Chair of
Software Engineering

TrucStudio
Truc Sharing

Master Thesis

By: Lukas Angerer
Supervised by: Michela Pedroni
Prof. Dr. Bertrand Meyer

Student Number: 03-906-591

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

inf | Informatik
Computer Science

TrucStudio - Truc Sharing

Lukas Angerer

February 24, 2008

Contents

1 Introduction **5**
- 1.1 Summary . . . 5
- 1.2 Structure of this document . . . 6

I User Manual **7**

2 Basic Truc Sharing **8**
- 2.1 Downloading existing content . . . 8
  - 2.1.1 Creating an account . . . 8
  - 2.1.2 Using the repository browser . . . 8
- 2.2 Uploading your own entities . . . 8
  - 2.2.1 Uploading changes and downloading newer versions . . . 10
  - 2.2.2 Merging . . . 10
- 2.3 Icons and their meaning . . . 11
  - 2.3.1 Mismatches . . . 11

3 Advanced Features **13**
- 3.1 Repository browser . . . 13
  - 3.1.1 Comparing different revisions . . . 13
  - 3.1.2 Comparing different entities . . . 13
- 3.2 Administrative features . . . 13
  - 3.2.1 Change password . . . 14
  - 3.2.2 Modify entity access . . . 14
  - 3.2.3 Modify privileges . . . 14
  - 3.2.4 Create project . . . 14
  - 3.2.5 Grant / revoke admin rights . . . 14
  - 3.2.6 Session management . . . 15
- 3.3 Configuration . . . 15

II Developer Manual **16**

4 Design **17**
- 4.1 Requirements . . . 17
  - 4.1.1 Collaborative environment . . . 17
  - 4.1.2 Truc sharing . . . 18
- 4.2 Design . . . 18

4.2.1 Data storage 18
4.2.2 Database 20
4.2.3 Communication 20

5 Implementation 22
5.1 Truc sharing 22
5.1.1 Local, working copy and remote entities 22
5.1.2 Revisions 22
5.1.3 Updating and merging 23
5.2 Model redesign 23
5.3 XML I/O 24
5.4 RPC Integration 24
5.4.1 Sending XML data 25
5.5 The entity cache 25
5.6 The diff-viewer 26
5.6.1 Property types 27
5.6.2 Property implementation 27
5.6.3 Diff-dialog 28

6 Services 29
6.1 Common 29
6.1.1 Request / response structure 29
6.1.2 Custom TrucStudio structs 31
6.2 Diagnostic service 33
6.2.1 diagnostic.echo 33
6.2.2 diagnostic.echo_reverse 34
6.2.3 diagnostic.version 35
6.3 Session service 36
6.3.1 session.start 36
6.4 User management service 38
6.4.1 user.create_user 38
6.4.2 user.create_group 39
6.4.3 user.change_password 40
6.4.4 user.change_admin 41
6.4.5 user.modify_user_privileges 42
6.4.6 user.modify_entity_access 44
6.5 Entity repository service 45
6.5.1 repository.list_entities 45
6.5.2 repository.show_log 47
6.5.3 repository.get_entities 48
6.5.4 repository.put_entities 50
6.5.5 repository.update_entities 52
6.5.6 repository.commit_entities 54

7 Eiffel-DOM 56
7.1 Overview 56
7.2 XPath functionality 56

8 Problems **58**
8.1 Bugs in libraries . . . 58
8.1.1 Date / time parsing . . . 58
8.2 Other inconveniences . . . 58
8.2.1 *EV_GRID* and Pick-And-Drop . . . 58
8.2.2 ODBC data sources on Windows / Linux . . . 59

9 Future Work & Conclusion **60**
9.1 Future work . . . 60
9.1.1 Improvements of the administrative features . . . 60
9.1.2 Repository browsing . . . 60
9.1.3 New server features . . . 60
9.1.4 Server as a windows service . . . 61
9.2 Conclusion . . . 61

III Compilation / Installation **62**

10 Compiling TrucStudio and the TrucStudio Server **63**
10.1 IDE & compiler . . . 63
10.2 Libraries . . . 63
10.2.1 Gobo . . . 63
10.2.2 Eposix . . . 64
10.2.3 ODBC (server only) . . . 64
10.2.4 Goanna & Log4E . . . 65
10.3 The source code . . . 65
10.3.1 src . . . 65

11 Installation **66**
11.1 Requirements . . . 66
11.1.1 Setting up the database . . . 66
11.2 **TrucStudio** . . . 68
11.3 **TrucStudio Server** . . . 68
11.3.1 Command line parameters . . . 68
11.3.2 Configuration . . . 68

Chapter 1

Introduction

1.1 Summary

TrucStudio [POM07] is a software tool for domain knowledge modeling and course management with the goal of assisting teachers and their staff in the planning and execution of courses. The core of **TrucStudio** are the concepts of cluster, Truc and notion as defined in [Mey06].

Creating a structured domain knowledge model is a collaborative process that usually requires several improvement cycles [POM+08]. The thesis at hand addresses this problem by introducing a "Truc server" where Trucs (as well as clusters, notions, courses and lectures) can be stored. The **TrucStudio Server** provides a versioning mechanism similar to those used to manage source code in software projects to ensure non-destructive, collaborative domain knowledge modeling. This versioning mechanism significantly reduces the effort needed for creating a new course by reusing existing Trucs.

Concurrent to the development of this project, there has been a master thesis that extended **TrucStudio** with an elaborate output generation system that allows the generation of several different output formats from within **TrucStudio** [Alb08]. The combination of the two projects can be used to facilitate the distribution of supporting course material and exchange **TrucStudio** data with non-**TrucStudio** users.

This documentation focuses on the changes and extensions that were made to the first version of **TrucStudio**. Some basic information about the "old" system may be helpful in understanding the changes that were made. The preceding **TrucStudio** documentation [Wid07], [Cro07] and a documentation Wiki that is currently under construction is available online under

`http://trucstudio.origo.ethz.ch/wiki/doc`

1.2 Structure of this document

The first part of this document contains the user manual for the **TrucStudio** Truc sharing extension. It provides an overview over the graphical user interface components that can be used to publish and download Trucs. Chapter 2 describes the basic concepts of Truc sharing and chapter 3 explains some more advanced features that may not be of interest to TrucStudio newcomers. The second part describes the design process (chapter 4) and the implementation details (chapters 5 to 8) of the project including the server API and it concludes with the discussion of possible future work. The third and last part gives a detailed technical description of how to compile and set up **TrucStudio** and the **TrucStudio Server** .

Part I

Truc Sharing User Manual

Chapter 2

Basic Truc Sharing

2.1 Downloading existing content

2.1.1 Creating an account

In order to connect to a **TrucStudio Server** you need a user account. An account can be created by selecting "TS Server ⇒ Create Account..." from the **TrucStudio** main menu. Each **TrucStudio Server** manages its own set of user accounts and projects, so if you would like to use multiple servers, you need to create an account on every server. Once an account is created, the login information can be used to start a session with the server and once a session is started, you can access the data on the server. The login window appears automatically as soon as you try to perform an operation that requires a server session.

2.1.2 Using the repository browser

Click on the **TrucStudio Server** icon in the main tool bar to switch to the repository browser view (Figure 2.1). To download an entity, you need to pick-and-drop the entity you want to download and select the "destination" in the cluster-truc-notion tree or the course-lecture tree. A Truc can only be downloaded into a cluster, a notion only into a Truc and so forth. When the download is completed, the entity is displayed in its target location.

2.2 Uploading your own entities

Sharing your own entities is just as simple as downloading existing ones. Select the entity you would like to share (one that has not been shared before) and right click it. In the context menu, select the "Share" menu entry. A dialog will pop up where you specify the destination server and project; this will become the permanent residence of this entity, so you might want to give that some thought. When the entity is uploaded, its icon will change, signaling the successful sharing process. Once an entity is uploaded, it cannot be uploaded again to a different server or a different project.

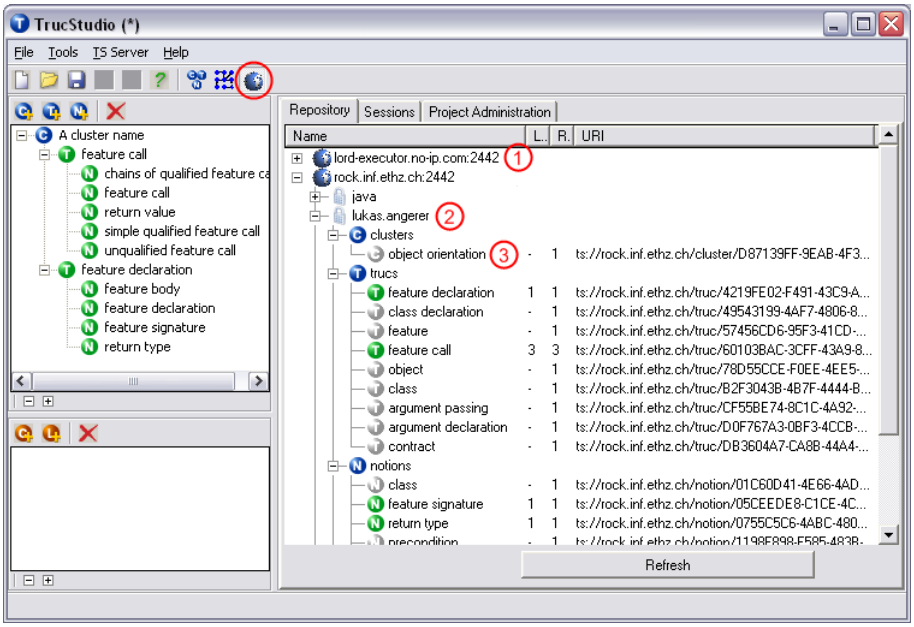

Figure 2.1: The repository browser

1. *Servers*:
   Expanding a server node will automatically prompt you with the login dialog if there isn't already an open session with the selected server. The list of servers contains all servers listed in the **TrucStudio** configuration file as well as all other servers with a currently open session.

2. *Projects*:
   For every server, there is a list of projects for which the user has read access to the associated entity store.

3. *Entities*:
   Every project is split into subsections containing the different entity types. The actual entities are marked with their respective entity type icon showing the current state of the entity. Right clicking on an entity will present you with a context specific menu containing the actions that can be performed on the entity depending on its state.

2.2.1 Uploading changes and downloading newer versions

Once changes have been made to a shared entity, you might want to upload these changes to the server. To do that, you have to right click on the entity in the tree on the left side or in the repository browser and select the "Upload" option. If nobody else has uploaded changes since your last download, the upload will most likely succeed and you can continue working on the entity. If the entity you are trying to upload has been modified by somebody else in the meantime, you will have to update your copy first (otherwise the upload will result in an error message). You can see whether there is an update available for an entity in the repository browser or you can simply update your entity and see what happens. Right click on a shared entity and select the "Download" menu entry. Depending on the state of the entity you have selected, this operation has different outcomes:

- If the selected entity has been modified and there is a newer version on the server, you will be presented with a difference view (or "diff-view" for short) of your local version and the version from the server. The diff-view will assist in the consolidation of the differences and facilitate the so-called merging process (see 2.2.2 for more information on the "diff-view" and the merging process). When the merging is completed, your local copy will be updated and any remaining modifications can be uploaded to the server.
- If no modifications have been made to the entity, the update will simply download the newest version from the server and replace the old one.
- When there is no newer version on the server, the download will not change anything. If the entity is marked as modified although there are no differences to the version from the server – which happens after manually changing an entity and then manually changing it back – the state of the entity will be reset to "not modified".

2.2.2 Merging

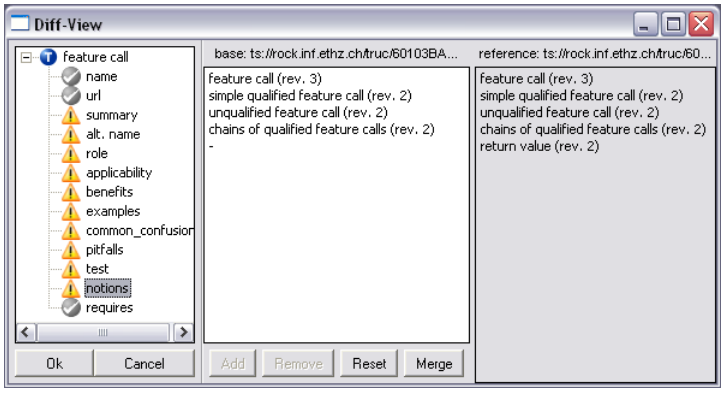

Figure 2.2: Diff-viewer showing two different versions of the Truc "feature call"

When a locally modified entity is updated (a newer version is downloaded from the server), the changes should obviously not be overwritten without the user's

consent. A diff-view (Figure 2.2) is presented that allows the comparison of the two different versions and can be used to "merge" them. In this context, merging means incorporating the changes from the newer version into the older (local) version or, depending on the point of view, applying the local modifications to the new version.
The diff-viewer shows a list of entities that have to be merged and for each of them a list of properties. Clicking on one of the properties will allow the user to edit the local version of the property on the left side while seeing the contents of the same property from the newer version on the right side. The changes made in the diff-viewer are only temporary and are only saved when the dialog is exited using the "Save" button.
Right clicking on an entity or property in the list of on the left shows a context menu with options to perform an "auto-merge" on the selected property/entity or all items in the list at once. The automated merging is only a guess of the program on what might be required to merge the properties, the results should be examined carefully before accepting the merge.

2.3 Icons and their meaning

There are several different color variations of the standard cluster, Truc, notion, course and lecture icons that signify different states of the entities. The following list is an explanation of the color codes using the cluster-series as an illustration.

 Local entity that has never been shared.

 Shared entity without local modifications.

 This shared entity has been modified since the last download.

 A "stub" entity that has not yet been downloaded (right-click & download will fix this).

 There is a mismatch between a local entity and a remote entity (they have the same UUID); see section 2.3.1 for more details.

 There is a newer version of this entity available on the server. [1]

 This entity has been modified and there is a newer version of the entity available on the server (downloading the new version will present the diff-viewer). [1]

 An entity that is present on the server and has not yet been downloaded. [1]

2.3.1 Mismatches

Mismatches can occur if **TrucStudio** accidentally generates a new UUID which has already been assigned to another shared entity. The probability of that happening are in the range of $1 : 2^{128}$, but in that case you should just create a

[1] This icon only appears in the repository browser

new entity, copy the data from the mismatched entity to the new one and delete the mismatched entity.
The other case in which a mismatch can occur is when local entities (from a `*.txm` file) are published and later on, the same local entities are loaded again. Obviously, the local entities will have the same UUIDs than their published counterparts and they will all be marked as mismatches in the repository browser. The solution for this sort of mismatch is to discard the old entities after publishing them; they are no longer necessary.

Chapter 3

Advanced Features

3.1 Repository browser

3.1.1 Comparing different revisions

In some cases it can be useful to see what has changed between different revisions of the same entity. In **TrucStudio** you can compare the working copy of an entity (the current local version) with any other revision of that entity. To do that, right click the entity in the repository browser and select the "Compare to revision..." from the context menu. After entering the desired revision number, **TrucStudio** opens the diff-viewer with the two revisions of the selected entity. The same process also works for entities that are not locally available at the moment. In that case **TrucStudio** will compare the newest version on the server with the selected version and the diff-view will not be editable since saving the changes is not possible.

3.1.2 Comparing different entities

You can also compare different entities with the diff-viewer by pick-and-dropping an entity in the repository browser on another entity in the repository browser. You have to make sure that the two entities are of the same type (two clusters, two Trucs, ...), otherwise you will receive an error message.

3.2 Administrative features

Administrative features can be found in the "Project Administration" tab of the online view (Figure 3.1). Every administrative task consists of three steps (as seen in Figure 3.1):

1. Select the server on which the operation should be carried out.
2. Select the task that should be performed.
3. Enter the task specific information and confirm the action.

Some of the tasks described below are only available to administrators and will not be visible to normal users.

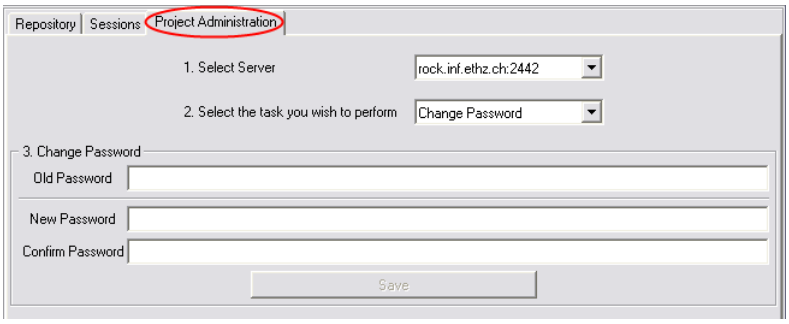

Figure 3.1: Administrative tab of the server view

3.2.1 Change password

To change the password for your user account, you need to provide the current password, the new password and click on the "Save" button. The next time you start a new session, you have to enter the new password – the current session will continue.

3.2.2 Modify entity access

The process of changing the entity availability for a project is straight forward: select the project, set the new access levels and press "Save". The changes will take effect immediately.

3.2.3 Modify privileges

To promote/demote a user within a project, you need to be an owner of that project (or an admin). The "Project" combo box contains a list of all projects for which the current user is an owner and after selecting one of the projects, the "User" combo box will contain the complete list of members and owners of that project. Changes made in this section only take effect when a new session is started.
Admins may also promote/demote users if they are not owners of the corresponding projects, but the project and user names have to be entered by hand.

3.2.4 Create project

Note: This option is only available to admins.
Enter the name of the new project and click on the "Create" button. If the project name does not conflict with an already existing project, the new project is created.
To access the new project, a new session has to be started.

3.2.5 Grant / revoke admin rights

Note: This option is only available to admins.
Admins may promote other users to admins and demote other admins to regular users by entering their username and clicking one of the "Make Admin" or

"Revoke Admin" buttons. It is not possible for an admin to revoke his/her own admin rights.

3.2.6 Session management

The XML-RPC protocol is stateless and the connection to the server is closed after every request, that's why **TrucStudio** uses sessions to authenticate a user for a predetermined duration. Most of the time the session management will not be noticed by the user, as it is designed to work behind the scenes, but in some cases the user will have to start a new session manually. For example, when the user is added to a new project, the changes will not be visible to the affected user until the next login.
The "Sessions" tab in the server view contains a list of all currently active sessions. Selecting one of the sessions and clicking the "Close Session" button will abort the session and the next time a request is sent to that server, the user is asked to log in again.

3.3 Configuration

Some **TrucStudio** settings are loaded from a configuration file at startup. The file is located under `config/config.xml` relative to the executable path. The configuration is stored as an XML file and – as long as the required structure is maintained – can be modified with a normal text editor (e.g. "Notepad" under Windows). The section that is relevant for entity sharing by default looks like this:

```
<settings>
  ...
  <ts_server>
    <host port="2442">rock.inf.ethz.ch</host>
  </ts_server>
</settings>
```

The `ts_server` element contains a list of **TrucStudio** servers that will be available in the repository browser on startup. Adding an additional server to this list is a simple matter of creating a new `host` element (with a `port` attribute) containing the hostname of the server. Additionally, it is possible to specify a default username for every server. For example, after adding the (imaginary) host "trucserver.org" which listens on the default **TrucStudio Server** port 2442 with the default login "john.doe", the configuration would look like this:

```
...
<ts_server>
  <host port="2442">rock.inf.ethz.ch</host>
  <host port="2442" login="john.doe">trucserver.org</host>
</ts_server>
...
```

Part II

Truc Sharing Developer Manual

Chapter 4

Design

4.1 Requirements

4.1.1 Collaborative environment

The main idea of the project is to extend the **TrucStudio** application with collaborative functionality so that the people working with **TrucStudio** can exchange data without having to switch to another application. To allow different users in different locations to access and possibly modify shared data, some basic features are required:

Central data repository

Since not every user can be expected to be online all the time, a remotely accessible server will be required that serves as a data repository and control authority. **TrucStudio** should be able to work with multiple servers at the same time (i.e. without having to restart).

User authorization

The system has to support different user groups with different access levels called "owner", "member" and "all". The information availability for these user groups can be adjusted by an owner of a project as well as designated administrators.

User authentication

Authentication is an essential part of any authorization system, therefore a **TrucStudio** user has to log on to the authentication service with his username and password before any data exchange can happen. Logging in will create a "session" with an associated session ID that can be used for future requests. Authenticated users will be able to access the contents of the entity repository according to their project memberships and access rights.

Offline work

Work with **TrucStudio** should still be possible while no Internet connection is available. All changes made during offline work can be sent to the server during the next online session.

Extensibility

To allow future extensions of both, the **TrucStudio** client and server, the system should be built as flexible as possible in terms of software architecture.

4.1.2 Truc sharing

To organize entity accessibility, every stored entity belongs to a project, which may be a user's personal project or a designated project group. For every project there is a global accessibility rule that defines the access rights of owners, members and non-members ("all") to entities stored in that project.

4.2 Design

This section describes the overall design of the **TrucStudio - Truc Sharing** system followed by a detailed description of the most important aspects.

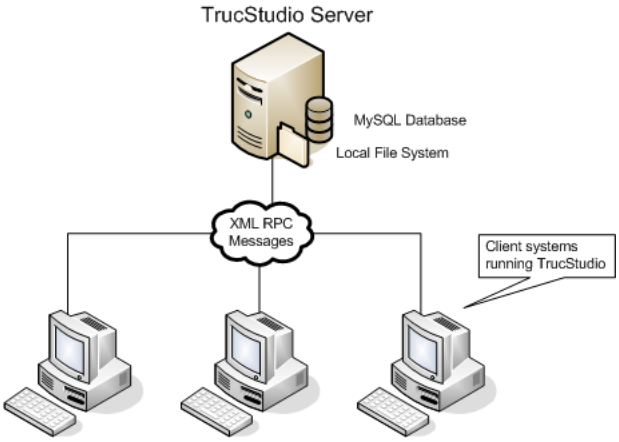

Figure 4.1: Client / Server Structure

The general design approach follows immediately from the requirements: **TrucStudio** will be extended to function as the client of a client-server system and a separate server application will be developed that handles the coordination and data management. To ensure future extensibility, **TrucStudio** and the **TrucStudio Server** application will use an XML-RPC style of communication.

4.2.1 Data storage

The problem of data sharing in a collaborative environment is a common problem. The goal is clear; let users edit shared data while preventing them from

constantly overwriting each other's work. Unfortunately there is no one-fits-all solution to this problem, instead the solution strongly depends on the working modalities of the users and the type of the data that is being shared. The following paragraphs illustrate some existing approaches to collaborative data manipulation and their advantages and drawbacks.

Simple file server (FTP)

The simplest approach is certainly a traditional file server (e.g. an FTP server) where users can upload and download data. The beauty of this solution lies in its simplicity. Uploading and downloading data can easily be implemented and there is a wide variety of free client and server software available. The handicap of such a simple server is self-evident. Instead of actually solving the problem of conflicting changes, it passes the responsibility on to the user. Accidental overwrites are bound to happen and can have very unpleasant consequences such as unrecoverable data loss.

Lock-modify-unlock

Many traditional version control systems use a lock-modify-unlock model where a user has to lock a file before he can work on it and release the lock after his change is complete. Only the user currently holding the lock on a file is allowed to modify it and if the user should refuse or forget to release the lock, all other users that want to modify that file are blocked.

Copy-modify-merge (SVN)

The popular Subversion [sub] version control system employs a sophisticated copy-modify-merge solution to allow truly concurrent work on shared data. Each user has a local copy of the shared data which he can modify. After the change is completed and nobody else has committed a change since the local copy was created, the data is committed to the repository. If another user has modified the data in the meantime, the local copy has to be updated first. During the update process, the changes from the other user(s) are merged into the working copy and all conflicting changes are reported back to the user. The user can then review the changes (and resolve possible conflicts) from the update and commit his work. Because of its flexibility, this approach is very popular among software developers, but the merging process is in practice limited to plain text files and managing binary data with Subversion can be quite challenging.

TrucStudio approach

The approach chosen for entity sharing in **TrucStudio** is a very simple mix of the above. When an entity is uploaded to the server, a new revision is created by simply storing the entity in its "native" **TrucStudio** XML format in the server's file system. No revisions are deleted no files overwritten; for every revision of every entity, there is an XML file. The server simply provides features to create a new revision and retrieve any version of any entity. The actual logic required for merging and updating the local working copy is implemented in the **TrucStudio** client and operates directly on the object structure in memory. This separation

of tasks allows a very simple and therefore reliable implementation of the server because it only has to perform a small set of basic data manipulations.

4.2.2 Database

To store user/project data, access rights and other data that has to be managed by the server, a relational database is used. The database is only directly accessible by the **TrucStudio Server** application and the client has to retrieve its data by going through the server application's communication interface. The choice of which database management system to use is a matter of personal preferences, since the functionality required by the **TrucStudio Server** application is supported by all common RDBMS. The only limiting factor in this respect is the EiffelStore library which mainly supports ODBC [odb] compatible databases – which comprises almost any available RDBMS. Choosing MySQL [mys] is therefore a decision that is based on previous experience with the product.

Database model

The database model that is used to store the user/project data and the entity meta data (see 4.2) is very simple. The login data is stored in the "`registereduser`" table and the projects in the "`usergroup`" table (the naming is an artifact of an earlier version). The project memberships are stored in the "`user_in_group`" table. For every entity that is stored on the server, there is an entry in the "`entity`" table which stores the current revision number of that entity, the URI and entity type. The "`access`" field is intended for future use and has no function at the moment. Every entity revision that is created can have an associated log message in the "`log_message`" table. The links between entities and log messages are maintained in the "`log`" table (at most one log message per revision of an entity). All primary object tables except the "`entity`" table use simple integer IDs as primary keys.
The MySQL database creation script that contains the table declarations including foreign key constraints and field types can be found in the **TrucStudio** Subversion repository under

```
https://svn.origo.ethz.ch/trucstudio/trunk/src/
    server/database_scripts/create_database.sql
```

4.2.3 Communication

The communication between the client (**TrucStudio**) and the server is an XML-RPC based message system. Each "call" from the client to the server (the server is assumed to be passive although this might not be the case in the actual implementation) is transformed into an XML message that is sent to the server as an HTTP request. The server unwraps the message, performs the requested operation and replies with an XML message in an HTTP response. The exact format of the messages and the different supported requests are described in chapter 6. While this method of communication might not be the most efficient one, it certainly is very flexible and as an added bonus, the system is not limited to one specific client since XML-RPC is an open standard.

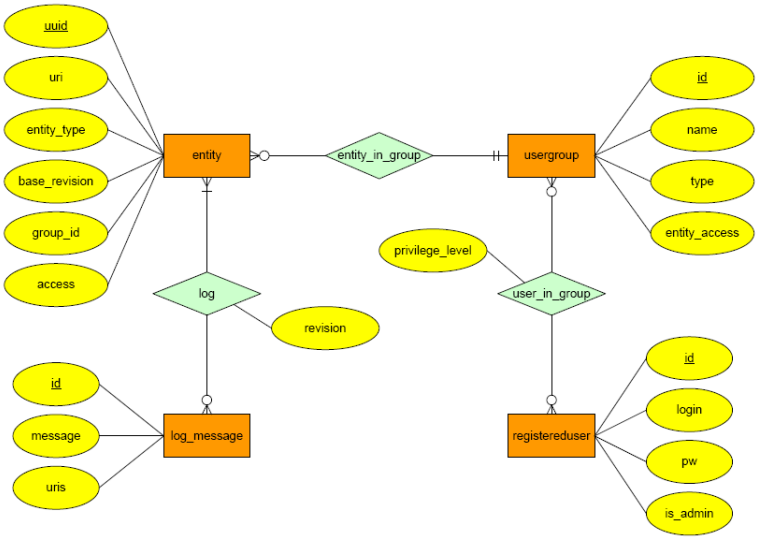

Figure 4.2: ER-Diagram of the database structure

Although the XML-RPC functionality that is required by the **TrucStudio Server** is very limited, it makes sense to use an existing full XML-RPC compatible implementation (again in the spirit of future extensibility). The Goanna [goa] library provides a full XML-RPC implementation and is also being used by the Origo [ori] project developed at ETH. By using a third party library it is possible to focus on the overall design of the project instead of spending a lot of time on detailed low-level implementations.

Chapter 5

Implementation

5.1 Truc sharing

Until the introduction of Truc sharing, any entity was either loaded from a file or created manually in **TrucStudio**. As a result, resolving entity references was never a problem, since referencing relationships could only be created from objects in the current working set to other objects in the current working set and saving the entity tree always resulted in a full serialization of the working set. Entities that can be stored remotely introduce an entirely new set of problems. It cannot be guaranteed that all entities which are referenced by entities in the working set are also in the working set, in fact they may not even be on the local machine or not available at all.

5.1.1 Local, working copy and remote entities

A local entity is one that has never been uploaded to the entity repository before and has a local URI (a URI where the host name is "local") associated with it. Although the entity itself is local it can nevertheless be referenced by and have references to non-local entities. On the other side, there are remote entities that have at one time been uploaded to an entity repository. Remote entities have a URI attribute specifying the server of their origin and optionally a revision number specifying the version of the entity that they represent. When a remote entity is downloaded, it is integrated into the current working set as a "working copy" which can be modified at will. Changes to a working copy are only propagated back to the server by manual uploads. This process prevents unwanted data degeneration on the server by enforcing conscious decisions to "save" the changes.

5.1.2 Revisions

Every modification of an entity that is committed to the server creates a new revision of said entity starting with revision 1 when an entity is uploaded for the first time. An entity URI may specify the exact revision to which it points, or if no revision is specified, it is assumed to point to the "HEAD" revision (the newest available revision). It is not possible to have several working copies of the same entity with different revisions in the same working set, since this would

break the consistency of the model. For example, a notion may only be part of one Truc and having two different versions of that Truc that both contain the same notion is prohibited.

5.1.3 Updating and merging

A new revision of an entity can only be created if the locally modified version is based on the most recent revision in the repository. If a user tries to commit from an earlier version, he gets notified that he has to update his working copy first. The reason for this restriction becomes clear in a simple multiuser example:

- User A checks out revision n of a certain entity X
- User B also checks out revision n of entity X
- User A makes some modifications to entity X and commits the changes creating revision $n + 1$

If user B would now also change X and commit his changes without first updating to revision $n+1$, he would "overwrite" A's changes without even noticing it. Although technically no data would be lost, this is in most cases not what users expect, and that is why we force the user to update his working copy before he can commit. Updating a working copy that doesn't have local modifications is done by replacing the current working copy with the most recent revision from the repository. If the working copy has local modifications, the user is presented with a diff-view of his local copy and the head revision. In this side by side comparison view of the two versions, the user decides which changes from the head revision he would like to include in his working copy. Once the update is completed, the user commits the merged changes back into the repository.

5.2 Model redesign

In the course of this master thesis, the class model building the core of the **TrucStudio** project has undergone major changes. This section will give future developers (and other interested readers) an overview over the changes made, the motivation behind the changes, and the state of the model as a whole.

Before the redesign, the model basically consisted of the implementation classes *TS_CLUSTER*, *TS_TRUC*, *TS_NOTION*, *TS_COURSE* and *TS_LECTURE* which all were directly derived from the deferred *TS_ENTITY* class. Although the mentioned implementation classes were very similar in their functionality, they were still different enough to require special handling in many places in the code. This resulted in a lot of "almost duplicate" features, that mainly differed in the types of their arguments and results. This code was very hard to maintain since it was easy to overlook one of the many little differences.
In the new model as shown in Figure 5.1, every entity is potentially owned by one or more entity containers. This models the DAG (directed acyclic graph) structure of the domain knowledge tree as well as the course/lecture model. The root of both the domain knowledge and course tree is the *TS_TRUC_MANAGER*

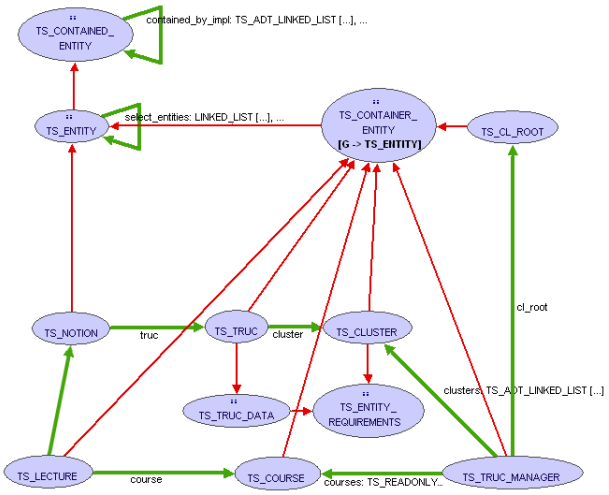

Figure 5.1: Class diagram of the new core model

5.3 XML I/O

All XML input/output related classes have been redesigned during the development of the entity sharing extension of **TrucStudio** to facilitate creating and parsing fragments of `*.txm` documents. To get rid of all the XML string concatenations in the previous versions, we use the Eiffel-DOM library (see chapter 7 for more information) for parsing and generation of XML output. For every major XML element in the `*.txm` format, there is a corresponding Eiffel class that can read the element from an *XML_ELEMENT* into the object it represents and serialize the object to XML. Even in its current state, using the Eiffel-DOM library greatly simplified the I/O classes in terms of understandability and error susceptibility.
One of the more subtle requirements of entity sharing is the need to update existing entities in-situ, because otherwise updating an entity would also require the update of all objects that reference the old entity. By using the entity cache, the new I/O classes can automatically overwrite an existing entity if the situation requires that.

5.4 RPC Integration

Since RPC is, as the name suggests more procedural than object oriented, integrating it into an object oriented design is not as easy as it could be. We chose to group the remote procedures with similar functionality together into Goanna "services" to improve maintainability. There is a session service that contains procedures needed for session management, an entity repository service with procedures related to repository management and a user management service for user and project administration.
For every service, there is a server side implementation in the form of a class that inherits from *GOA_SERVICE* and provides the service features. On the client side there is a proxy class for each service with similar feature signatures

that marshal the given parameters into an XML-RPC call and unmarshall the response. Any remote procedure call will therefore go through its proxy feature where the parameters will be serialized and sent to the server. The server parses the incoming XML message and passes the parameters on to the corresponding service feature. The feature then performs the requested operation and sends a result in the form of another XML message back to the client. There, the proxy feature de-serializes the response and returns the actual result.
Of course, there are lots of things that can go wrong with a remote procedure call and these cases have to be handled. Every RPC feature has a result type of *TS_XRPC_RESPONSE* which can contain an error message instead of the actual result, so on the client side RPC error handling is reduced to checking the received response for an error message and displaying it to the user. On the server side, every unrecoverable error like invalid input parameters has to be caught and translated into a meaningful error message. Even more important than a meaningful error message is, that in the case of a failure, the data collection does not get corrupted. This means that any modification to the database that has been made up until the point of failure has to be reverted before any error message can be returned.

5.4.1 Sending XML data

The XML-RPC specification doesn't allow "inlining" XML parameters in the way that is required for entity sharing. Considering that most of the transmitted data volume will probably be entities in the form of XML, this might seem like a major setback, but fortunately there is a simple workaround for this dilemma. Instead of sending the XML data in plain text, it is converted to a base64 encoded[1] string which is then passed as XML-RPC parameter. Of course the encoding and decoding implies a performance penalty, but tests have shown that the encoding/decoding is only a fractional part of the overall communication delay and has been deemed an acceptable loss.

5.5 The entity cache

The class *TS_ENTITY_CACHE* is the "birth place" of all entities. It is used to control aliasing and prevent accidental creation of duplicate entities.
When a **TrucStudio** file is loaded, the XML document is converted into an object structure of entities. If the XML declaration of an entity A contains a reference to another entity B, then the resulting object A' will have a reference to the object B'. But what if object B' does not yet exist when object A' is created? And what if there are other entities referencing B that are processed before B? One possible solution for this problem is to route all entity requests through the entity cache. When a new entity object is created from its XML declaration, every reference (in the form of an entity URI) is passed to the entity cache and the corresponding entity is returned. If the requested entity has not been created or referenced before, the cache simply creates a new uninitialized stub which is then added to the cache so that all future requests for the entity with the same URI will return the same object.

[1] See http://en.wikipedia.org/wiki/Base64 for an explanation of base64 encoding

The cache also provides a method to create so-called "dummy" entities that can be used for example to compare two different revisions of the same entity without breaking the model consistency. The main differences between a dummy and a real entity are that in a dummy some of the invariants are disabled and references to other entities are handled differently.

5.6 The diff-viewer

The diff-viewer is the core of the merging process and therefore a very important aspect of the entity sharing mechanism. The task of the diff-viewer is to present two different versions of a property in a side-by-side view with the option of adapting the value of one of the property versions. When presenting the diff-view of the summary of a Truc for example, the diff-viewer shows two text fields. The left text field is editable and contains the value of the "base" version while the right text field is read-only and contains the value of the "reference" version. Please note that, although in this text it is usually assumed that the values that are compared are different versions from the same entity, this is not strictly necessary. The implementation also allows the comparison of values from two different entities as long as the entities have the same type – and yes, it would theoretically also be possible to compare values from different entity types, but we had to draw the line somewhere.

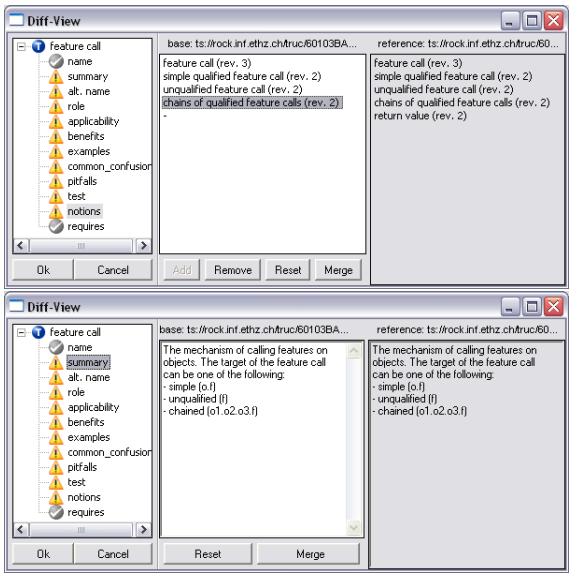

Figure 5.2: The upper diff-viewer shows a list type property while the bottom image shows a text type property

The controls and interaction possibilities in a diff-view are determined by the type of the property that is being displayed. Each property type can compare two instances of the specified type as illustrated in Figure 5.2. A list property can for example compare two lists of objects (to be more specific, two *DS_LINEAR[ANY]*). The property implements a comparison algorithm and can

check whether the two versions are "the same" according to the property rules. For every property type, there is a corresponding view that consists of a number of UI controls that present the values of the two versions and allow the user to modify the base version (if this is applicable) – the reference version is always immutable. Following is a list of the currently supported property types with a general description and some type specific implementation details.

5.6.1 Property types

TS_INT_DIFF_PROPERTY

The integer diff-property is the simplest property. It takes two *INTEGER*s and presents each of them in a spin box. The implementation is straight forward and uses the equality check of integers.

TS_STRING_DIFF_PROPERTY

String comparison is almost as simple as the integer diff-property. The main difference is that, since *STRING* is a reference type, there are a lot of calls to {*STRING*}.twin to avoid accidental aliasing.

TS_LIST_DIFF_PROPERTY and *TS_ENTITY_LIST_DIFF_PROPERTY*

Entity properties like the Trucs contained in a cluster or the notions contained in a Truc are sets of entities and when comparing two different sets, the order is irrelevant. For this purpose, the list diff-property implements a comparison algorithm that takes two *DS_LINEAR[ANY]* objects that are assumed to be sets of compatible objects (or entities in the case of the entity list diff-property). The view for these properties consists of two *EV_LIST*s with a list item for every object in the union of the two sets. Elements that are not present in one set, but contained in the other are visualized as "-" items. Selecting a list item in the base list allows the user to add/remove the object from the base set.
The entity list diff is a simple restriction on the general list diff-property that uses the URI of entities to determine object equality.

TS_ORDERED_LIST_DIFF_PROPERTY

There is currently only one list property where the order of the contained objects matters: the lectures contained in a course. Since the requirement of ordered comparison does not integrate very well with the general list property, the implementation for the ordered list is substantially different from the list diff-property. The view for the ordered list property looks very similar to the one for unordered lists, but it contains some additional controls to manipulate the object order in the base version.

5.6.2 Property implementation

When a new instance of a property class is created, the actual values of the two different versions are "cloned" and stored within the property object, so that modifications of the values in the property will not affect the original values. Any manipulation in the view is translated into a corresponding change in the

property value. Only by leaving the diff-dialog using the "OK" button are the changes propagated to the actual values of the affected object.

5.6.3 Diff-dialog

The *TS_DIFF_DIALOG* has two major sections (see Figure 5.2):

- The "container" tree view lists all entities in the current diff-set and for each entity node, the list of relevant properties of that entity. Next to each property there is an icon that marks the property as "conflicted" when the two property versions differ or as "ok" otherwise.
- The diff split view contains controls appropriate for the currently selected property type.

Clicking the "Cancel" button aborts the merge operation, discards all modifications made in the diff-dialog and the entities remain unchanged.

Chapter 6

Services

6.1 Common

This chapter describes the XML-RPC API provided by the **TrucStudio Server** in their version 1.0.r901 (version 1.0; created from revision 901) – see also section 6.2.
Sections 6.2 through 6.5 contain the API reference for all XML-RPC services provided by the **TrucStudio Server**.

6.1.1 Request / response structure

The general form of an XML-RPC request/response looks as follows:

```
[Request]
<?xml version="1.0"?>
<methodCall>
  <methodName>{service}.{method_name}</methodName>
  <params>
    <param>{parameter}</param>
    ...
  </params>
</methodCall>

[Response]
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>{result}</param>
  </params>
</methodResponse>
```

There are two different types of errors that can occur during XML-RPC calls which are handled differently in TrucStudio.

- **XML-RPC errors**
  are malformed requests that are caught and handled by the Goanna library. They manifest as normal XML-RPC faults (the "methodResponse"

element will contain a "fault" element with a detailed error description. The exact form of the fault is determined by the Goanna library and may change with future changes of the Goanna library.

- **Service errors**
  are semantic errors in the input data that result in the service not being able to successfully complete the request. Those errors are processed on the **TrucStudio** level in the service proxy features.

TrucStudio service responses have the general form

```
[Response]
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <struct>
        <member>
          <name>is_ok</name>
          <value>
            <boolean>{OK_value}</boolean>
          </value>
        </member>

        <member>
          <name>error</name>
          <value>
            <string>{error_message}</string>
          </value>
        </member>

        or

        [<member>
          <name>{result_name}</name>
          <value>{actual_result}</value>
        </member>]

      </struct>
    </param>
  </params>
</methodResponse>
```

If the "is_ok" member has a value of "1", this means that the request was processed successfully, if the value is "0", there has been an error and the "error" member will be present. If no error occurred, and the service procedure is expected to return a result, the member after "is_ok" will contain the result as it is specified by the service procedure declaration. Note that if no return value type is specified in the service API reference, the XML-RPC response will only contain the "is_ok" member without any additional results (except of course the error message if the request failed).

6.1.2 Custom TrucStudio structs

In the **TrucStudio** project, the actual data is transmitted in the form of custom XML-RPC structs. For each of these structs, there is a corresponding Eiffel class that provides features to serialize the object into its XML-RPC format and deserialize the struct back into its object form.

This section describes the different structs in in the form of member declaration tables. Each row states the name of a member, its type and the last column describes the value of the member. The title for each table is the name of the Eiffel class name since XML-RPC structs are not named.

TS_ACCESS

Name	Type	Description
group	int	the group id of the group this access declaration is referencing
name	string	the name of the group
access	string	the access level as a single character ("o", "m" or "-")
entities	string	the permissions of the entity store – three characters ("w", "r" or "-") specifying the permissions for owners (first character), members (second character) and everybody else (third character)

TS_ENTITY_INFO

Name	Type	Description
uri	string	the entity URI including the entity name
entity_type	string	type of the entity ("cluster", "truc", ...) – matches the entity type of the URI
base_revision	int	the most recent revision number of the entity

TS_LOG_ENTRY

Name	Type	Description
revision	int	the revision number of the entity this log entry is referring to
message	string	the log message for this revision
uris	array of string	list of URIs, specifying all entities that were affected by the same commit action

TS_USER

Name	Type	Description
is_admin	int	a value of "1" indicates that the user has administrative privileges, a value of "0" denotes a normal user
rights	array of *TS_ACCESS* structs	list of access right declarations for the user

TS_MEMBER

Name	Type	Description
id	int	user ID of the current member
name	string	name of the current member
privilege_level	string	three character privilege level specification

6.2 Diagnostic service

The diagnostic service is mostly used for debugging purposes. The "echo" methods allow checking basic server availability and the "version" method can be used by clients to determine the compatibility with the rest of the provided XML-RPC API.

6.2.1 diagnostic.echo

Simply returns the string that is passed as argument.

Parameters

Param #	1
Desc.	an arbitrary string
Type	string
Example	

```
<param>
 <value>
  <string>this is a test</string>
 </value>
</param>
```

Return value

Name	session
Desc.	the very same string that was sent as argument
Type	string
Example	

```
<?xml version="1.0"?>
<methodResponse>
 <params>
  <param>
   <value>
    <string>this is a test</string>
   </value>
  </param>
 </params>
</methodResponse>
```

6.2.2 diagnostic.echo_reverse

Reverses the string that is passed as argument.

Parameters

Param #	1
Desc.	an arbitrary string
Type	string
Example	

```
<param>
 <value>
  <string>this is a test</string>
 </value>
</param>
```

Return value

Name	session
Desc.	the reversed argument string
Type	string
Example	

```
<?xml version="1.0"?>
<methodResponse>
 <params>
  <param>
   <value>
    <string>tset a si siht</string>
   </value>
  </param>
 </params>
</methodResponse>
```

6.2.3 diagnostic.version

Returns the version of the server that is being used and the version of the XML-RPC API that the server uses. This information is useful for compatibility checks.

Parameters

none

Return value

Name	server_version & api_version
Desc.	the version of the server and the API
Type	string
Example	

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <struct>
        <member>
          <name>is_ok</name>
          <value>
            <boolean>{OK_value}</boolean>
          </value>
        </member>
        <member>
          <name>server_version</name>
          <value>
            <string>1.0.r901</string>
          </value>
        </member>
        <member>
          <name>api_version</name>
          <value>
            <string>1.0.r901</string>
          </value>
        </member>
      </struct>
    </param>
  </params>
</methodResponse>
```

6.3 Session service

Most of the service procedures require a valid session ID to perform their task. To get a session ID, the user has to authenticate himself with the session service using his/her username and password. The session service will then start a new session that will be associated with the authenticated user and return a new session ID. Since sessions have an expiration date associated with them, it is not necessary to "close" a session.

6.3.1 session.start

Starts a new session for the user with the given login and password. If the login attempt is successful, the procedure will return the assigned session ID. If the server couldn't verify the login information, the response will contain an error.

Parameters

Param #	1
Desc.	the login (username) of the user
Type	string
Example	

```
<param>
 <value>
  <string>john.doe</string>
 </value>
</param>
```

Param #	2
Desc.	the user's password
Type	string
Example	

```
<param>
 <value>
  <string>topsecretpassword</string>
 </value>
</param>
```

Return value

Name	session
Desc.	the assigned session UUID in the format xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx (8-4-4-4-12 hexadecimal digits 0-9 and A-F)
Type	string
Example	

```
<value>
 <string>3F2504E0-4F89-11D3-9A0C-0305E82C3301</string>
</value>
```

6.4 User management service

The user management service provides remote procedures to perform administrative tasks that are not directly related to entity sharing. Via the user management service it is possible to create new user accounts.

6.4.1 user.create_user

Creates a new user account on the server where the request is processed. If the specified username is already in use, the request will fail and report the error.

Parameters

Param #	1
Desc.	the username for the new account
Type	string
Example	

```
<param>
 <value>
  <string>john.doe</string>
 </value>
</param>
```

Param #	2
Desc.	the password for the new account
Type	string
Example	

```
<param>
 <value>
  <string>topsecretpassword</string>
 </value>
</param>
```

Return value

none

6.4.2 user.create_group

Creates a new virtual project for entity storage. The name of the project has to be unique, otherwise the request will return an error. This procedure requires a user with administrative privileges.

Parameters

Param #	1
Desc.	the session UUID in the format xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx (8-4-4-4-12 hexadecimal digits 0-9 and A-F)
Type	string
Example	

```
<param>
 <value>
  <string>3F2504E0-4F89-11D3-9A0C-0305E82C3301</string>
 </value>
</param>
```

Param #	2
Desc.	the name of the new project
Type	string
Example	

```
<param>
 <value>
  <string>john.doe-s_project</string>
 </value>
</param>
```

Return value

none

6.4.3 user.change_password

Changes the password associated with a username. To change the password, the old and new password have to be provided and the operation is only completed successfully if the old password matches the password stored in the server's database.

Parameters

Param #	1
Desc.	the session UUID in the format xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx (8-4-4-4-12 hexadecimal digits 0-9 and A-F)
Type	string
Example	

```
<param>
 <value>
  <string>3F2504E0-4F89-11D3-9A0C-0305E82C3301</string>
 </value>
</param>
```

Param #	2
Desc.	the old (i.e. current) password
Type	string
Example	

```
<param>
 <value>
  <string>old_secret</string>
 </value>
</param>
```

Param #	3
Desc.	the new password
Type	string
Example	

```
<param>
 <value>
  <string>new_secret</string>
 </value>
</param>
```

Return value

none

6.4.4 user.change_admin

With this procedure, administrators can grants/revoke administrative privileges to other users. There is a built-in fail-safe mechanism that prevents administrators from revoking their own admin rights, so trying to do that will result in an error.

Parameters

Param #	1
Desc.	the session UUID in the format xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx (8-4-4-4-12 hexadecimal digits 0-9 and A-F)
Type	string
Example	

```
<param>
 <value>
  <string>3F2504E0-4F89-11D3-9A0C-0305E82C3301</string>
 </value>
</param>
```

Param #	2
Desc.	the username of the user whose admin rights should be granted/revoked
Type	string
Example	

```
<param>
 <value>
  <string>john.doe</string>
 </value>
</param>
```

Param #	3
Desc.	"true" to grant admin rights to the specified user, "false" to revoke his/her admin rights
Type	boolean
Example	

```
<param>
 <value>
  <string>1</string>
 </value>
</param>
```

Return value

none

6.4.5 user.modify_user_privileges

This procedure allows project owners to add new members, promote/demote members and owners and remove members from the group. To do this, a project owner has to specify the user whose privileges should be modified, the project for which it should happen and the new privilege level of the user as a single character ('o' for owner, 'm' for member or 'a' for all the others).

Parameters

Param #	1
Desc.	the session UUID in the format xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx (8-4-4-4-12 hexadecimal digits 0-9 and A-F)
Type	string
Example	

```
<param>
 <value>
  <string>3F2504E0-4F89-11D3-9A0C-0305E82C3301</string>
 </value>
</param>
```

Param #	2
Desc.	the username of the user whose privileges should be modified
Type	string
Example	

```
<param>
 <value>
  <string>john.doe</string>
 </value>
</param>
```

Param #	3
Desc.	the name of the project (the user that requests this operation has to be an owner of this project)
Type	string
Example	

```
<param>
 <value>
  <string>secret_project</string>
 </value>
</param>
```

Param #	4
Desc.	the new privilege level for the specified user (one single character)
Type	string
Example	

```
<param>
 <value>
  <string>o</string>
 </value>
</param>
```

Return value

none

6.4.6 user.modify_entity_access

Project owners can modify the read/write access rights for owners, members and everybody else with this procedure. The access rights specification has to be of the form $[w|r|-][w|r|-][w|r|-]$ (three characters, each either 'w', 'r' or '-').

Parameters

Param #	1
Desc.	the session UUID in the format xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx (8-4-4-4-12 hexadecimal digits 0-9 and A-F)
Type	string
Example	

```
<param>
 <value>
  <string>3F2504E0-4F89-11D3-9A0C-0305E82C3301</string>
 </value>
</param>
```

Param #	2
Desc.	the name of the project whose entity access rights should be modified
Type	string
Example	

```
<param>
 <value>
  <string>secret_project</string>
 </value>
</param>
```

Param #	3
Desc.	the new access rights specifications (three character code)
Type	string
Example	

```
<param>
 <value>
  <string>w--</string>
 </value>
</param>
```

Return value

none

6.5 Entity repository service

The entity repository service contains procedures to query and manipulate the entities that are stored on the server. Entities are always carried as base64 encoded XML elements.

6.5.1 repository.list_entities

Creates a listing of all entities stored within the specified project's entity store. The returned *TS_ENTITY_INFO* structs contain the URIs of the entities that can be used for subsequent "repository.get_entities" calls. If the entity store of the given project is empty, the result struct will not contain an "entities" member. This should be interpreted as an empty array (XML-RPC does not allow empty arrays) and processed accordingly.

Parameters

Param #	1
Desc.	the session UUID in the format xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx (8-4-4-4-12 hexadecimal digits 0-9 and A-F)
Type	string
Example	

```
<param>
 <value>
  <string>3F2504E0-4F89-11D3-9A0C-0305E82C3301</string>
 </value>
</param>
```

Param #	2
Desc.	the name of the project for which the entities should be listed
Type	string
Example	

```
<param>
 <value>
  <string>object-orientation</string>
 </value>
</param>
```

Return value

Name	entities
Desc.	a list of entity info declarations - one for every entity in the requested group
Type	array of *TS_ENTIT_INFO* structs
Example	

```
<array>
 <data>
  <value>{entity_info_struct}</value>
  ...
 </data>
</array>
```

6.5.2 repository.show_log

Gets the list of log entries associated with the requested entity. The result is a list of *TS_LOG_ENTRY* structs, one for each revision of the entity.

Parameters

Param #	1
Desc.	the session UUID in the format xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx (8-4-4-4-12 hexadecimal digits 0-9 and A-F)
Type	string
Example	

```
<param>
 <value>
  <string>3F2504E0-4F89-11D3-9A0C-0305E82C3301</string>
 </value>
</param>
```

Param #	2
Desc.	the entity URI of the entity for which the log listing should be generated
Type	string
Example	

```
<param>
 <value>
  <string>{an_entity_uri}</string>
 </value>
</param>
```

Return value

Name	log_entries
Desc.	a list of log entries - one for every revision of the entity
Type	array of *TS_LOG_ENTRY* structs
Example	

```
<array>
 <data>
  <value>{log_entry_struct}</value>
  ...
 </data>
</array>
```

6.5.3 repository.get_entities

Fetches the entities that are specified by the list of URIs. If the operation is successful, the result will contain one entity for every URI that is passed as argument. The procedure performs no sanity check on the requested entity URIs, which means that the result may contain the same entity more than once if the request requires that.

Parameters

Param #	1
Desc.	the session UUID in the format xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx (8-4-4-4-12 hexadecimal digits 0-9 and A-F)
Type	string
Example	

```
<param>
 <value>
  <string>3F2504E0-4F89-11D3-9A0C-0305E82C3301</string>
 </value>
</param>
```

Param #	2
Desc.	an array of entity URIs
Type	array of strings
Example	

```
<param>
 <array>
  <data>
   <value>
    <string>{an_entity_uri}</string>
   </value>
   ...
  </data>
 </array>
</param>
```

Return value

Name	entities
Desc.	the list of the requested entities
Type	array of base64 strings
Example	

```
<array>
 <data>
  <value>{base64_encoded_entity}</value>
  ...
 </data>
</array>
```

6.5.4 repository.put_entities

Stores the given entities on the server, creating the first revision (future commits to these entities have to be done using the "commit_entities" procedure). Each entity is assigned its permanent URI and all references to entities in the entity list are updated accordingly. The entities are stored under the project specified by the second parameter. If an error occurrs while processing the request, no changes to the database will be made (atomic operation).

Parameters

Param #	1
Desc.	the session UUID in the format xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx (8-4-4-4-12 hexadecimal digits 0-9 and A-F)
Type	string
Example	

```
<param>
 <value>
  <string>3F2504E0-4F89-11D3-9A0C-0305E82C3301</string>
 </value>
</param>
```

Param #	2
Desc.	the name of the project under which the entities should be stored
Type	string
Example	

```
<param>
 <value>
  <string>oo_programming</string>
 </value>
</param>
```

Param #	3
Desc.	list of base64 encoded entities
Type	array of base64 strings
Example	

```
<array>
 <data>
  <value>{base64_encoded_entity}</value>
  ...
 </data>
</array>
```

Param #	4
Desc.	the log message that will be stored along with the shared entities
Type	string
Example	

```
<param>
 <value>
  <string>my initial import</string>
 </value>
</param>
```

Return value

Name	entities
Desc.	list of the stored entities including their assigned URIs and updated references
Type	array of base64 strings
Example	

```
<array>
 <data>
  <value>{base64_encoded_entity}</value>
  ...
 </data>
</array>
```

6.5.5 repository.update_entities

Fetches the newest versions of those entities that actually have a newer version than the one specified in the URI. The actual number of entities in the result may be smaller than the number of URIs in the second parameter since there might have been no updates. The response has the same form as the responses of the "repository.get_entities" and "repository.put_entities" procedures.

Parameters

Param #	1
Desc.	the session UUID in the format xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx (8-4-4-4-12 hexadecimal digits 0-9 and A-F)
Type	string
Example	

```
<param>
 <value>
  <string>3F2504E0-4F89-11D3-9A0C-0305E82C3301</string>
 </value>
</param>
```

Param #	2
Desc.	an array of entity URIs
Type	array of strings
Example	

```
<param>
 <array>
  <data>
   <value>
    <string>{an_entity_uri}</string>
   </value>
   ...
  </data>
 </array>
</param>
```

Return value

Name	entities
Desc.	list of the stored entities including their assigned URIs and updated references
Type	array of base64 strings
Example	

```
<array>
 <data>
  <value>{base64_encoded_entity}</value>
  ...
 </data>
</array>
```

6.5.6 repository.commit_entities

The given entities are committed to the database, creating a new revision for each of them. The URIs are updated to reflect the created revision but the content of the entities is not checked. This means that committing an entity without any modifications will still result in a new revision which should be avoided.

Parameters

Param #	1
Desc.	the session UUID in the format xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx (8-4-4-4-12 hexadecimal digits 0-9 and A-F)
Type	string
Example	

```
<param>
 <value>
  <string>3F2504E0-4F89-11D3-9A0C-0305E82C3301</string>
 </value>
</param>
```

Param #	2
Desc.	list of base64 encoded entities
Type	array of base64 strings
Example	

```
<array>
 <data>
  <value>{base64_encoded_entity}</value>
  ...
 </data>
</array>
```

Param #	3
Desc.	the log message that will be stored with every committed entity
Type	string
Example	

```
<param>
 <value>
  <string>my initial import</string>
 </value>
</param>
```

Return value

Name	entities
Desc.	list of the committed entities including their updated URIs
Type	array of base64 strings
Example	

```
<array>
 <data>
  <value>{base64_encoded_entity}</value>
  ...
 </data>
</array>
```

Chapter 7

Eiffel-DOM

7.1 Overview

Working on an application that stores its data in the form of XML documents obviously requires lots of parsing, navigating and writing XML files and strings. So does implementing an XML-RPC based server. Since this is such a fundamental requirement for this sort of application, a good XML library that facilitates the handling of XML documents with a clean and easy to understand API is a basic prerequisite. All previous versions of **TrucStudio** handled their XML demands using the Gobo [gob] library. Usage of this library resulted in rather ugly code in the sense that it is hard to read and every developer working on the **TrucStudio** project had his/her own approach to using the library (especially the methods for creating XML output resulted in almost unmaintainable code).

For these reasons we decided relatively early in the project to change the way that XML input/output is handled in **TrucStudio** and an independent library project was started. The goal of the library is to provide an XML processing API that enables its user to create easily readable code with very little effort. Positive experiences with the .Net XML API lead us to choose it as an archetype for the new library. The .Net XML API provides a (mostly) DOM-compliant interface to navigate and create XML documents and additionally, the DOM tree can be navigated using simple XPath expressions.

7.2 XPath functionality

Once an XML document is loaded into memory as a DOM tree structure, developers usually like to access specific parts of the document to query data or modify the DOM structure. In the DOM API this is usually done by navigating through the tree node by node and depending on the size and structure of the document, this can result in very large and "ugly" node access statements. In Eiffel-DOM there is another way to access nodes, using the *select_single_node* and *select_nodes* features of the *XML_NODE* class. Both these features take one string object representing an XPath expression and return a single node and a list of nodes respectively. The set of currently supported XPath expressions is:

- All XPath axes and their abbreviated versions except the "namespace" axis
  - ancestor
  - ancestor-or-self
  - attribute ()
  - child (/)
  - descendant
  - descendant-or-self (//)
  - following
  - following-sibling
  - parent (..)
  - preceding
  - preceding-sibling
  - self (.)
- text() and node() tests

Not yet implemented are the features

- predicates ("/A/B/C[2]" or "/A/B/C[@id = 12]")
- function calls
- arithmetic expressions

Although this might seem rather limited in terms of XPath functionality, the combination of DOM and the provided subset of XPath expressions allows very simple and efficient manipulation of XML documents.

Chapter 8

Problems

8.1 Bugs in libraries

8.1.1 Date / time parsing

One of the more entertaining bugs that was encountered during development of the program was the date/time parsing bug in the Eiffel time library. Parsing a date/time string that was generated by {*DATE_TIME*}.*out* with the feature {*DATE_TIME*}.*make_from_string_default* will parse the string wrong "half the time". The problem is that the default format string that is used to generate and parse the string uses the American time format (twelve hour representation with AM/PM) which works fine when generating a string, but will create an AM date/time object when the parsed string contains a PM. So, the first twelve hours of the day, everything is OK and the second twelve hours of the day, the parsing goes wrong...
The problem was solved by using a custom format string of the form "yyyy-[0]mm-[0]dd [0]hh:[0]mi:[0]ss.ff3".

8.2 Other inconveniences

8.2.1 *EV_GRID* and Pick-And-Drop

Most of the Vision2 widgets work very similar; pick-and-drop can be enabled for every displayed item independently. The nodes in an *EV_TREE*, the list items in an *EV_LIST*, etc. The developer can assign a "pebble" and specify the possible objects that can be dropped on it for each of them individually. The *EV_GRID* however does not work the same way. The rows and items are not pick-and-droppable, so to implement pick-and-drop behavior, pick-and-drop has to be enabled on the entire grid. Doing this has several drawbacks:

- Selecting the pebble for the pick-and-drop action has to be done dynamically using the *item_at_virtual_position* feature.
- If one of the "virtual" drop targets (that are also determined dynamically) accepts a certain pebble type, the entire grid has to accept that type. Visual feedback for invalid drop targets is therefore very limited.

8.2.2 ODBC data sources on Windows / Linux

Under Windows as well as Linux, it is possible to specify a username and password in the declaration of an ODBC data source. In both systems, providing this login data works as expected: connecting to the data source does not require the user to enter any more authentication data. In Eiffel programs using the EiffelStore library, the same rules apply only for Windows programs, under Linux connecting to the data source fails if the username and password are not provided explicitly in the code again.
To cope with this behavior, it is possible to specify a username and password for the ODBC data source in the server configuration file (see section 11.3.2). If they are specified, the default values provided in the ODBC data source declaration are overruled.

Chapter 9

Future Work & Conclusion

9.1 Future work

9.1.1 Improvements of the administrative features

Currently, the possibilities to administrate the entity repository via **TrucStudio** are quite limited. Possible extensions include:

- Allow the moving of entities from one project to another
- Renaming of projects
- Automatic removal of unused/empty projects
- Automatic removal of inactive user accounts
- "Deleting" entities (not removing them from the server, but hiding them from non-administrators)
- Moving entities (and their version history) from one server to another

It may even be desirable to create a separate administration application that works independently from the **TrucStudio** client.

9.1.2 Repository browsing

The repository browser could be extended with a search option to find specific entities by name or specific keywords in their content properties.

9.1.3 New server features

The XML-RPC based server application has much potential for future projects. The server could be used to send automated reminder emails based on scheduled course data or to manage course related material like slides and handouts along with the corresponding entities.

9.1.4 Server as a windows service

In theory it should be possible to run the **TrucStudio Server** as a windows system service. The windows service manager communicates with services by sending specific commands like "start", "stop", "pause", ... The server application would have to be extended to cope with this form of communication. If that is possible, it would be a simple matter of registering the server as a system service which could be done with a very simple batch file.

9.2 Conclusion

The main goal of this project was to extend the **TrucStudio** system with data sharing functionality that facilitates collaborative development of domain knowledge models and course models.

Extending the existing system with an entity repository server has significantly simplified the development and exchange of **TrucStudio** models. By imitating the behavior of established version management systems like Subversion, the effort to learn how to use the new entity sharing mechanism is kept very low. In combination with the new output generation mechanism, this opens up new possibilities for distribution of course material and other lecture related information.

The redesign of the underlying **TrucStudio** class model has improved code maintainability and the flexible architecture of the server application allows easy integration of new functionality. These aspects will certainly benefit future contributors of the project, which in turn will result in better and/or more **TrucStudio** features.

Part III

Compilation / Installation

Chapter 10

Compiling TrucStudio and the TrucStudio Server

10.1 IDE & compiler

First of all, you will need a recent Eiffel compiler. We used EiffelStudio [eif08] 6.0.6.9618 on Windows and EiffelStudio 6.1.7.574 on Linux during the development of **TrucStudio**. Of course, the compilation process should work with all versions greater or equal to the ones specified, but there is no guarantee.

10.2 Libraries

10.2.1 Gobo

The version of the Gobo library that comes with the EiffelStudio versions mentioned above contains some bugs that were fixed during the **TrucStudio** development. Since some of these bugs will inevitably crash the program (if it even compiles), it is essential to upgrade to a more recent version - specifically the head revision of the Gobo Subversion repository. Here is, how to do that (using UNIX-style path notation):

- Go to your `$ISE_EIFFEL/library/gobo/` directory
- Rename the folder "svn" to something like "old_svn"
- Create a new "svn" folder and check out

  `https://gobo-eiffel.svn.sourceforge.net/svnroot/gobo-eiffel/gobo/trunk`

  using your favorite Subversion client
- Set the `$GOBO` environment variable to `$ISE_EIFFEL/library/gobo/svn`
- Add the directory `$ISE_EIFFEL/library/gobo/spec/$ISE_PLATFORM/bin` to your `$PATH` environment variable
- Execute the command `geant install` in the svn directory

10.2.2 Eposix

- Download the eposix library from

  `http://www.berenddeboer.net/eposix/`

- Unpack the eposix folder into `$ISE_EIFFEL/library/`
- Set the `$EPOSIX` environment variable to point to the eposix folder
- Compile the necessary library files yourself or use the precompiled ones from the **TrucStudio** repository as explained in the next section

Compiling Eposix

Compiling the Eposix library yourself can be quite difficult depending on the setup of the machine you are using to do the compilation. However, if you prefer to compile it yourself (or if your OS/compiler combination is not supported by the precompiled libraries) you should start with the `INSTALL` file in the Eposix directory.

Precompiled libraries

The precompiled libraries can be found in the **TrucStudio** subversion repository under

`https://svn.origo.ethz.ch/trucstudio/trunk/lib/precompiled/`

The folder contains precompiled libraries for Windows/MSC (`libeposix_msc.lib`) and Unix (`libeposix.a`, `libeposix.la`). To move the library to the proper directory, you can use the install scripts `eposix.bat` / `eposix.sh` in

`https://svn.origo.ethz.ch/trucstudio/trunk/lib/`

If you experience problems with the precompiled libraries, it is probably necessary to compile them yourself.

10.2.3 ODBC (server only)

Note: In order to compile the Eiffel ODBC driver, you need the ODBC driver and header files for your database system. In the case of MySQL/Ubuntu, this would be the packages libmyodbc, unixodbc-dev, libiodbc2 and libiodbc2-dev. To use the Eiffel ODBC driver you first need to create the C library that Eiffel uses as its interface. Go to

`$ISE_EIFFEL/library/store/dbms/rdbms/odbc/Clib`

and execute the command `finish_freezing -library`. After that you should have the odbc library file in

`$ISE_EIFFEL/library/store/spec/$ISE_PLATFORM/lib`

10.2.4 Goanna & Log4E

The Goanna and Log4E libraries are both part of the Goanna [goa] project and are automatically checked out with the **TrucStudio** trunk. They are integrated as Subversion externals in the "lib" subdirectory. After checking out the **TrucStudio** trunk from the repository, execute the `goanna.bat` or `goanna.sh` script (depending on your operating system of course). This will "precompile" some of the classes in the Goanna library using geant. When the Goanna library is updated, it might be necessary to repeat this last step to make sure that all required files have been created.

10.3 The source code

Checking out the **TrucStudio** source code from

`https://svn.origo.ethz.ch/trucstudio/trunk`

will automatically check out the Goanna, Log4E and Eiffel-DOM libraries in the "lib" sub folder. The actual source code for the different sub-projects and internal libraries is located in the "src" folder. After the checkout you should execute `$TS_ROOT/lib/goanna.sh` or the `*.bat` version to precompile some of the Goanna classes. Also, if you would like to use the precompiled Eposix libraries, now would be the time to execute `$TS_ROOT/lib/eposix.sh` or `*.bat` to copy the libraries into the appropriate EiffelStudio directory.

10.3.1 src

The "src" folder contains the following sub folders:

- `format_spec`
  Contains the original TXM format specifications as well as XSL files to convert several legacy formats into the most recent version.
- `server`
  This is the Eiffel source code of the **TrucStudio Server** project
- `test_example`
  Contains example code on how to use the Gobo Test library
- `tslib`
  The tslib library contains common classes of the **TrucStudio Server** and the **TrucStudio** client. The classes are mostly related to XML-RPC based communication
- `tstudio`
  Contains the source code for the **TrucStudio** client application as well as all resources that are required to run the program.

Once the required dependencies are set up properly (see section 10.2) the process of compilation is actually straight forward. Open the `*.ecf` project configuration file with EiffelStudio and compile it.

Chapter 11

Installation

11.1 Requirements

The **TrucStudio Server** requires an ODBC database with a specific structure to operate. This section describes how to set up an ODBC data source and how to create the necessary SQL tables.

11.1.1 Setting up the database

This section assumes that you use a MySQL database. Using another SQL implementation should work as well, but some changes might have to be made to make things work.
Initializing the **TrucStudio Server** database is a simple matter of running the SQL script

```
database_scripts/create_database.sql
```

from the unpacked release package or the server folder from the repository on your database by typing

```
mysql -u username -pyourpassword < create_database.sql
```

where "username"/"yourpassword" is the username/password of a user with administrative privileges.

Warning: If there is already a **TrucStudio Server** database, it will get newly created and all data will be lost!
The initial dataset contains a **TrucStudio** user "admin" with password "admin" that has (of course) admin rights. Before you do anything else, you should change the password for the "admin" user to something more secure; you can do this either by directly setting the password in the database or using the **TrucStudio** administrative features.

Important: If you are using a database that does not support InnoDB tables, it is essential that you choose a replacement table type that supports transactions, otherwise the data integrity of the server database will be compromised.

ODBC installation

Windows

- Download and install the MyODBC driver for your MySQL server version from `http://dev.mysql.com/downloads/`
- Go to your system settings "Administration ⇒ Data Sources (ODBC)"
- Select the "System-DSN" tab and click "Add..."
- Chose the MySQL ODBC driver and click "Finish"
- Enter the connection information for your MySQL database and set the "Data Source Name" to `ts_server_db`
- After clicking "OK", your ODBC data source is set up and ready to be used

Unix

- Download and install the packages "libiodbc2" and "libmyodbc" (Ubuntu package names - depending on your system, they might be called differently)
- Open `/etc/odbcinst.ini` and add the following section

```
[MySQL]
Description = MySQL driver
Driver      = /usr/lib/odbc/libmyodbc.so
Driver64    =
Setup       = /usr/lib/odbc/libodbcmyS.so
Setup64     =
UsageCount  =
CPTimeout   =
CPReuse     =
FileUsage   = 1
```

- Open `/etc/odbc.ini` and add the following section

```
[ts_server_db]
Driver      = MySQL
Description = TrucStudio Server Database
Server      = localhost
User        = tsserver
Password    = trucstudio
Database    = tsserver
Option      = 3
```

(This assumes that there is a MySQL user "tsserver" with password "trucstudio" with full access on the "tsserver" database on the local machine)

11.2 TrucStudio

Download and extract the **TrucStudio** release package for your operating system from the **TrucStudio** home page [tru]. **TrucStudio** can now be started by executing the `trucstudio.exe` or `trucstudio` binary.

11.3 TrucStudio Server

Download and extract the **TrucStudio Server** release package. Although the server application can also run under windows, running the server as a system service is currently only supported under Linux.
You can install the server as a unix daemon that will start at system startup by typing

```
$ sudo ./install.sh
```

This will copy the `tsserver` binary into `/usr/bin` and create a default configuration file in `/etc/tsserver.config`. If this mode of operation is chosen, the server will run in `/var/www/tsserver` and all entities that are uploaded to the server will be stored in that directory. After the installation, make sure that the configuration file in `/etc/tsserver.config` matches your database setup and the hostname of your machine.
The daemon can be controlled by a command like

```
$ sudo /etc/init.d/tss start|stop|restart
```

Instead of registering the server as a system service, you can also run it directly from the folder containing the release. In that case the control command for the server would look like

```
$ sudo ./tss.sh start|stop|restart
```

In this mode, the server loads the configuration from the `tsserver.config` file in the release folder, so you should again make sure that the configuration is correct.

11.3.1 Command line parameters

The command line parameter `-c configfile` can be used to start the server using a designated configuration file. If the parameter is omitted, the application will try to load `./tsserver.config`.

11.3.2 Configuration

Following is the configuration file that comes with the release version of the server and a description of the different elements.

```
<?xml version="1.0" encoding="UTF-8"?>

<configuration>
 <server_settings>
  <port>2442</port>
```

```
  <url>rock.inf.ethz.ch</url>
  <session_timeout>120</session_timeout>
 </server_settings>
 <database>
  <dsn>ts_server_db</dsn>
  <login>tsserver</login>
  <password>trucstudio</password>
 </database>
 <filesystem>
  <root>./root</root>
  <entities>./entities</entities>
 </filesystem>
</configuration>
```

- **server_settings**
  - **port**: The port on which the server will listen for incoming connections.
  - **url**: The external host name by which the server is accessed. This will be stored as the "host" part of entity URIs.
  - **session_timeout**: The lifespan of a session on the server in minutes. After the session timeout has elapsed, the user will have to log in again.
- **database**
  - **dsn**: The name of the ODBC data source.
  - **login**: The login for the ODBC data source (depending on the system, this may be omitted).
  - **password**: The password for the ODBC data source (depending on the system, this may be omitted).
- **filesystem**
  - **root**: The root of the server's file system relative to the executable (currently this attribute is not used).
  - **entities**: The folder in which entities are stored (relative to the executable path).

Bibliography

[Alb08] Enrico Albonico. TrucStudio - Output Generation, 2008. Master Thesis.

[Cro07] Michele Croci. TrucStudio. A course management tool, 2007. Master Thesis.

[eif08] Eiffel studio – http://eiffelstudio.origo.ethz.ch/, 1985-2008.

[goa] Project Goanna, http://goanna.sourceforge.net/.

[gob] Gobo Eiffel Project, http://www.gobosoft.com/eiffel/gobo/index.html.

[Mey06] Bertrand Meyer. Testable, reusable units of cognition. *IEEE Computer*, 39(4):20–24, 2006.

[mys] MySQL AB – The world's most popular open source database, http://www.mysql.com/.

[odb] Open Database Connectivity (Wikipedia Entry).

[ori] Origo.

[POM07] Michela Pedroni, Manuel Oriol, and Bertrand Meyer. A framework for describing and comparing courses and curricula. In *to appear in ITICSE '07*, New York, NY, USA, 2007. ACM Press.

[POM+08] Michela Pedroni, Manuel Oriol, Bertrand Meyer, Enrico Albonico, and Lukas Angerer. Course management with TrucStudio, Submitted to ITICSE 2008. 2008.

[sub] Subversion – Version Control System, http://subversion.tigris.org/.

[tru] TrucStudio, http://trucstudio.origo.ethz.ch/.

[Wid07] Leo Widmer. TrucStudio. A Prototype, 2007. Master Thesis.