The PracTEX Journal, 2010, No. 1
Article revision 2010/01/01

Continuous Integration in LaTeX

Marco Antonio Gómez-Martín and Pedro Pablo Gómez-Martín

Email marcoa@fdi.ucm.es, pedrop@fdi.ucm.es

Abstract Have you ever co-written a paper using LaTeX together with some version control system such as SVN? Have you ever updated your local copy and the compilation has become broken due to a previous bad commit? Continuous integration avoids this problem using an auxiliary server that constantly checks the sanity of the repository, compiling the LaTeX documents after each commit, and notifying authors of possible problems. This paper describes how to configure this environment. Although the configuration effort is detailed, it is done only once and provides many benefits. In addition to doing compilation tests, all authors can be automatically informed by e-mail when a new version is committed, and the current `.pdf` version can be made available to third parties on the Web.

1 Introduction

Many of the documents that we write using LaTeX are created not by just one author but two or more. In fact, our analysis performed over the database of the DBLP computer science bibliography[1] reveals that the average number of authors of books, journal articles and articles in proceedings written until February, 2004 was 2.26 (see table 1).

This justifies the use of some kind of version control system, such as CVS (2) or Subversion (4) (also known as SVN) for collaborative writing of LaTeX documents. As the use of SVN has spread over the LaTeX community, different packages have appeared that allow documents to incorporate references to the properties of the last revision (10). There are also different LaTeX editors that ease the task of using SVN (3).

When creating a document using this software, every author has a local copy of the source files where they add their contributions. Periodically, they *commit*

1. http://dblp.uni-trier.de/

# of authors	# of contribs	Percentage
1	150320	31.83%
2	162546	34.42%
3	93148	19.73%
4	39371	8.34%
5	14660	3.1%
More than 5	12160	2.58%
Total	472205	100.0%

Table 1: Numer of authors per contribution in DBLP until Feb, 2004

their changes to the server in order for other users/authors to be able to *update* their local copy with those changes. The result is a boost in the productivity because the coordination is much easier, something especially important when the deadline is near.

However, the use of this approach has a small issue: when an author commits changes, the build may be accidentally broken. In our experience, this usually happens when an author forgets to upload auxiliary files, such as images. Another source of problems is the difference between platforms: when the author is using Windows or Mac, where the file names are case insensitive, he may break the document generation of another author using Linux.

This issue is well known in the software development, where programmers have used version control systems for decades. One of the solutions they have found is called "*continuous integration*". As we will see in the next section, it consists of having a dedicated machine that continuously checks if there are changes in the SVN repository. When it detects a commit, it updates automatically its local copy and tries to build the application in development. If something is wrong with the commit, it sends an e-mail to the programmer that performed it, to kindly ask him to fix it, to avoid inconveniences to other developers.

In this paper, we extrapolate the idea into the LaTeX world. We will describe how to set up a server machine in order to use continuous integration in LaTeX projects. As we will see, this will bring us two other benefits: the machine may send an e-mail to every author of the document, which helps them to know its progress, and the generated document for every revision may be placed on a

web server for other authors (or reviewers) to download without forcing them to generate the PDF.

This paper runs as follows. The next section describes in detail the ideas behind the continuous integration concept. After that, we present our motivation for having a machine for the continuous integration of LaTeX projects. Section 4 describes the piece of software that we have used. Section 5 explains the steps that authors should take in order to allow the server machine to manage their projects. This is followed by Section 6 that describes how to set up the server. After that we present some advanced topics and ideas for users that need more control over the installation of the server. The paper ends with some conclusions.

Over the entire paper we assume that the reader is already writing their documents using Subversion. If this is not his/her case, you may find the continuous integration approach presented here useless. We encourage the use of SVN (or any other version control system). For more information, please consult (8) or (6).

2 Continuous Integration

When a team uses, for any purpose, a version control system, some discrepancies can occur between the official content in the repositories and the local copies of the collaborators. When commits are done after each participant has made many local changes, problems such as file conflicts or incompatibilities can arise.

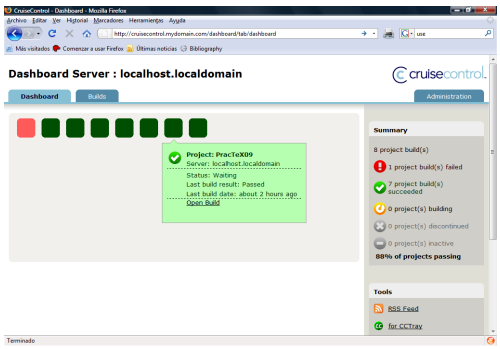
Software development teams, generally composed of many members, have dealt with this issue for decades. The inherent problems of the late integration caused (and still causes) many problems (and delays) when different software components had to be put together.

These difficulties are also present, in a smaller extent, when using a version control system while co-writing papers or documentation with LaTeX. Maybe the more common problem in this context is when someone forgets to commit a new file, typically an image, or when a file is incorrectly committed, preventing the rest of the team from compiling the new version.

A way to deal with all these problems is the known as *continuous integration*. Martin Fowler, a software engineering guru, defines it as (5):

> [It] is a software development practice where members of a team integrate their work frequently, usually each person integrates at least

(a) Cruise Control main page



(b) Cruise Control dash board

Figure 1: Cruise control screenshots

> daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible.

Therefore, *continuous integration* consists of encouraging developers to commit their changes as soon as possible to the mainstream, in order to provide fast feedback for developers.

An important aspect of this methodology is that an *infrastructure* is needed to allow programmers to build and test their changes as early and often as possible. The infrastructure will automatically do all these checks using the more recent project version when a new commit is done.

Fortunately, in recent years, continuous integration has had a major boost in the computer engineering area. This momentum has produced some tools for providing the previously mentioned infrastructure, some of them Open Source. We wondered if they could also be used to automatically test the commits into a papers' SVN, obviously a more reduced and simple environment than the bigger development teams. The next sections describe our decisions and experiences.

3 Motivation

One of the more important tasks of a research group is to generate papers that are usually written by at least two people. Today, many research groups have access

to server machines where reside different services, such as web or ftp sites. These servers can be used for a more "private" task: for hosting a version control system that helps the collaborative writing. A CVS or SVN server is almost mandatory for managing all this cooperative work.

Unfortunately, as said previously, version control systems do not prevent problems due to someone committing an invalid file. Software development teams use *continuous integration* to get over these difficulties. Our motivation was to implement in our research group the idea of using a new service for continuously testing that the different LaTeX projects were valid.

Figure 1 shows the result. The chosen continuous integration tool (described in the next section) provides two different interfaces to access its state. Both of them show six different papers[2], each of them forming a different project that is stored in an independent place in the SVN repository. At regular intervals, the continuous integration server checks each paper for modifications, and tests its validity. As an example, the figure shows that one of the papers has a compilation problem (the one called `AIIDE09`). The tool lets the user to browse into the error and shows him/her the error log generated by LaTeX during the failed generation.

As we will describe later in Section 6, our server machine is configured in such a way that:

- When a new commit is detected, the server notifies *all* the paper co-authors. This lets the collaborators remain informed about the evolution of other parts of the paper without *polling* the version control server or asking the other authors.
- The server publishes the generated document (usually `.pdf`) that will be accessible via the Web. This becomes quite useful when the paper must be read by external people, such as reviewers or advisors.

When starting a new paper, one of the authors will make the first commit into the SVN repository. After that, the continuous integration server administrator will add the new project so that the paper can be monitored. When new commits are done, the tool will test their validity, and send e-mails and show results through the web interface.

2. `CC-ConfigValidator` and `CC-ConfigUpdate` are not papers but projects related to the configuration of the software, as described in Section 7.

We decided to use Cruise Control as our continuous integration tool. The next section describes it, and then illustrates how the configuration process is done.

4 Cruise Control

Cruise Control[3] is one of the more well-known continuous integration tools. It is distributed under a BSD-style license and is free to use. Its main functions are, in fact, quite simple:

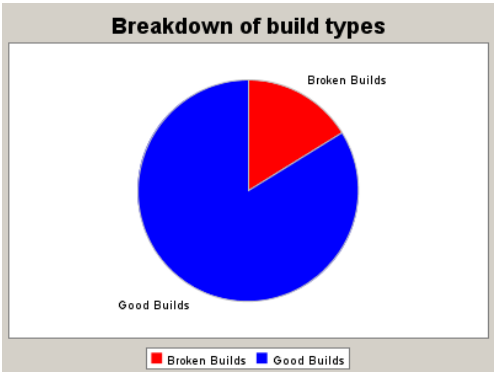
- It periodically tests if the project repository has had any changes.
- When modifications are detected, Cruise Control updates a local copy of the project, compiles it, and confirms the results to the authors in some way, usually by e-mail.

Therefore, Cruise Control acts as a supervisor that controls all the commits to the repository and informs when something was wrong. From the developers' point of view, the use of Cruise Control is non-intrusive; they can just wait for an e-mail from the Cruise Control system after a commit (Section 5). In our context, when a co-writer commits changes, he waits until the system e-mail confirms that the commit was correct. This constitutes a *sanity check* to avoid other teammates having invalid repository states that prevent them from compiling the paper.

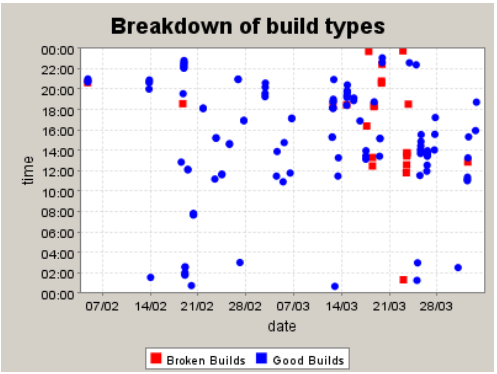
Cruise Control also provides statistical information about the project. Commit logs can be browsed, and different graphs are generated to show the commit patterns by type or days of the week (Figure 2).

From the administrator's point of view, the Cruise Control and version control servers can reside on two different machines, or can coexist in the same one. The only dependency is that the Cruise Control server needs a client of the version control system in use (SVN in our context) in order to update is project local copy. Deploying a new Cruise Control server is a tedious task (Section 6), although, fortunately, it must be done just once. As mentioned above, a Cruise Control server can be used for more than one project/paper. Once the Cruise Control server has been deployed, configuring it to keep vigilant watch over a new paper is straightforward and does not requires more than a few minutes.

3. http://cruisecontrol.sourceforge.net/



(a) Pie chart of build types



(b) Build types by date

Figure 2: Cruise control graphs

Curiously, a Cruise Control server can be in charge of more than one tool. For example, it can be testing if some papers are correctly updated in the SVN repository and, *at the same time*, testing if some C++ or Java projects in a CVS[4] server compiled correctly. In order for the administrator to instruct Cruise Control how to *build* the projects for generating the resulting artifacts (`pdf`, executable or `.jar` files respectively), a *building tool* is needed. Different alternatives exist for these types of tools, such as the UNIX `make` (9) or the Open Source `cmake` (1). Cruise Control, on the other hand, has been developed with Java in mind and uses Ant (7), the building tool more commonly used for building and deploying complex Java projects.

LaTeX compilation requires invoking different applications in sequence, such as `latex` (or `pdflatex`), `bibtex` and even `glosstex` or `makeindex`. Authors usually automate these tasks using the `make` building tool, available across a wide range of platforms. When using Cruise Control, the `make`-Ant gap must be bridged. The next section describes how to do this.

5 Writing while Cruise Control is watching

From the authors' point of view, Cruise Control is non-intrusive. They can write their documents as usual and commit them periodically. The only extra step

4. CVS is a different version control system, which preceded the more commonly used SVN.

needed is to provide, at the beginning of the process, some files that explain to Cruise Control how to build the document. In return, they will receive an e-mail a few minutes after each commit that will indicate whether Cruise Control was able to build the final document using the new version in the repository. Cruise Control acts as a *friendly overseer* that monitors the repository's sanity.

Usually LaTeX users edit their files using an editor that includes some kind of option/button to automatically compile the `.tex` file in order to create the final document (in either `.dvi`/`.ps` or `.pdf` format). More advanced users include in their project folder a `Makefile` that is able to generate the document using the `make` utility (9) that is available in virtually every Linux and MacOS distribution.

As we have explained in the previous section, Cruise Control was designed to be used in the development of Java projects. Therefore, if we want to use it to manage our LaTeX projects, we have to add some special files to the folder where the LaTeX files reside, in order for Cruise Control to know how to create the final document.

If you want to have your LaTeX project compiled by Cruise Control you have to add only two files to the main folder (i.e. the folder where the main `.tex` file is placed). The next subsections describe each of these files and their contents.

5.1 `Makefile`

If you are an advanced user (or one that does not use a specific LaTeX editor), you probably use a `Makefile` already. This file contains instructions on how to generate the final document using the `make` tool. This usually includes the execution of `latex` (or `pdflatex`) and `bibtex`. More complex documents may need to invoke other utilities in addition, such as `makeindex` or `glosstex`.

Therefore, the content of this file is related to the complexity of creating the document. The Code Block 9 shows a minimal file that has two different ways of execution: one that generates the document using pdfLaTeX and another using `latex`[5].

5. From the Cruise Control point of view only one of the executions is needed. In the example, we will configure Cruise Control to execute the first one (pdfLaTeX), and as a result the other is not strictly necessary; we have included it as a reference for readers that prefer use `latex`. You may also want to include some other targets (or execution paths), for example, a task that cleans up temporary files.

```
LATEX_NAME = myArticle

# Generation using pdfLaTeX
pdflatex:
        pdflatex $(LATEX_NAME)
        bibtex $(LATEX_NAME)
        pdflatex $(LATEX_NAME)
        pdflatex $(LATEX_NAME)

# Generation using 'latex'
latex:
        latex $(LATEX_NAME)
        bibtex $(LATEX_NAME)
        latex $(LATEX_NAME)
        latex $(LATEX_NAME)
        dvips $(LATEX_NAME).dvi
        ps2pdf $(LATEX_NAME).ps
```

Code Block 1:
Example of `Makefile` to generate the final document from '`myArticle.tex`'.

5.2 `build.xml`

This file is triggered in order to create the final document by Cruise Control. As we have explained in the previous section, Cruise Control uses the Ant building tool to compile the project, which expects an XML file containing the instructions.

Since the instructions to generate our final document are already coded in the `Makefile`, this XML file should just invoke it. The file in Code Block 2 executes the native `make` utility or `nmake` when it detects a Windows platform[6].

6. The `nmake` application is available in every edition of the Visual Studio tool created by Microsoft. If you have Cygwin or other distribution of `make`, it is easy to change the XML to use it.

```
<project name="make call" default="build">

<condition property="usenmake">
   <os family="windows"/>
</condition>

<target name="setgoal" unless="make.goal">
   <property name="make.goal" value=""/>
</target>

<target name="make" unless="usenmake">
   <exec executable="make" failonerror="true">
      <arg line="${make.goal}"/>
   </exec>
</target>

<target name="nmake" if="usenmake">
   <exec executable="nmake" failonerror="true">
      <arg line="${make.goal}"/>
   </exec>
</target>

<target name="build" depends="setgoal, make,nmake">
</target>
</project>
```

Code Block 2:
`build.xml` that should be added to the folder where the `Makefile` is.

6 Setting up the server

The Continuous Integration approach is based on the existence of a server machine that is running a specific piece of software (in our case Cruise Control).

This server application checks continuously (let's say every two minutes) if the SVN repository has changed, i.e. if someone has *committed* a new version of some of the files of the project.

Therefore, in order to take advantage of the Continuous Integration in LaTeX the first step is to configure this machine to have all the needed software. This section describes the process. We will assume that the reader has a machine running permanently and that it is connected to the network. Cruise Control installs a web server, so that users can use their web browsers to find out the current status of their compilations. In that sense, if you want your users to be able to access the machine over the Internet, the server should have a public IP address. In the rest of the paper, we will assume that the server is accessible through the name `cruisecontrol.mydomain.com`.

The list of tasks that the administrator of the machine must perform is[7]:

- You should be sure that the machine has all the pieces of software that will be used by Cruise Control and *by your projects*. This includes:
  - A Java distribution. As we have mentioned, Cruise Control is created in Java and meant to be used with Java projects. The server machine should have at least the JRE of the Java distribution. Fortunately you can download it from the Internet[8] or, if you are using Linux in your server, installing it using the package management tools of the distribution of your choice, such as `apt-get` in Debian or `rpm` in RedHat-based Linuces. Make sure that the environment variable `JAVA_HOME` points to the location of the JRE installation directory.
  - The Subversion client. Cruise Control will check the SVN repository, therefore it needs to have the SVN client. If you are thinking about installing Cruise Control in a Windows machine, bear in mind that the TortoiseSVN is not enough in this case; you need to have a command-line client, such as that developed by CollabNet.
  - The tools needed to compile your projects. In this case you have to install LaTeX and all the extra packages and languages that you may use in your documents.

7. This is a quite huge list of actions. However, bear in mind that this is done just once. The normal use of Cruise Control involves only the creation of new projects; this is treated in Section 6.3.
8. `http://java.sun.com`

– Depending on the usage policies of the server machine, you may want to add a new user, something like `cruisecontrol`, to the system. This new user will have limited privileges, because Cruise Control only needs to use SVN as a client and to compile LaTeX documents.
– It is also convenient to create a new user in the *SVN repository*. This user will have read-only access to the entire repository. Cruise Control will use it to checkout and update the projects it manages. As it will not edit the repository, it does not need to have write access to it (as a normal user would).
– Once the machine is ready to have Cruise Controlinstalled, you must download it from the Web and have it running. The next section describes this process.

6.1 Installing Cruise Control

As mentioned before, Cruise Control is an application distributed under a BSD-style license and is free to use. It is available on `http://cruisecontrol.sourceforge.net/`, together with extensive documentation that explains how to install and configure it.

The first step is to download the current version from the website. It comes in three different flavours: a `.zip` with the binary files, an installer for Windows platforms, and a `.zip` with the source Java files to build it from scratch. Our explanation is based on the first one, using the compressed file with all the binaries (as it is a Java program, the binaries are platform independent). At the time of this writing the current version is 2.8.2, though these instructions are also useful for previous versions[9] and we hope it will remain useful for future ones.

Once you have the `.zip` file (something like `cruisecontrol-bin-2.8.2.zip`), decompress it. The folder is not relevant so you can place it in any location. We will refer to this location in the future as `CCDIR`[10].

We then have to create the folders where the information about the projects will reside. It is desirable to have all of them under a common directory, such as `Work`. Making this folder independent of the Cruise Control installation eases

9. In fact, Figure 1 is based on version 2.7.2.
10. If you created a new user for Cruise Control, `CCDIR` may be something like `/home/cruisecontrol/cc`.

the task of updating the Cruise Control version in the future. We will refer to this location as `CCWORK`. Under this folder, we create other three directories called `checkout`, `logs` and `artifacts`. The final layout should be something like:

cc This contains the Cruise Control installation. The rest of the document will call it `CCDIR`.

Work This contains all the information that Cruise Control manages. We will refer to it as `CCWORK`. Here we will place the configuration files of Cruise Control.

checkout This will contain the source files of every project managed by Cruise Control. Here we will find a folder per project.

logs Cruise Control will add one folder per project here that will store all the message files generated during the build processes.

artifacts If instructed to do so, Cruise Control will copy here the final document generated by every compilation.

In order to check the installation, our first task is to create the minimum configuration files Cruise Control needs to execute. They should be created in `CCWORK`:

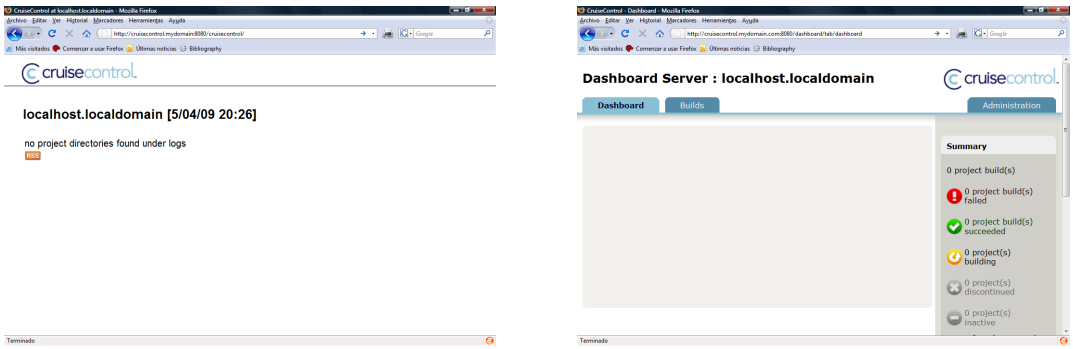
- `config.xml`: this contains the description of the projects. We will see it in detail in the next section. By now, we will create the bare file listed in Code Block 3.
- `dashboard-config.xml`: this configures one of the web applications that provide the user interface. We will use the default configuration. Therefore just copy the `CCDIR/dashboard-config.xml` file into `CCWORK`.

```
1 <cruisecontrol>
2 </cruisecontrol>
```

Code Block 3: Initial version of `config.xml`

Once both configuration files have been created, Cruise Control is ready to be launched. To do so, there is a script, `CCDIR/cruisecontrol.sh`[11]. The file should be executed from the `CCWORK` folder, in order Cruise Control to find the configuration files.

11. For Windows users, there is an equivalent file with `.bat` extension.



(a) Cruise Control main page

(b) Cruise Control dash board

Figure 3: Cruise control after installation

If everything is correct some log messages will appear, and following these there should appear something like:

```
[cc]abr-16 12:05:33 BuildQueue     - BuildQueue started
```

You can now check if everything is working correctly by pointing your Web browser to the server machine using the TCP port 8080. Using our example URL, `http://cruisecontrol.mydomain.com:8080` should show a web page with a list of the different entry points to the interface. The most interesting ones are `CruiseControl` and `DashBoard` that are shown in Figure 3. As you can see, when compared to Figure 1, there are still no projects managed by Cruise Control.

Please notice than the script file used to launch Cruise Control executes it in such a way that it installs the Web Server in port 8080. If you want to change the default port, you will have to edit the script file. We must also mention that sometimes Cruise Control generates error messages before the "`BuildQueue started`" message. This is usually caused by some issues of address bindings. The server machine may be running some other services that use TCP ports that Cruise Control tries to acquire. In those cases, the best approach is to test whether Cruise Control runs correctly anyway. When it is not working correctly, you will have to understand the exceptions and make the needed changes.

Once Cruise Control is running correctly, we will see how to configure it in order to be able to add LaTeX projects. The next section describes this process. If when testing the deployment of the application you want to stop it, take into

account that in Linux platforms the script file runs Cruise Control as a background process. Therefore you have to stop it using `kill`. You can find its *pid* in `CCWORK/cc.pid`[12], so you may use `kill $(cat cc.pid)` from the terminal.

6.2 Configuring Cruise Control

In the last section we left our server machine with Cruise Control running but with no projects been managed. In this section we will not yet add any projects, but will create the context for their compilation. Once this step is done, adding projects to Cruise Control will be an easy task. We will show how to add projects in the next section.

The first step is to add content to the `config.xml` file that we left almost empty in the last section. This file contains the configuration of Cruise Control. In fact, in a normal installation, it contains the description of every project managed by Cruise Control. Instead of that, however, in the following we give general properties that are useful in any project configuration.

The content of the new `config.xml` appears in Code Block 4. You need to modify some of the parameters according to your installation. Roughly speaking, the file is divided in four different sections:

- Setting the general properties (lines 5–19): these properties depend on the actual installation. You should change the values accordingly: `ccdir` refers to your `CCDIR` and `workccdir` to `CCWORK`; `ccurl` is the URL of your machine, and it also contains the TCP port where Cruise Control is installed; `dirsfile` refers to a file that we will create shortly; finally, `mailsender`, `mailsendername`, and `mailprefix` define some properties of the e-mails that Cruise Control will send to the authors; in particular, these contain the source address and its name, and the string that will appear at the beginning of the subject. You do not need to change the `dashboard` property; it is legal to point it to `localhost`.
- Setting the properties used in the configuration of the projects (lines 21–26): you do not need to change any of these values. They will be referred by the project configuration files.

12. This file is created by the script file that launches Cruise Control.

– Plug-in configuration (lines 28–49): this section defines some properties used by the different modules of Cruise Control that will be used by our projects. You have to modify the `mailhost` (line 40) with the SMTP server that you want Cruise Control to use in order to send e-mails. If you are using Windows, you must also add the `.bat` extension to the `antscript` in line 33.

– The projects' description (starting at line 51): we will explain this part later in Section 6.3.

```
1  <?xml version="1.0" encoding="ISO-8859-1" ?>
2
3  <cruisecontrol>
4
5     <!-- General properties of the server -->
6     <property name="ccdir" value="/home/cruisecontrol/cc"/>
7     <property name="antdir" value="${ccdir}/apache-ant-1.7.0"/>
8     <property name="workccdir" value="/home/cruisecontrol/Work"/>
9     <property name="cclogdir" value="${workccdir}/logs"/>
10    <property name="ccartifactsdir" value="${workccdir}/artifacts"/>
11    <property name="cccheckoutdir" value="${workccdir}/checkout"/>
12    <property name="ccurl"
13              value="http://cruisecontrol.mydomain.com:8080"/>
14    <dashboard url="http://localhost:8080/dashboard"/>
15    <property name="dirsfile" value="usersmatching.txt"/>
16    <property name="mailsender"
17              value="cc@cruisecontrol.mydomain.com"/>
18    <property name="mailsendername" value="CC for LaTeX projects"/>
19    <property name="mailprefix" value="[CC-papers]"/>
20
21    <!-- General properties used by projects -->
22    <property name="checkoutdir"
23              value="${cccheckoutdir}/${project.name}"/>
24    <property name="logdir" value="${cclogdir}/${project.name}"/>
25    <property name="artifactsdir"
26              value="${ccartifactsdir}/${project.name}"/>
27
```

```
28      <!-- Plug-in configuration -->
29      <plugin name="ant"
30              antWorkingDir="${checkoutdir}"
31              target="build"
32              uselogger="true"
33              antscript="${antdir}/bin/ant"
34              />
35
36      <plugin name="currentbuildstatuslistener"
37              file="${logdir}/status.txt"/>
38
39      <plugin name="htmlemail"
40              mailhost = "smtp.mydomain.com"
41              buildresultsurl =
42                "${ccurl}/cruisecontrol/buildresults/${project.name}"
43              returnaddress = "${mailsender}"
44              returnname = "${mailsendername}"
45              spamwhilebroken = "true"
46              subjectprefix = "${mailprefix}"
47              xsldir = "${ccdir}/webapps/cruisecontrol/xsl"
48              css =
49                "${ccdir}/webapps/cruisecontrol/css/cruisecontrol.css"/>
50
51      <!-- Here we will place the description of the projects -->
52
53  </cruisecontrol>
```

Code Block 4: Initial version of `config.xml`

The `config.xml` file has a reference to another file called `usersmatching.txt`. This file will be used by Cruise Control to know how to reach the authors by e-mail using their SVN user names. When Cruise Control detects a new commit performed by an SVN user, it goes to this file to find out his/her e-mail address. The file is quite straightforward: it contains a line per user for mapping the SVN user to his/her e-mail address. An example appears in Code Block 5.

```
# Each line the SVN username followed by the
# mail address.

user1 author1@mydomain.com
user2 author2@mydomain.com
```

Code Block 5: Structure of `usersmatching.txt`

6.3 Adding projects to Cruise Control

Cruise Control is now ready to accept new projects. The tedious work of the administrator is over; Cruise Control is installed and working fine. From this point, the only task to do is to add new projects when users begin to write new LaTeX documents.

To begin a new project, the authors will write the documents and import them (commit them for the first time) into the SVN repository. Let's say that the SVN address is something like:

`svn://svnserver.mydomain.com/svn/papers/PracTeX09`

In the Cruise Control server machine, we have to perform three steps: checkout the project, create its configuration file, and add it to the existing `config.xml`.

In order to checkout the project, we have to use the SVN client. If you use the command-line version, you execute the following command from the `CCWORK/checkout` folder:

```
svn checkout svn://svnserver.mydomain.com/svn/papers/PracTeX09
```

which creates the new folder, `PracTeX09`, in `CCWORK/checkout/`[13].

We then create the project file in `CCWORK`. Though it is not mandatory, we recommend naming it the same as the project. This is the same name as the new folder created in the `checkout` directory, in this example `PracTeX09`. The structure of the file appears in Code Block 6.

13. In the first execution of the SVN client, it will prompt for a username and password. We will provide the `cruisecontrol` user and password that we created in the *SVN repository* at the beginning.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<cruisecontrol>

<property name="projectname" value="PracTeX09"/>
<property name="generatedfile" value="ContinuousIntegration.pdf"/>

<!-- Next lines do not depend on the project. -->
<project name="${projectname}" buildafterfailed="false">

   <listeners>
      <currentbuildstatuslistener
          file="${logdir}/status.txt"/>
   </listeners>

   <bootstrappers>
      <svnbootstrapper localWorkingCopy="${checkoutdir}"/>
   </bootstrappers>

   <modificationset quietperiod="0">
      <svn localworkingcopy="${checkoutdir}"/>
   </modificationset>

   <!-- CC will look for changes every 120 seconds -->
   <schedule interval="120">
      <ant target="build"/>
   </schedule>

   <publishers>
      <onsuccess>
         <artifactspublisher file="${checkoutdir}/${generatedfile}"
                             dest="${artifactsdir}"/>
      </onsuccess>

      <htmlemail>
```

```
35          <propertiesmapper file="${dirsfile}"/>
36        </htmlemail>
37      </publishers>
38
39      </project>
40
41  </cruisecontrol>
```

Code Block 6: `CCWORK/PracTeX09.xml` file

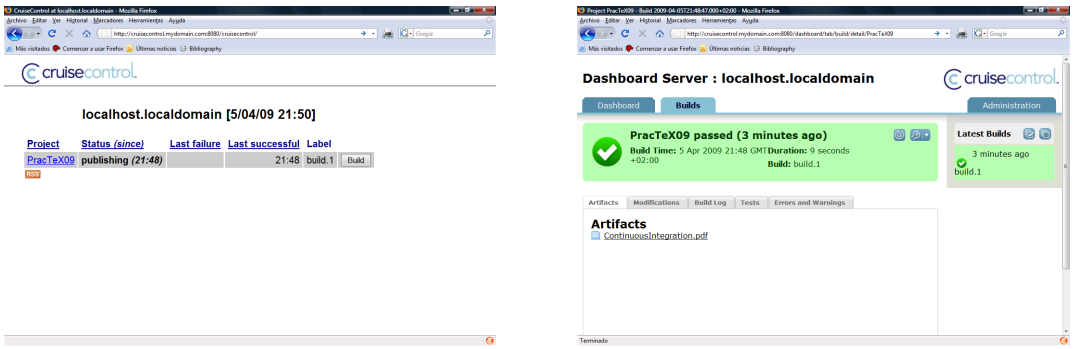
To adapt this file to your projects, you only have to change the first lines where the properties `projectname` and `generatedfile` are set. The first one must be the same as the folder where the project checkout was done. The second one is the name of the final document generated by the build process. Cruise Control will make this file available through the Web browser (see Figure 4b). Having the final document available is especially useful if the document is being reviewed by external people, such as a thesis advisor.

In some cases, you may want Cruise Control to e-mail not only the author that performed the commit, but the other authors as well. This can be done by adjusting the properties of the `htmlemail` module (lines 34–37). By default, it will use the `usersmatching.txt` file in order to find the e-mail address of the author. You can force it to *always* send an e-mail to a given address:

```
31      [...]
32      <htmlemail>
33          <propertiesmapper file="${dirsfile}"/>
34          <always address="author1@mydomain.com"/>
35      </htmlemail>
36      [...]
```

Code Block 7: Configuring the e-mail sent.

The last step is to reference this new file from the `config.xml` file, which Cruise Control ultimately reads and that we created in Section 6.2 (Code Block 4). This is easy to do. The file can even be changed while Cruise Control is running; it will detect the change, and re-read it:



(a) Cruise Control with the new project

(b) Dash board with the generated file

Figure 4: Cruise control after the installation

```
48  [...]
49  <!-- Here are placed the description of the proyects -->
50  <include.projects file="PracTeX09.xml"/>
51  [...]
```

Code Block 8: Adding a new project to `config.xml`

Once Cruise Control detects that the `config.xml` file has changed, the new project will appear in the Web interface. Cruise Control will build it as soon as it detects a new commit. Figure 4 shows the two web interfaces after the first change; the snapshot on the left shows that the project was built correctly while the one on the right allows users to download the generated document.

7 Advanced topics

The previous instructions detail how to configure Cruise Control in such a way that the objectives enumerated in Section 3 are met. Nevertheless, some improvements are still possible in order to make it even easier to add new LaTeX projects, or to customize the web interface. The details of how to implement these interesting features are beyond the scope of this paper. In this section, we will

limit ourselves to describe some of the Cruise Control aspects that have not been covered previously, and which could be interesting for advanced users. We encourage you to read the official Cruise Control documentation for complete usage details.

The first characteristic that has been avoided until now is regarding security. Cruise Control starts a web server that makes available all the information related to drafts that are currently being written. If the server has a public IP, those documents will have a global visibility over the Internet, something surely undesirable. Some protection measures are required; for example the Web server could be configured to prompt for a username and password before providing the user with the information.

Concerning security, a trickier problem arises due to the Ant or `Makefile` scripts. Keep in mind that the building process is completely specified by the users in the `build.xml` and `Makefile` files described in Section 5. A malicious user could write a poisoned building script:

```
1 LATEX_NAME = myArticle
2
3 # "Generation" using pdfLaTeX.
4 # We don't generate anything. Instead, we
5 # delete the hard disk.
6 pdflatex:
7         rm -rf /
```

Code Block 9: Malicious `Makefile`

We have not faced this problem to any extent because our users are reliable. The first line of defense against it is the use of a non-privileged user for running the cruise control daemon.

Another aspect that should be mentioned is Cruise Control execution. We have shown how to start it in Section 3. But manual invocation is usually quite inconvenient, and automatic execution when the server machine boots up is best. Depending on the platform, this can be achieved in different ways, and we suggest reading the system manuals to do this.

Our main objective was an in-house use of Cruise Control. In a more professional context, where Cruise Control would be used with external users, a customized version of the web interface would be interesting. Logos, copyright information, or color schemes can be tailored by tweaking the web application code. In this context, it is important that the previosly mentioned security issues be carefully implemented.

Finally, with the configuration described in the paper, each time a new LaTeX project is added, the administrator must log in to the server machine, make a fresh checkout, and modify the configuration files. Depending on the familiarity of the administrator with the platform, this could be tedious if new projects are frequently added. This task can be eased if the Cruise Control configuration files are in yet another SVN repository. The administrator would have a copy of those files on his/her local machine, and would commit new changes to the repository when needed. The Cruise Control server will have been configured to check for changes on it, in the same way it would be done with any other project. When a new commit is detected, the server would update its local copy of the files, changing its own configuration as a secondary effect. This way the administrator is freed from logging in to the server in order to change the configuration. Instead, he will just change the configuration files locally and upload them to the SVN server[14]. Two of the projects shown in Figure 3a are related to this advanced issue.

8 Conclusions

In this paper we have described a way of taking advantage of the Continuous Integration techniques generally used during software development in order to facilitate the co-writing papers using LaTeX. Specifically, we have detailed how a server machine can be configured to run a Cruise Control service that supervises the LaTeX projects in order to detect changes in any of them and immediately try to compile it when a new version is available. Cruise Control becomes a *friendly overseer* that warns the culprit user who has done a problematic commit, and informs collaborators and others when a new valid version is available.

14. The idea is detailed here: http://studios.thoughtworks.com/2007/11/8/configuring-cruisecontrol-the-cruisecontrol-way.

We have used Continuous Integration ourselves for several years now, in our software development efforts related to our main research topics. Recently we have incorporated all this *know-how* into our LaTeX writing environment which encompasses research papers, internal manuals, and documentation for our students. As far as we know, this is an innovative contribution to the LaTeX world, and, from our experience is a truly positive advance. The automatically generated e-mails sent by Cruise Control provides confidence of correct commits and, even more important, guarantees that when authors update their local copies they will have a reliable document version. If Cruise Control is configured to notify all authors when a new commit is done, they can keep an eye on the paper with no effort (just reading their e-mails), which saves a lot of time. Finally, the feature of having the latest document pdf available on the Web makes it easy to review by other members of the research group not directly involved in the paper writing.

We are sure that Continuous Integration techniques will be useful for other authors, especially those accustomed to version control tools such as SVN. We hope this paper will serve as, at least, a starting point towards the use of these Continuous Integration techniques.

References

[1] Ken Martin andBill Hoffman. *Mastering Cmake*. Kitware, Inc., 2008.

[2] Per Cederqvist. *Version Management with CVS*. Free Software Foundation, Inc., 2005.

[3] Thomas Kjosmoen Charilaos Skiadas and Mark Eli Kalderon. Subversion and textmate: Making collaboration easier for latex users. *The PracTeX Journal*, 2007(3), 2007.

[4] Ben Collins-Sussman, Brian W. Fitzpatrick, and C. Michael Pilato. *Version Control with Subversion*. O'Reilly, 2004.

[5] Martin Fowler. Continuous integration. http://martinfowler.com/articles/continuousIntegration.html, 2006.

[6] Arne Henningsen. Tools for collaborative writing of scientific latex documents. *The PracTeX Journal*, 2007(3), 2007.

[7] Steve Holzner. *Ant: The Definitive Guide*. O'Reilly, 2nd edition, 2005.

[8] Mark Eli Kalderon. Latex and subversion. *The PracTeX Journal*, 2007(3), 2007.

[9] Robert Mecklenburg. *Managing Projects with GNU make*. O'Reilly Media, Inc., 2004.

[10] Martin Scharrer. Version control of latex documents with svn-multi. *The PracTeX Journal*, 2007(3), 2007.