

```
/asset/n/3/msu/smith/...
```

where the first two components after "asset" designate the version, followed by the domain and the publishing author.

The above example explicitly asks for version number 3 of the asset. Other ways are

```
/asset/as_of/2014-01-08_04:05:06/msu/smith/...
```

which asks for the version as-of a certain date. Finally,

```
/asset/-/-/msu/smith/...
```

asks for the most recent version.

If at all possible, relative URLs should be used, so version arguments like "as_of" get preserved to also get the right versions of dependent files.

6.5.2 Profiles and metadata

Both users and courses have profiles, which store basic information like their full name or title, or any configuration preferences. Assets have metadata, which is used for various cataloging purposes and populated with both static and dynamic metadata. These are stored in MongoDB..

6.5.3 Storage

Assets are physically stored on the file system under their entity, not their URL, see Section 8.3 on page 27. An asset entity can have more than one URL. The URLs are a property of the entities and translated using the translation handler, see Section 4.2 on page 11.

The authoritative copy of assets sits on the author's homeserver. During replication (Section 5.5 on page 17), copies are made between servers.

6.5.4 Publication and Versioning

Assets move from the "_wrk" version to the real next version during the process of publication. As a new version of a resource is generated, this triggers the updating across servers through replication (see Section 5.5 on page 17).

Chapter 7

Authorization

7.1 Roles

7.1.1 Role definition

Authorization is handled via roles. Each role has a certain set of privileges, which are defined within their realms through roles.json. Here's an excerpt:

```
{
"superuser"           : { "realm"  : "system",
                          "system" : {
                                       "view_role"       : "1",
                                       "modify_role"     : {
                                                             "domain_coordinator" : "1"
                                                           }
                                     }
                        },
"domain_coordinator" : { "realm"  : "domain",
                          "domain" : {
                                       "view_role"       : "1",
                                       "modify_role"     : {
                                                             "course_coordinator" : "1"
                                                           },
                                       "modity_auth"     : "1"
                                     }
                        },
"course_coordinator" : { "realm"  : "regular",
                          "course" : {
                                       "view_role"       : "1",
                                       "modify_role"     : {
                                                             "course_coordinator" : "1",
                                                             "instructor"         : "1",
                                                             "teaching_assistant" : "1",
                                                             "student"            : "1"
                                                           },
                                       "modify_settings": "1",
                                       "edit_toc"       : "1",
                                       "access_content" : {
                                                             "open"               : "1",
                                                             "closed"             : "1",
                                                             "hidden"             : "1"
                                                           },
                                       "modify_grade"   : "1",
                                       "notify"         : "1"
                                     }
                        },
```

```
"instructor"         : { "realm"  : "regular",
                         "course" : {
                                      "access_content" : {
                                                           "open"               : "1",
                                                           "closed"             : "1",
                                                           "hidden"             : "1"
                                                         }
                                    },
                         "section": {
                                      "view_role"      : "1",
                                      "modify_grade"   : "1",
                                      "notify"         : "1"
                                    }
                       },
"teaching_assistant" : { "realm"  : "regular",
                         "course" : {
                                      "access_content" : {
                                                           "open"               : "1"
                                                         }
                                    },
                         "section": {
                                      "view_role"      : "1",
                                      "modify_grade"   : "1",
                                      "notify"         : "1"
                                    }
                       },
"student"            : { "realm"  : "regular",
                         "course" : {
                                      "access_content" : {
                                                           "open"               : "1"
                                                         }
                                    }
                       },
"community_organizer": { "realm"  : "community"
                       },
"member"             : { "realm"  : "community"
                       },
"author"             : { "realm"  : "user"
                       },
"co_author"          : { "realm"  : "user"
                       }
}
```

Internally, the role names like course_coordinator are used everywhere and also sent to the interface. The localization is used to make these human-readable like "Course Coordinator" or "Kurskoordinator."

7.1.2 Role storage

Roles are stored in two places: with the user (authoritative) and in lookup tables. The module lc_entity_roles manages both and no manipulation of roles should happen below this level.

With user

A role record of a user might look like this:

```
 {
'course' => {
            'msu' => {
                      'N41nvoxGJ9NvZxuIrD3' => {
```

```
                                        'any' => {
                                               'member' => {
                                                         'enddate' => '2018-01-08 04:05:06',
                                                         'manualenrolldomain' => 'msu',
                                                         'startdate' => '1998-01-08 04:05:06',
                                                         'manualenrollentity' => 'ggf21wqffas'
                                                          }
                                                 }
                                      },
                    'Kk5vRpSiCp63Wu46Mox' => {
                                        'any' => {
                                                'course_coordinator' => {
                                                          'enddate' => '2018-01-08 04:05:06',
                                                          'startdate' => '1998-01-08 04:05:06',
                                                          'manualenrolldomain' => 'msu',
                                                          'manualenrollentity' => 'ggf21wqffas'
                                                           }
                                                 }
                                        },
                    'ISz8egcz03Uyyr8LAdz' => {
                                        'section' => {
                                                     '006' => {
                                                        'instructor' => {
                                                           'enddate' => '2015-01-08 04:05:06',
                                                           'startdate' => '1998-01-08 04:05:06',
                                                           'manualenrolldomain' => 'msu',
                                                           'manualenrollentity' => 'ggf21wqffas'
                                                               }
                                                      },
                                                     '010' => {
                                                        'teaching_assistant' => {
                                                           'enddate' => '2017-01-08 04:05:06',
                                                           'manualenrolldomain' => 'msu',
                                                           'startdate' => '1998-01-08 04:05:06',
                                                           'manualenrollentity' => 'ggf21wqffas'
                                                                }
                                                       }
                                                      }
                                      },
                  },
             }
'domain' => {
             'msu' => {
                       'domain_coordinator' => {
                                        'enddate' => '2016-01-08 04:05:06',
                                        'manualenrolldomain' => 'msu',
                                        'startdate' => '1299-01-08 04:05:06',
                                        'manualenrollentity' => 'qhhhf21wqffas'
                                      }
                  }
         },
'system' => {
        'superuser' => {
                         'enddate' => '2016-01-08 04:05:06',
                         'startdate' => '1299-01-08 04:05:06',
                         'manualenrolldomain' => 'msu',
                         'manualenrollentity' => 'qhhhf21wqffas'
                       }
      }
```

```
}
```

With domain or system

There are also lookup tables to see who all has a certain role. These are shown in Section 8.1.1 on page 26.

Chapter 8

Storage

8.1 Databases

The system uses two databases: PostgreSQL and MongoDB.

- PostgreSQL is a traditional relational database which is used for predictable data that can reside in tables. These are often lookup-tables that need fast search
- MongoDB is a noSQL database which is used for flexible, structured data. Searches are rare and not performance-critical.

Data only sits on the homeserver of the user or course.

8.1.1 PostgreSQL tables

The tables in PostgreSQL are

URL table

The table is used to look up the entity code for a certain URL. More than one URL can point to the same asset. The domain is not stored, since it is already encoded in the URL.

```
#
# Make the URLS table
#
create table urls
(url text primary key not null,
entity text not null)
```

User table

This table is used to look up the entity code of usernames. More than one username can point to the same entity code. Both entity code and username would be in the same domain.

```
#
# Make the user lookup table
# Get the entity for a username
#
create table userlookup
(username text not null,
domain text not null,
entity text not null,
primary key (username,domain))
```

PID table

Same as user table for PIDs.

```
#
# Make the pid lookup table
# Get the entity for a PID
#
create table pidlookup
(pid text not null,
domain text not null,
entity text not null,
primary key (pid,domain))
```

Course ID table

Same as username and PID tables for course IDs.

```
#
# Make the courseID lookup table
# Get the entity for a courseID
#
create table courselookup
(courseid text not null,
domain text not null,
entity text not null,
primary key (courseid,domain))
```

Homeserver table

Table used to look up the homeserver of an entity:domain.

```
#
# Make the homeserver lookup table
#
create table homeserverlookup
(entity text not null,
domain text not null,
homeserver text not null,
primary key (entity,domain))
```

Roles

This is for looking up who has certain roles. It is not the authoritative version, the authoritative record is stored with the user (see Section 7.1.2 on page 22).

```
#
# The role table
# These are the roles on this server
# The primary cluster server (and only the primary cluster server)
# also has the system and domain-wide roles
# This is for lookup, the actual roles are with the users
#
create table rolelist
(roleentity text,
roledomain text,
rolesection text,
userentity text not null,
userdomain text not null,
role text not null,
startdate timestamp,
enddate timestamp,
manualenrollentity text,
manualenrolldomain text,
primary key (roleentity,roledomain,rolesection,userentity,userdomain,role))
```

Assessment table

This has the standardized data about assessments in courses, which is used for gradebooks. More structured data to be called up when the history of a problem is desired, or if it is to be brought up on the screen, resides in MongoDB.

```
#
# This is the big table of course assessments on the homeserver of the courses
# Authoritative
#
create table assessments
(courseentity text not null,
coursedomain text not null,
userentity text not null,
userdomain text not null,
resourceid text not null,
partid text not null,
scoretype text,
score text,
totaltries text,
countedtries text,
status text,
responsedetailsjson text,
primary key (courseentity,coursedomain,userentity,userdomain,resourceid,partid))
```

8.2 Caching

The system uses two caching mechanisms: in-memory and Memcached

- In-memory is occasionally used if a particular process needs to preserve variables, so they do not need to be reinitialized every time. Examples are database handles, etc.
- Memcached is the main caching mechanism, which caches data in memory across processes. The cache items have associated expiration times. The time to refresh these caches is distributed across processes and users.

8.3 File system

8.3.1 Assets

Assets are stored on the file system in /home/loncapa/res. The authoritative copy of assets is on the author's homeserver (see section 5.3.2 on page 15), but they are copied through replication (see section 5.5 on page 17) to other servers.

The filesystem uses the entity code as part of the location. For example,

```
/home/loncapa/res/msu/M/4/O/W/M4OWjvPTQCIE6o8RTMJ_3
```

is version 3 of the asset with entity code M4OWjvPTQCIE6o8RTMJ in domain msu.

Before publication, assets have the extension _wrk,

```
/home/loncapa/res/msu/M/4/O/W/M4OWjvPTQCIE6o8RTMJ_wrk
```

8.3.2 Tables of Content

Tables of content of courses are stored as assets in JSON format. The URL of the table of contents is always *domain/courseid*/`toc.json` (where it is stored is of course elsewhere).

An example is

```
[

    {
        "content":
            [
```

```
                {
                    "content":
                        [

                            {
                                "active":1,
                                "url":"\/ostfalia\/smith\/energy\/energy.html",
                                "id":"Jia0iYTrMUiQfWrKuQN_2478_1406980671",
                                "hidden":0,
                                "type":"asset",
                                "title":"capacity energy field resistance"
                            },

                            {
                                "content":
                                    [

                                        {
                                            "active":1,
                                            "url":"\/msu\/jones\/voltage\/power.html",
                                            "id":"Jthi8TegI0GZ9Dr812N_2478_1406980671",
                                            "hidden":0,
                                            "type":"asset",
                                            "title":"capacity resistance field"
                                        },
...
                                    ],
                                "active":1,
                                "id":"JnJ4eBNgBgCaP8VIMq5_2478_1406980671",
                                "hidden":0,
                                "type":"folder",
                                "title":"Chapter electronic electronic energy electronic"
                            },
...
```


Folder content is stored (in order) in arrays. Each resource in a course has a unique ID, e.g., "Jthi8TegI0GZ9Dr812N_2478_1406980671", called "assetid". These IDs are system-wide unique.

When this folder-nested table of contents is sent to the client, it is "linearized" with a parent field.

When performance data is stored, it is stored under this assetid along with the course. As assetids are unique, whole segments of tables of content can be copied between courses.

8.4 Programmatic access

Databases should not be touched by any higher order handlers. While drivers are located in the /databases GIT directory, e.g., lc_memcached.pm lc_mongodb.pm lc_postgresql.pm, these should not be called. Instead, handlers should access the abstractions in the /entities GIT directory, and there only the routines that are not starting with "remote" or "local".

8.5 Tokens

Tokens are a way to exchange records between servers. They are only stored in Memcache for a few minutes. Each token has an ID under which another server can retrieve the information.

8.6 Namespaces

Namespaces are a mechanism to store away datastructures under arbitrary names, similar to how namespaces are handled in the first-generation LON-CAPA. This is a very flexible mechanism, but not performance-optimized and not efficiently searchable. It should be used for convenient storages of occassionally-used datastructures.

Chapter 9

XML

9.1 Document types

Uploaded documents that are not modified are kept as-is, and served with as little changes as possible (to ensure that uploaded dynamic websites keep working). This includes HTML documents and documents in many other formats (CSS, Javascript, XML, media files etc...). HTML documents that are modified with LON-CAPA, as well as new content documents, are using a specific XML format and are called *LON-CAPA documents*.

9.2 Format and structure of LON-CAPA documents

LON-CAPA documents use the XML syntax (they are *well-formed*). The XML language is defined by an XML schema, `loncapa.xsd`. LON-CAPA documents are normally valid with this schema, but they can also be invalid in some cases (for instance, when they have been converted from invalid documents and could not be fixed safely, or when they are incomplete). LON-CAPA will assume that these documents follow the schema, but will also gracefully handle invalid documents with controlled degradation or error messages. The schema is using a mixture of HTML and LON-CAPA-specific elements, and the root element is `loncapa`. All these elements are processed with the built-in parser (see chapter 11 on page 35): the rendering of any tag can be defined or modified.

9.2.1 Classic LON-CAPA Content

More than 99% of the classic LON-CAPA content needs to be automatically convertable into the new format. The conversion is done by `convert.pl`, which can convert a whole directory with its sub-directories.

9.2.2 Overview of differences to classic LON-CAPA

Major differences to classic LON-CAPA are:

- The syntax is XML instead of HTML.
- Newly created HTML documents and problem documents no longer have a different format: they are now all LON-CAPA documents.
- There can be more than one problem in a document:

```
<loncapa>
  <section>
    <h1>Electrical Current</h1>
    <section>
      <h1>Charge per Time</h1>
        ... (random HTML) ...
      <problem>
        ...
      </problem>
    </section>
    <section>
      <h1>Measuring Current</h1>
```

```
      ...
      <problem>
        ...
      </problem>
    </section>
  </section>
</loncapa>
```

- `problem` elements always have parts.
- Several elements get renamed, e.g.,
  - `<script type='loncapa/perl'>` becomes `<perl>`
  - `<m>` becomes HTML (when it was not math), `<tm>` (for inline math) or `<dtm>` (for display math).
- Because of the XML syntax, CDATA sections or entities are used on disk, e.g.,
  - `<perl><![CDATA[if ($a<$b) { $a=42; }]]></perl>`
  - `<tm>a&lt;b</tm>`
- All tags are lowercase.
- LaTeX can no longer be used outside of math. `<tex>` is no longer supported. Printing is now done with HTML and CSS. Print-specific content can be added with `<print>`, and print-specific style or layout can be changed with CSS (with @media print).

9.3 Editing

9.3.1 Main editors

Editors will be client-side. There will be two user interfaces:

- Graphical editor
- Text editor

The graphical editor will only able to handle well-formed documents. The text editor should be able to handle even non-well-formed documents (i.e., any text document). Both editors should be able to handle invalid documents, although of course the interface might be affected, depending on the validity problem. They will both help to create valid documents, and the graphical editor will always produce well-formed documents. The text editor will be "pseudo source code," in that it will not expose CDATA sections and character entities. It is meant as a textual user interface, not a source editor. More information about the graphical editor is available in chapter 15 on page 53.

9.3.2 Source editor

There will be an emergency raw source code editor, which could be used to directly access the source code. This editor could be used to rescue files that become corrupted, or to clean up imported code.

9.3.3 Publication

Documents that are not well-formed cannot be published. A number of additional constraints will be added for publication (to be defined). There will also be warnings for all invalid documents.

Chapter 10

Conversion

10.1 Scope

The conversion converts the latest version of all HTML, problem, exam, survey and library documents.

10.2 Script usage

`loncapa/conversion/convert.pl` converts a document or a directory recursively when given its path on the command line. Currently it creates `.lc` files, without touching the original files.

10.3 Encoding

To read the documents correctly, the character encoding must be known. It is assumed to be UTF-8 by LON-CAPA, but this assumption is wrong for many documents. Many of them are encoded in cp1252 or ISO-8859-1, but there are also some using MacRoman. There is no way to guess without error the character encoding used when it is not UNICODE, so we have to make a guess. To do that, the conversion counts the most common non-ASCII characters in the documents (especially in English, German, Spanish and Portuguese). It converts all documents to UTF-8.

10.4 Syntax

Because the new documents use the XML syntax, a syntax conversion is necessary. It tries to preserve what was expected from documents, with the LON-CAPA tags (interpreted by LON-CAPA) and the HTML ones (mostly interpreted by web browsers). This is not easy, because the documents are not well-formed, and cannot be parsed without risks for errors. They also contain many syntax errors, which the conversion tries to fix. CDATA sections are used for blocks of code that might contain many special characters. Otherwise character entities are used to replace special XML characters.

10.5 Document structure

The new document structure encloses the previous content of the file by a root `loncapa` element, or (for libraries) a `library` element. `head` is replaced by `htmlhead`, and moved to the beginning. `body` is replaced by an empty `htmlbody` element positioned after `htmlhead` (if present), with the attributes from the `body` element. Most documents will not have any `htmlhead` or `htmlbody` element, because they are not necessary. The non-LON-CAPA `meta` elements are removed.

10.6 Removed elements

The following elements are removed without replacement: `startouttext` and `endouttext` (also removed when misspelled), `startpartmarker`, `endpartmarker`, `displayweight`, `displaystudentphoto`, `basefont`, `displaytitle`, `displayduedate`, `allow`, `allows`, `x-claris-tagview`, `x-claris-window`, `x-sas-window`.

10.7 tex, web and m

Because LaTeX is no longer used for printing, `<tex>`, `<web>`, `&tex` and `&web` are replaced by HTML when it is possible to do so automatically. When it is not, `tex` and `web` will have to be fixed by hand (replaced by equivalent CSS). In this case the conversion issues a warning. `<m>` is replaced by a mix of HTML, `tm` and `dtm`. The conversion to HTML uses `tth`.

10.8 HTML errors

HTML errors are fixed whenever possible, to make the resulting documents valid. Here is a list of errors that are fixed:

- inline elements containing block elements.
- `font` and `u` elements are replaced by `span` with CSS, or with CSS in the enclosing element.
- missing cells are added in tables.
- lists are fixed (things like `ul/ul`).
- `image` elements are replaced by `img`.
- deprecated style attributes are replaced by CSS (there are lots of them).
- `<center>` is replaced by a `div` or CSS, depending on the context.
- `<nobr>` is replaced by a `span` with CSS.
- `p` elements are added where needed, replacing `br` elements.
- empty style elements are removed.

10.9 LON-CAPA elements

- `responseparam` is renamed `parameter`.
- `<script type="loncapa/perl">` is replaced by `<perl>`.
- `<notsolved>` tags are removed in the case `<hintgroup showoncorrect="no"><notsolved>...</notsolved></hintgroup>` and in the case `<notsolved><hintgroup showoncorrect="no">...</hintgroup></notsolved>`.
- `<part>` elements are added so that responses are always inside a part. Warnings are issued when documents cannot be fixed.
- Hints are changed according to the new schema. For instance,

  ```
  <numericalresponse>
    <hintgroup>text1<numericalhint name="c"/><hintpart on="c">text2</hintpart></hintgroup>
  </numericalresponse>
  ```

  is replaced by

  ```
  <numericalresponse>
    <numericalhintcondition name="c"/>
  </numericalresponse>
  <hint>text1</hint><hint on="c">text2</hint>
  ```

  but `hintgroup` is kept when it is useful. Also `"hint"` elements (that should not exists) are replaced by their content, and `on="default"` is replaced by `default="yes"`.
- A unique `id` is added to problems and parts that don't have one.
- `conceptgroup/@concept` is renamed `display` and a new `id` attribute is created.
- Whitespace is removed inside elements with an empty content model.
- All attributes are turned into lowercase.

- Some spelling mistakes are fixed for `numericalresponse/@unit`.
- `&format` and `&prettyprint` are replaced by `<num>` whenever possible.
- `&chemparse` is replaced by `<chem>`.
- If the function call is enclosed in `<display>`, the `<display>` element is removed.

10.10 Pretty-print

After conversion, the document is rewritten in a nice, easy to read way.

Chapter 11

XML Parser

11.1 Invocation

The XML parser is at the heart of much of LON-CAPA's operation. As opposed to the original LON-CAPA, it does not distinguish between problems and non-problems, and it also serves system pages. It is called via lc.conf:

```
<LocationMatch "\.(xml|html|htm|xhtml|xhtm)$">
SetHandler perl-script
PerlHandler Apache::lc_asset_xml
</LocationMatch>
```

The main handler is thus lc_asset_xml. All other modules need to register their tags with this parser.

The XML parser actually since the early days of LON-CAPA used an HTML parser,

```
use HTML::TokeParser();
```

so it is more robust against trying to make sense of malformed XML, however, no malformed document should be stored. As this is the same parser mechanism as is used in LON-CAPA 2.x, porting of the extensive homework functionality will be facilitated. Documentation for this module is online.

11.2 Tag types

Tags are defined via subroutines. The name of these subroutines has three parts: start or end, tagname, and target. For example, here is lc_xml_localize, which defines the localize-tag:

```
package Apache::lc_xml_localize;

use strict;
use Apache::lc_ui_localize;

our @ISA = qw(Exporter);

# Export all tags that this module defines in the list below
our @EXPORT = qw(start_localize_html start_localize_tex);

sub start_localize_html {
   my ($p,$safe,$stack,$token)=@_;
   my $text=$p->get_text('/localize');
   $p->get_token;
   pop(@{$stack->{'tags'}});
   return &mt($text,
              split(/\s*\,\s*/,&Apache::lc_asset_safeeval::texteval($safe,$token->[2]->{'parameters'})));
}

sub start_localize_tex {
   my ($p,$safe,$stack,$token)=@_;
```

```
   return &mt($p->get_text('/localize'),
              split(/\s*\,\s*/,&Apache::lc_asset_safeeval::texteval($safe,$token->[2]->{'parameters'})));
}

1;
__END__
```

Target html is for web, target tex for printing. Defined tags need to be exported. In lc_asset_xml, lc_xml_localize needs to be used (without parenthesis).

If a tag is not defined, it is just printed to the screen on the web and ignored in print.

11.3 Stacks and safeeval

The stack is used to store the arguments of tags, so they can be retrieved later or inside of nested tags. The safeeval-space is a Perl safespace in which calculations and storage can happen (same as in the old LON-CAPA).

Chapter 12

Interface Programming

12.1 Interface Pages

12.1.1 General Mechanism

Interface pages (as opposed to asset pages) in LON-CAPA are typically controlled by an HTML, a JavaScript, and a Perl script file located in the `loncapa/app` directory.

- `loncapa/app/html/pages` and `loncapa/app/html/modals` contain the HTML/XML framework which is displayed to the user. The page contains the general layout, and it combines HTML-tags with LON-CAPA XML-tags. As this page is processed by the parser (see Chapter 11 on page 35), the LON-CAPA tags are replaced by dynamically generated HTML on the way to the browser.
- The HTML file will usually load JavaScript files located in `loncapa/app/scripts` to control client-side functionality.
- As the scripts make AJAX requests to update parts of the page, server-side functionality is provided by Perl modules located in `loncapa/app/handlers`. These modules often return simple text or JSON, which is processed client-side and injected into the page.

In rare cases, XML-tags establish areas which are not updated by client-side JavaScript, but which require server-side functionality. Usually, this is the case when server-provided data is not structured enough to map well into JSON, e.g., when targeted follow-up questions in various formats need to be generated on the fly. The handlers for this server-side generated HTML, which is then directly injected into the page, are located in `loncapa/xml/xml_includes`.

12.1.2 Examples

Let's walk through a couple examples to see how these different files are used. We begin at the homepage of the site which loads `index.html`. This sets the general structure of the page and defines space for things like the page header and the menu. It also loads the JavaScript file, `lc_default.js`, which includes common JavaScript functions. One such function that is called upon loading is `menubar()`. This function dynamically generates the menu based on the context. It communicates with the server through a JSON call, `$.getJSON("menu",...)`. `lc.conf` directs all "menu" requests to the Perl handler, `lc_ui_menu.pm`. The handler returns a JSON object containing the appropriate menu data.

Another function in `lc_defaults.js` is `display_asset(newuri)` which replaces the main iframe with the contents of the passed uri. `lc_default.js` also calls the JavaScript function, `dashboard()`, which loads the Dashboard. It uses `display_asset()` to load `lc_dashboard.html` into the iframe.

Portfolio

Now let's see what happens when we go to the Portfolio space. Clicking on the menu option, Portfolio, calls the JavaScript function, portfolio(), which uses `display_asset()` to load `lc_portfolio.html` into the iframe. `lc_portfolio.html` loads `lc_portfolio.js`, containing additional JavaScript functions needed in the portfolio space. `lc_portfolio.html` also creates the datatable with the following code:

lc_portfolio.html

```
<lcform id="portfolio" screendefaults="portfolio">
  <lcformtable id="paths">
    <span class="lcerror"><localize>A problem occured, please try again later.
```

```
        </localize></span>
    <lcformtableinput description="Path" id="pathrow" type="portfoliopath" />
  </lcformtable>
  <lcfileupload id="newfile" target='/upload_file' successcall="uploadsuccess"
      failurecall="uploadfailure" />
  <lcdatatable id="portfoliolist" class="portfoliomanager" pathfield="path" />
</lcform>
```

Upon loading, `lc_portfolio.js` calls `init_datatable()` which populates the user's portfolio. It does so with the following command:

lc_portfolio.js

```
$('#portfoliolist').dataTable( {"sAjaxSource" : '/portfolio?'+$('#portfolio').serialize()+
    '&command=listdirectory&showhidden='+showhidden+'&noCache='+noCache,...
```

The beginning of this statement, `$('#portfoliolist').dataTable(`, selects the table with id "portfoliolist" defined in `lc_portfolio.html` and populates it with the results of the `dataTable()` call. The first argument of `dataTable()`, `sAjaxSource`, contains the URL from which the data is pulled. `lc.conf` directs this request to the handlers `lc_auth_acc.pm` and `lc_ui_portfolio.pm`.

lc.conf

```
<Location /portfolio>
SetHandler perl-script
PerlAccessHandler Apache::lc_auth_acc
PerlHandler Apache::lc_ui_portfolio
</Location>
```

`lc_auth_acc.pm` passes the query string from `sAjaxSource` along to `lc_entity_sessions.pm`.

lc_auth_acc.pm

```
sub handler {
  my $r = shift;
  return &get_session($r);
}

sub get_session {
  my $r = shift;
  ...
  &Apache::lc_entity_sessions::get_posted_content($r);
  ...
}
```

lc_entity_sessions.pm

```
...
use vars qw($lc_session);

sub posted_content {
  return %{$lc_session->{'content'}};
}

sub get_posted_content {
  my ($r)=@_;
  my $query = new CGI($r);
  my %content=$query->Vars;
  ...
  $lc_session->{'content'}=\%content;
}
```

`get_session()` in `lc_auth_acc.pm` calls `get_posted_content()` in `lc_entity_sessions.pm` which uses CGI to load the query string into the global hash variable, `$lc_session`. `lc_ui_portfolio.pm`, called by `lc.conf` does most of the work of populating the portfolio.

lc_ui_portfolio.pm

```
sub handler
  my %content=&Apache::lc_entity_sessions::posted_content();
  ...
  if ($content{'command'} eq 'listdirectory') {
    # Do a directory listing
    $r->content_type('application/json; charset=utf-8');
    $r->print(&listdirectory($path,$content{'showhidden'}));

sub listdirectory
  ...
  my $output;
  $output->{'aaData'}=[];
  ...
  my $dir_list=&Apache::lc_entity_urls::full_dir_list($path);
  foreach my $file (@{$dir_list}) {
    ...
    push(@{$output->{'aaData'}},
    ...
  }
  return &Apache::lc_json_utils::perl_to_json($output);
```

This handler calls `posted_content()` from `lc_entity_sessions.pm` to retrieve the contents of `$lc_session`. Recall the `command=listdirectory` part of the `sAjaxSource` query. `lc_ui_portfolio.pm` checks for this and then proceeds to call `listdirectory()`.

`listdirectory()` creates a hash reference called `$output` to hold the contents of the portfolio. The `'aaData'` entry is defined as an empty list. The metadata for each portfolio element is pushed to this list. Finally, `perl_to_json()` is used to convert the list containing hash reference, `$output`, to a string in proper JSON format. The perl module `JSON::DWIW` ("Does what I want") handles the conversion.

The JSON object is returned as the arguement of `$('#portfoliolist').dataTable()` in `lc_ui_portfolio.js` which originally made the request. The portfolio table is populated and displayed.

Upload content

From the portfolio page, new content can be added by clicking the "Upload file" button. Generating the upload button involves the XML parsing described in Chapter 11. `lc_portfolio.html` contains the line:

lc_portfolio.html

```
<lcfileupload id="newfile" target='/upload_file' successcall="uploadsuccess"
    failurecall="uploadfailure" />
```

`lcfileupload` is a custom tag that must be interpreted. `lc.conf` parses HTML files with `lc_asset_xml.pm`.

lc.conf

```
<LocationMatch "(?i)\.(xml|html|htm|xhtml|xhtm|problem)$">
SetHandler perl-script
PerlHandler Apache::lc_asset_xml
</LocationMatch>
```

lc_asset_xml.pm

```
sub handler {
  my $r = shift;
  my $fn=$r->filename();
  ...
    $r->print((&target_render($fn,['html'],{}))[0]);
  ...
}

sub target_render {
  my ($fn,$targets,$stack,$content,$context,$outputid)=@_;
  ...
  my $p=HTML::TokeParser->new($fn);
  ...
  my $output=&parser($p,$safe,$stack,$status,$targets->[-1]);
  return ($output,$stack);
}
```

The main handler passes the filename to `&target_render()` along with the target, 'html'. `&target_render()` creates a new parser object for `lc_portfolio.html` and passes it to `&parser()`.

lc_asset_xml.pm (cont.)

```
sub parser {
  my ($p,$safe,$stack,$status,$target)=@_;
  ...
  while (my $token = $p->get_token) {
    ...
    } elsif ($token->[0] eq 'S') {
      # A start tag - evaluate the attributes in here
      ...
      $tmpout=&process_tag('start',$token->[1],$p,$safe,$stack,$status,$target,$token);
      ...
      $output.=$tmpout
  ...
  return $output;
}

sub process_tag {
  my ($type,$tag,$p,$safe,$stack,$status,$target,$token)=@_;
  ...
  my $outtag=$type.'_'.$tag.'_'.$target;
  ...
  }
  ...
  if (defined(&$outtag)) {
    $tag_output.=&{$outtag}($p,$safe,$stack,$token);
  ...
}
```

The `while` loop processes each line. The `<lcfileupload/>` statement is considered a start tag and is passed to `&process_tag()`. `&process_tag()` calls `&start_lcfileupload_html()` imported from `lc_xml_forms.pm`.

lc_xml_forms.pm

```
sub start_lcfileupload_html {
  my ($p,$safe,$stack,$token)=@_;
  my $id=$token->[2]->{'id'};
  my $name=$token->[2]->{'name'};
  my $target=$token->[2]->{'target'};
  my $description=$token->[2]->{'description'};
```

```
  my $success=$token->[2]->{'successcall'};
  my $fail=$token->[2]->{'failurecall'};
  unless ($name) { $name=$id; }
  unless ($description) { $description="Upload file"; }
  my $output='<label class="lcfileuploadlabel" for="'.$id.'" id="'.$id.'label">'
      .&mt($description).'</label>';
  $output.='<input id="'.$id.'" name="'.$name.'" class="lcinnerfileupload"
      type="file" onChange="do_upload(this.form,event,'."'$target','$id',
      '$success','$fail'".')" />';
  $output.=&hidden_field($id.'_path','');
  return $output;
}
```

The following HTML is generated and ultimately passed to the client:

```
<label id="newfilelabel" class="lcfileuploadlabel" for="newfile">Upload file</label>
<input id="newfile" class="lcinnerfileupload" type="file"
    onchange="do_upload(this.form,event,'/upload_file','newfile',
    'uploadsuccess','uploadfailure')" name="newfile">
</input>
```

Many of the menu options follow this pattern. Menu options execute JavaScript functions which replace the contents of the iframe through a call to `display_asset()`. The important contents of the page can be quickly changed without having to rebuild the entire page. The replacement HTML can load a JavaScript file containing any additional functions needed in that setting. Those JavaScript files can communicate with the server through JSON requests. `lc.conf` directs the JSON request to a specific Perl handler depending on the given path.

As one more example, let's see what happens when we go to Preferences. Clicking Preferences from the menu calls the JavaScript function `preferences()` in `lc_defaults.js`. This in turn calls `display_asset("/pages/lc_course_preferences.html")` to replace the iframe. `lc_course_preferences.html` loads `loncapa/app/scripts/lc_preferences.js` The only server communication occurs when the user clicks the "Store" button. Since data is being passed *to* the server opposed to just being retrieved, an AJAX POST request is used instead of the GET request (`$.getJSON`). The options set in the preferences form are passed with

```
$.ajax({url:  "/preferences", type:  "POST", ...})
```

`lc.conf` directs "preferences" calls to `loncapa/app/handlers/lc_ui_preferences.pm`.

12.1.3 Where should I be working?

If you are looking to change the structure or appearance of a page, then you will probably want to edit its associated HTML file located in `loncapa/app/html/pages`. Adding or changing client-side functionality will require editing the page's JavaScript file located in `loncapa/app/scripts`.

Server-side functionality is again controlled by Perl handlers located in `loncapa/app/handlers`. Data is passed to and retrieved from the server through AJAX/JSON requests that are placed in the page's JavaScript file. Perl handlers operate at a low level and should not need to be modified very often. There is probably already a Perl handler that will meet your needs.

Chapter 13

Internationalization

13.1 Encoding

Unicode UTF-8 is used throughout. All assets, program code, translation files, stored data, etc, is stored in utf-8 in order to avoid encoding problems down the road.

13.2 Maketext

LON-CAPA uses Maketext for internationalization, both inside of documents and called programatically. For each language, a translation file needs to be written. The format is simple:

```
package Apache::lc_localize::de;
use base qw(Apache::lc_localize);
use utf8;

%Lexicon=('_AUTO' => 1,
'language_code'      => 'de',
'language_direction' => 'ltr',
'language_description' => 'Deutsch',
'date_locale'  => '$weekday, $day. $month $year, $twentyfour:$minutes:$seconds Uhr',
'date_short_locale' => '$day.$month.$year',
'date_months'  => 'Jan.,Feb.,März,April,Mai,Juni,Juli,Aug.,Sep.,Okt.,Nov.,Dez.',
'date_days'    => 'So.,Mo.,Di.,Mi.,Do.,Fr.,Sa.',
'date_am' => 'vormittags',
'date_pm' => 'nachmittags',
'date_format' => '24',

'superuser' => 'Superuser',
'domain_coordinator' => 'Domänenkoordinator',
'course_coordinator' => 'Kurskoordinator',
'instructor' => 'Dozent',
'teaching_assistant' => 'Tutor',
'student' => 'Studierender',
'community_organizer' => 'Gemeinschaftsorganisator',
'member' => 'Mitglied',
'author' => 'Autor',
'co_author' => 'Co-Autor',

'Modify Selected Entries' => 'Ausgewählte Einträge ändern',

'Add New Entry' => 'Neuen Eintrag hinzufügen',

"Showing [_1] to [_2] of [_3] entries" => "Einträge [_1] bis [_2] von [_3] insgesamt",

...
```

```
);

1;
__END__
```

The files need to be utf-8-encoded, and we need to make sure that Perl knows this by specifying "use utf8."

Some configuration entries such as date_locale are mandatory. Where translations are missing, English is used. The [_n] are placeholders for variables.

13.3 Invocation in documents

In documents, phrases can be translated using the <localize>-tag, e.g.

```
<span class="lcsuccess"><localize>Your preferences were saved.</localize></span>
```

Placeholders can be defined using the parameter argument, e.g., <localize parameters="42,'Fred'">. This tag is defined in module lc_xml_localize.

13.4 Invocation from Perl

Internationalization is available from Perl via the %mt()-function. The first argument is the text, any following arguments come from the remaining arguments, e.g.,

```
&mt("Showing [_1] to [_2] of [_3] entries",$start,$finish,$total)
```

13.5 Internal codes

Translation should happen as close to the interface level as possible. If server-side functionality needs to store or generate things that need to be used both by humans and by other application components, those should not be translated too early. A good way are internal codes, and an example are roles. They are stored by codes and translated on the way out.

The file lc_localize.pm has such Code to English "translations," e.g.,

```
package Apache::lc_localize::en;
use base qw(Apache::lc_localize);
%Lexicon=('_AUTO' => 1,
'language_code'      => 'en',
'language_direction' => 'ltr',
'language_description' => 'English',
'date_short_locale' => '$month/$day/$year',

'superuser' => 'Superuser',
'domain_coordinator' => 'Domain Coordinator',
'course_coordinator' => 'Course Coordinator',
'instructor' => 'Instructor',
'teaching_assistant' => 'Teaching Assistant',
'student' => 'Student',
'community_organizer' => 'Community Organizer',
'member' => 'Member',
'author' => 'Author',
'co_author' => 'Co-Author'
);
```

13.6 Special cases

13.6.1 JavaScript configuration files

Some Javascript utilities expect a configuration file. This can be generated dynamically like for example in lc_ui_datatable_i14n, which is invoked by lc.conf:

```
sub handler {
# Get request object
   my $r = shift;
   $r->content_type('application/json; charset=utf-8');
   my $items={
    "sEmptyTable"    => &mt("No data available in table"),
    "sInfo"          => &mt("Showing [_1] to [_2] of [_3] entries",'_START_','_END_','_TOTAL_'),
    "sInfoEmpty"     => &mt("Showing 0 to 0 of 0 entries"),
    "sInfoFiltered"  => &mt("(filtered from [_1] total entries)",'_MAX_'),
    "sInfoPostFix"   => "",
    "sInfoThousands" => &mt(","),
    "sLengthMenu"    => &mt("Show [_1] entries",'_MENU_'),
    "sLoadingRecords"=> &mt("Loading..."),
    "sProcessing"    => &mt("Processing..."),
    "sSearch"        => &mt("Search:"),
    "sZeroRecords"   => &mt("No matching records found"),
    "oPaginate" => {
        "sFirst"    => &mt("First"),
        "sLast"     => &mt("Last"),
        "sNext"     => &mt("Next"),
        "sPrevious" => &mt("Previous")
                    },
    "oAria" => {
        "sSortAscending"  => &mt(": activate to sort column ascending"),
        "sSortDescending" => &mt(": activate to sort column descending")
              }
   };
   $r->print(&Apache::lc_json_utils::perl_to_json($items));
   return OK;
}
```

This corresponds to the code on the page,

```
$('#courselist').dataTable( {
      "bStateSave": true,
      "oLanguage" : {
         "sUrl" : "/datatable_i14n"
      },
      "aoColumns" : [
         { "bSortable": false },
         null,
         null,
         {"iDataSort": 4},
         {"bVisible": false}
      ]
    } );
```

which loads it.

13.6.2 Loading locatilization files

Some client-side JavaScript has its own localization files as part of the distibution. Here, the LON-CAPA localization file can be used to pick the right one. For example, the CKeditor gets its language from the HTML tag

```
<html lang="de" dir="ltr">
```

which is based on the German translation file

```
'language_code'      => 'de',
'language_direction' => 'ltr',
```

— but it could also take a separate language parameter, if ever these languages need to differ.

Chapter 14

Math editor, parser and equation syntax

14.1 Introduction

Historically, LON-CAPA has used several syntaxes for equations, such as Maxima and R. These syntaxes have evolved on the client side to be easier to write, in particular with implicit operators. Yet, many students have struggled to enter equations correctly. The consequences for incorrectly writing an equation vary depending on the context, from a simple admonition to a loss of points in an exam.

It is essentially a user interface problem, but the interpretation of a complex syntax can require a lot of code, and with implicit operators a correct interpretation actually depends on the context. For evaluations, LON-CAPA 2 uses some Perl transformations, Maxima, R, and some old C/lex&yacc code based on CAPA. It has become difficult to make this system evolve or even to maintain it.

To improve the user interface, limit the risk of misinterpretation and allow for future customization, the following decisions were made:

- The user interface needs a real-time feedback, so that users never send an equation with a syntax error. It should also help them to learn the syntax.
- Users should use a single, well-defined syntax (other syntaxes might be kept running for backward-compatibility).
- New server-side code should be written to replace everything that does not require a CAS. It should be flexible to allow for changes in the syntax, and easier to maintain.
- On the client and server sides, the equation parser has to understand mathematical expressions with units, variables and constants.

14.2 A new user interface

LON-CAPA 2 provides 2 user interfaces to write equations: one with a simple text field (users need to know the syntax), and one with a WYSIWYG Java applet. Due to new system and browser incompatibilities with Java, the applet does not work anymore in many environments.

A WYSIWYG editor is a popular demand today, but we have not found a libre, easy to use and reliable implementation in Javascript. Also, it might not be the best interface for end users: while it is nice to be able to input equations without having to know the syntax, using a WYSIWYG editor is not very easy for users who will have to enter many equations, and it hides some mathematical information. Writing a new WYSIWYG editor from scratch would be too long given our current developer team. This situation could change in the future, as a libre Javascript WYSIWYG editor will probably be created eventually.

An alternative today is to use a real-time preview for the equation, showing how the equation is interpreted as it is typed. The preview also needs to show the position of a syntax error. The text field + real-time preview solution helps to avoid sending an equation with a bad syntax, or an equation that would be misinterpreted. It also helps to learn operator precedence, as `1+2/3` will not show up like `(1+2)/3`.

This solution was chosen for the new user interface. The implementation uses MathJax to display equations, reducing the Javascript code to a parser exporting equations to MathML. As with LON-CAPA 2, equations may use implicit operators. Since it is not always possible to guess automatically if a name is representing a variable or a unit, two separate modes are used to interpret equations: *unit*, for expressions with units and constants, and *symbolic*, for expressions with variables and constants. A short list of constants is used to recognize them.

The editor can be tried from the git by opening the following file in a web browser (it does not require a LON-CAPA install):

`loncapa/app/scripts/maxima_editor/test.html`

14.2.1 Usage

In an HTML, script elements are used to reference the editor minimized version and MathJax. A new text area with automatic equation preview can then be inserted in the document with the math class:

```
<textarea class="math" data-implicit_operators="true" data-unit_mode="true"
    data-constants="c, pi, e, hbar, amu, G" spellcheck="false" autofocus="autofocus"></textarea>
```

14.3 The server-side parser

The server-side parser is similar to the one written in Javascript, and in fact is a direct translation of it in Perl to make it easier to maintain. The MathML export is replaced by a calculation code and CAS exports. It also uses the unit and symbolic modes. The parser is located in the git at `math/math_parser`. Two test scripts can be used to try it, make sure it works well, and learn usage: `test/math_parser_test_cases.pl` and `test/math_parser_manual_test.pl` (they require a LON-CAPA install to work).

14.3.1 Usage

The parser constructor takes 2 parameters: `implicit_operators`, a boolean (1 or 0) that tells if implicit operators are allowed, and `unit_mode`, another boolean to specify the mode (unit or symbolic). The parser's parse takes an equation string as a parameter, and returns an `ENode` representing a parsed tree for the equation. This `ENode` object can then be used for conversion or calculation.

14.3.2 Calculation environment

An object of the class `CalcEnv` is used to specify the mode (unit or symbolic), additional units, and variable values. It is passed to the `calc` method of `ENode` for calculation.

14.3.3 Conversion to Maxima syntax

An `ENode` can be used for a Maxima syntax export, with the `toMaxima()` method.

14.3.4 Catching errors

Exceptions are thrown whenever a problem occurs. `ParseException` and `CalcException` can be caught with try/catch, and it is possible to get the raw error message or a localized one.

14.3.5 Example with unit node

```
my $implicit_operators = 1;
my $unit_mode = 1;
my $equation = "4 peck + 2 bushel";
my $p = Parser->new($implicit_operators, $unit_mode);
my $root = $p->parse($equation);
my $env = CalcEnv->new($unit_mode);
$env->setUnit("peck", "2 gallon");
$env->setUnit("bushel", "8 gallon");
$env->setUnit("gallon", "4.4 L");
print "Maxima syntax: ".$root->toMaxima()."\n";
print "Value: ".$root->calc($env)->toString()."\n";
```

14.3.6 Example with symbolic node

```
my $implicit_operators = 1;
my $unit_mode = 0;
my $equation = "1/(x-y-z)";
my $p = Parser->new($implicit_operators, $unit_mode);
my $root = $p->parse($equation);
my $env = CalcEnv->new($unit_mode);
$env->setVariable("x", 1/2);
```

```
    $env->setVariable("y", 1/3);
    $env->setVariable("z", 1/7);
    print "Maxima syntax: ".$root->toMaxima()."\n";
    print "Value: ".$root->calc($env)->toString()."\n";
```

Note that these examples are using aliases to avoid giving the full package name for classes:

```
use aliased 'Apache::math::math_parser::CalcEnv';
use aliased 'Apache::math::math_parser::Parser';
```

14.3.7 Comparing results of calculations

The `calc` method of `ENode` returns an object of one of these classes: Quantity, QVector, QMatrix, QSet, QInterval or QIntervalUnion. The type of the result can be checked with the `isa` method, for instance:

```
    my $result = $root->calc($env);
    if ($result->isa(QInterval)) {
        print "This is an interval.\n";
    }
```

All the returned objects have the `equals` method, which returns 1 if the two objects are equal, 0 in all other cases (including when the type is different). This can be used to check if a result matches some expectation. For instance:

```
    my $result = $p->parse("1+2")->calc($env);
    my $expected = $p->parse("3")->calc($env);
    if (!$expected->equals($result)) {
        print "This is not the expected result.\n";
    }
```

To compare the result more precisely with the expected result, the `compare` method can be used. It is also implemented in all returned classes. It returns a code that explains why the two objects are different. The returned code is enumerated in `Quantity` : `IDENTICAL`, `WRONG_TYPE`, `WRONG_DIMENSIONS`, `MISSING_UNITS`, `ADDED_UNITS`, `WRONG_UNITS`, `WRONG_VALUE`, `WRONG_ENDPOINT`.

```
    my $result = $p->parse("1+2")->calc($env);
    my $expected = $p->parse("3")->calc($env);
    my $code = $expected->compare($result, $tolerance);
    if ($code != Quantity->IDENTICAL) {
        print "This is not the expected result.\n";
    }
```

14.3.8 API

The API can be useful to do more operations with the results. Constructors for the various calculation result classes can also be used to create these objects without the parser and do further calculations. The HTML documentation can be created automatically with the lcdoc.pl Perl script, in the git:

```
cd loncapa/docs
mkdir -p code/lcdoc/math/math_parser
bin/lcdoc.pl -d code/lcdoc/math/math_parser ../math/math_parser/*.pm
```

14.4 Equation syntax

14.4.1 Spaces

Spaces are always ignored.

14.4.2 Decimal separators and function parameter separators

By default, "," and "." can be used as decimal separators. ";" is used to separate function and vector/matrix parameters.
Variables Variable names are used directly, without any special character before.

14.4.3 Constants

Constant names are used directly. LON-CAPA has a list of known constants.

14.4.4 Units

Unit names are used directly. LON-CAPA has a list of known units.

14.4.5 Parenthesis

Parenthesis can be used to specify evaluation order.

14.4.6 Complex numbers

Complex numbers are understood, and will be used in calculations. This means that `"i"` should never be used as a variable.

14.4.7 Operators

- arithmetic: `+ - * / ^`
  (for vectors and matrices, `*` is an element-by-element multiplication)
- factorial: `!`
- relational: `= # < <= >= >`
- percent of a constant: `%`
  Example: `2%c = 2/100*c`
- units: `‘`
  Example: `2‘m + 3‘m = 5‘m`
- scalar product for vectors, multiplication for matrices: `.`

14.4.8 Implicit operators

`*` and `‘` are implicit in LON-CAPA. The parser will try to guess which operator is missing whenever possible. The choice between `*` and `‘` depends on the mode for interpreting equations.
Example: `2c+3m/s` is understood in unit mode to be `2*c + 3‘(m/s)`. In symbolic mode, it would be interpreted `2*c + (3*m)/s` (`m` and `s` being variables).

14.4.9 Functions

Functions use the syntax `f(a;b)`.

- basic:

```
pow(x;y)=x^y, sqrt(x), abs(x), exp(x)=e^x, factorial(x)=x!,
ln(x), log(x)=ln(x), log10(x)=ln(x)/ln(10)
```

- less common functions:

```
mod(x;y) (modulo), sgn(x) (sign of x), ceil(x) (ceiling), floor(x),
binomial(n;p)=n!/(p!*(n-p)!);
```

- requiring symbolic mode:

```
sum(f(x);x;x1;x2), product(f(x);x;x1;x2)
```

  "i" cannot be used as a variable name because it can be confused with the imaginary number.
  Example: `sum(a^2; a; 1; 5)`

- requiring a CAS like Maxima for calculation:

```
diff(expr; x; n), integrate(expr; x; a; b), limit(expr; x; val; dir)
```

The syntax is the same as Maxima.

- trigonometry:

```
sin(x), cos(x), tan(x), asin(x), acos(x), atan(x), atan2(x;y),
sinh(x), cosh(x), tanh(x), asinh(x), acosh(x), atanh(x)
```

14.4.10 Vectors and matrices

Vectors and matrices are defined with square brackets. A matrix is made of a list of row vectors.

- vectors: `[1;2;3]`
- matrices: `[[1;2];[3,4;5,6]]` (remember the comma is a decimal separator)

14.4.11 Sets

Sets are defined with curly brackets: `{1;2;3}`
Possible operations:

- union: `{1;2}+{1;3} = union({1;2};{1;3}) = {1;2;3}`
- intersection: `intersection({1;2};{1;3}) = {1}`

14.4.12 Intervals

Intervals are defined with square brackets and parenthesis, and colon as a separator.

- closed interval: `[1:2]`
- open interval: `(1:2)`
- left-open interval: `(1:2]`
- right-open interval: `[1:2)`

As with sets, the `+` operator and the `union` and `intersection` functions can be used.

Chapter 15

Graphical editor

15.1 Introduction

One of the goals of the new version is to make it easier for new users to start with LON-CAPA. An important task for new users is to create new content. The current graphical editor, called "colorful editor", has one major issue: it requires sending the document to the server with a button for each change in the structure. It is also unable to handle copy/paste actions. This has led many users to edit documents with the source editor instead.

The goal of the new editor is to be user-friendly, which implies that user interactions are dynamic and well integrated with the browser. To reach this goal, the code for the editor has to be moved to the client-side, which requires a rewrite in a different programming language. Also, edition of mixed LON-CAPA and HTML elements should be seamless.

A popular request is a WYSIWYG editor. LON-CAPA elements can be described as high-level building blocks for a program, which even include Perl scripts, and as such cannot be edited in a WYSIWYG way. HTML elements, used for static content, can on the other hand be edited in a WYSIWYG way, but this can lower the quality of the documents served to students, make it difficult to update the content without dealing with the presentation, and prevent updates of the language. The chosen balance for the new graphical editor is to display most LON-CAPA elements in a WYSIWYM way (as was done in the colorful editor), but use more WYSIWYG displays with HTML elements, while still trying to promote semantic editing when it is not cumbersome for authors. An example of the balance is the use of WYSIWYG styles like bold and italic, with the absence of styles like underline and font size in the toolbar, and the addition of HTML5 section elements to attach semantics to blocks of text.

15.2 Choice of tools

After a general search for tools to create the new editor, two GPL-compatible tools were fully evaluated: CKEditor and Daxe.

CKEditor is a light-weight WYSIWYG HTML editor written in Javascript which is easily extensible. The edition is based on the contenteditable attribute, which means that the browser handles a lot of the editing process. The toolbar can be extended with new buttons, and a little library makes it easy to create new dialogs to insert elements. Widgets were recently introduced to create non-HTML element displays.

Daxe is also a light-weight graphical editor, but it can handle any XML document (not just HTML). It is meant to be configured for a given XML language with an XML schema, a configuration file, and code for element displays. The displays can be WYSIWYG, but most built-in element displays are WYSIWYM instead of WYSIWYG. Daxe is written in Dart converted automatically into Javascript (Daxe stands for DArt Xml Editor), and element displays are also written in Dart (although they can use Javascript if necessary).

Some of the criteria used to chose between these two tools were:

- Maintenance cost. A new developer has to be able to understand and extend or fix the code easily.
- Extensibility: how easy is it to add a new element, and new language constraints ?
- UI Customizability: it should be easy to change the user interface, from cursor position to global layout.
- Ability to handle WYSIWYG and WYSIWYM displays.
- Possibility to reuse some built-in displays.
- LON-CAPA developer experience with the tool.

To sum it up, Daxe wins (more or less) on all points. Dart is a better programming language for large projects, the fact that Daxe is basing the language constraints on an XML schema makes it easier to add new elements and other language constraints, Daxe (which is not using the contenteditable attribute) can allow precise UI changes at a low level, it has handled a mix of WYSIWYG and WYSIWYM displays from the start, and it was created by the author of these lines (Damien). CKEditor's only advantage is to already have more WYSIWYG displays for HTML elements.

15.3 Development

15.3.1 Tasks

Development for the new graphical editor can be described by the following tasks:

- Creation and Improvement of the XML schema, including reference documentation in English for each element.
- Creation of a Daxe configuration file for the LON-CAPA language, including localized documentation for each element.
- Improvement of Daxe, in particular for HTML WYSIWYG displays. These changes can benefit people outside of LON-CAPA.
- Development of a LON-CAPA extension of Daxe, with custom menus, toolbar items, and element displays.

A parallel task that is not strictly development of the editor is the conversion of current LON-CAPA documents into well-formed, mostly valid, XML documents.

15.3.2 Git paths

In the git, the schema for the current version of LON-CAPA is at `conversion/old_loncapa.xsd` (a caveat: LON-CAPA documents are currently not XML and cannot be validated). The new schema is at `conversion/loncapa.xsd`. Document conversion is done with `conversion/clean.bash`.

Daxe is located at `xml/editor/daxe`. The LON-CAPA web application is at `xml/editor/loncapa_daxe`. It links to Daxe with a relative path in `xml/editor/loncapa_daxe/pubspec.xml`. The schema it uses is at `xml/editor/loncapa_daxe/web/config/loncapa.xsd`, which should be the same as `conversion/loncapa.xsd`. The configuration file is at `xml/editor/loncapa_daxe/web/config/loncapa_config.xml`.

15.3.3 Edition and launch in Dart Editor

The Dart editor, while not necessary, is very useful for Dart development. This editor currently (August 2014) has two major bugs on Ubuntu/KUbuntu, so it is good to know the following workarounds:

- It links to libudev.so.0, so a symbolic link can help if a different version is installed:
  `sudo ln -s /lib/x86_64-linux-gnu/libudev.so.1 /lib/x86_64-linux-gnu/libudev.so.0`
- It crashes with KDE's (default) oxygen theme, so the following workaround is needed to start it when using oxygen:
  `GTK2_RC_FILES=/usr/share/themes/Raleigh/gtk-2.0/gtkrc DartEditor`

Other Linux distros are not supported. While it is possible to compile the Dart SDK, compiling Dartium is a lot harder and it is harder to debug without Dartium.

Once Dart Editor is started, the directories `xml/editor/daxe` and `xml/editor/loncapa_daxe` should be opened with the menu `File-Open existing Folder...`. They will reopen automatically the next time Dart Editor is opened.

The `Run-Manage Launches...` menu can be used to configure launches with Dartium and other browsers (Dartium is a version of Chromium with an enabled Dart VM). Dartium is good for development, but testing with other browsers (which requires an automatic conversion into Javascript) is also necessary. The following text should be used in the "HTML file" field: `/loncapa_daxe/web/loncapa_daxe.html?config=config/loncapa_config.xml` This will start the editor with an empty file. A file can also be edited by specifying its relative path (from the web directory) in the
URL: `/loncapa_daxe/web/loncapa_daxe.html?file=test.problem&config=config/loncapa_config.xml`

Note that there is a bug in the Dart SDK 1.5 that prevents the use of Daxe in Javascript (Dart Issue 19888). It is fixed in v1.6. Since early 2013 (before version 1), there has been some issues with the conversion from Dart to Javascript, but Google has been fast to fix them when they were reported at `https://code.google.com/p/dart/issues/list`.

15.3.4 Export to Javascript

Note that an export is not necessary to just test Daxe with Javascript (as explained above). To export the Javascript code and related files for use in LON-CAPA, select a LON-CAPA Daxe file (as opposed to Daxe) in the Dart editor, and select the menu `Tools - Pub Build (generates JS)`. A whole directory is built in `loncapa_daxe/build`. This does not create a directory with a minimal set of files. `loncapa_daxe/build.sh` and `install.sh` can be used for that.

Chapter 16

Glossary

Asset An HTML page, homework problem, image, movie, etc.

Author A user who has author privileges, i.e., who can publish assets

Cluster Group of machines linked into a network governed by a cluster table

Course An entity with members and a table of contents. Used as a general term to also include communities

Community Special kind of "course" without a gradebook, usually used for collaboration

Domain Logical segmentation of a LON-CAPA cluster, usually by institution

Entity Code Part of the unique identifier of assets, users, and courses, the other part being the domain

Group A special kind of section, mostly for teamwork. Students can be in more than one section at a time

Homeserver Node in a cluster which holds the authoritative and permanent copy of a user's or course's data and assets

Portfolio The space where users store their assets

Realm The extend of privileges, i.e., system-wide, domain-wide, course-wide, section-wide, or user-wide

Role Users can have a number of (time-limited) roles which each grant a certain set of privileges within a particular realm

Section Sub-group of a course, for example a lab section. A student can only be in one section at a time

WYSIWYG What You See Is What You Get, meant to look the same when edited and rendered (on the web in LON-CAPA's context)

WYSIWYM What You See Is What You Mean, displayed in the best way to convey the semantics

Index