Covariant Conversions (CoCo): A Design Pattern for Type-Safe Modular Software Evolution in Object-Oriented Systems (Artifact)

Jan Bessai
Technische Universität Dortmund, Germany

George T. Heineman
Worcester Polytechnic Institute, MA, USA

Boris Düdder
University of Copenhagen, Denmark

Abstract

This artifact contains code illustrating the Covariant Conversions (CoCo) design pattern in Java, Scala, and C#. The CoCo pattern shows how to solve the expression problem in modern object-oriented languages without the need for language extensions. It structures code so that adding new classes *and* new methods is possible without changes to existing implementations. The artifact is a live image of a Linux machine for archival purposes. It allows to boot into an environment which has an IDE installed to inspect the code. Build infrastructure to compile, run, test, and benchmark the code without internet access is also included.

2012 ACM Subject Classification Software and its engineering → Software evolution; Software and its engineering → design patterns; Software and its engineering → abstraction, modeling and modularity

Keywords and phrases Expression problem, software evolution, type safety, producer method, binary method

Digital Object Identifier 10.4230/DARTS.7.2.4

Acknowledgements Creating the live image would have been impossible without the marvelous instructions by Will Haley [4].

Related Article Jan Bessai, George T. Heineman, and Boris Düdder, "Covariant Conversions (CoCo): A Design Pattern for Type-Safe Modular Software Evolution in Object-Oriented Systems", in 35th European Conference on Object-Oriented Programming (ECOOP 2021), LIPIcs, Vol. 194, pp. 4:1–4:25, 2021. `https://doi.org/10.4230/LIPIcs.ECOOP.2021.4`

Related Conference 35th European Conference on Object-Oriented Programming (ECOOP 2021), July 12–16, 2021, Aarhus, Denmark (Virtual Conference)

1 Scope

The related article presents the Covariant Conversions (CoCo) design pattern by showing an example and further evaluates its application using two case-studies. The example is a simple object-oriented model for processing XML files and implemented in Java. It is encompassed by modular client code, which is also discussed in the article. The purpose of the implementation is to provide further details to the UML diagrams presented in the article, as well as a proof for the claim that CoCo can be implemented in pure Java without additional language extensions. The first case study is an extended version of a standard expression problem example of arithmetic expressions [8]. It is implemented in Java, Scala, and C# to support the claim that the CoCo pattern works with mainstream object-oriented programming languages. The artifact also includes alternative Java implementations which use other established expression problem approaches. Tradeoffs of these approaches with CoCo become most clear when looking at their actual code and running the included benchmarks. The second case study is a Java implementation of parts

of the compiler suite from the text-book "Types and programming languages" by Pierce [6]. It serves to be able to compare CoCo with two other approaches, EVF [9] and Castor [10], both by Zhang and Oliveira.

2 Content

The artifact package includes:

- A live image to boot a Linux machine
- The code discussed above and in the accompanying article
  (in folder `/home/coco/ecoop2021artifacts` of the booted system)
- A minimal graphical environment with an IDE (VSCodium) to inspect the code
- The necessary infrastructure to compile, run, benchmark and test the code
- More detailed instructions
  (in file `/home/coco/Desktop/README.txt` of the booted system)

3 Getting the artifact

The artifact endorsed by the Artifact Evaluation Committee is available free of charge on the Dagstuhl Research Online Publication Server (DROPS). In addition, the artifact is also available at Zenodo [1] and Github [2], where direct access to the source code included in the image is possible and updates will be posted if necessary.

4 Tested platforms

You can boot the image directly on your machine, or in a virtual machine. We have tested it to work with `qemu-5.2.0` on a Linux host, starting it with

```
qemu-system-x86_64 \
  -enable-kvm -smp 2,sockets=1,cores=2,threads=1 -cpu host \
  -m 4096 -cdrom ecoop_coco.iso
```

which will cause it use use 2 cores of the host system (kvm line), and 4096 MB of RAM. Less RAM is not recommended, because performance tests might fail and the IDE might crash. At the cost of performance, less CPU cores may be used, or they may be emulated instead of using (kvm) virtualization.
The image has also been tested in VirtualBox 6 on Linux, Mac, and Windows hosts, where you need to perform the following steps:

1. Click on "New"
2. Choose a name for the machine
3. Set Type to "Linux"
4. Set Version to "Debian 64 Bit"
5. Set the memory size to 4096MB or more
6. Select "Do not add a virtual hard disk"
7. Click on Create
8. Right Click on the newly selected machine and select Settings
9. Add a new Optical Device under "Storage" by clicking on the Optical-Device-Logo next to the IDE Controller
10. Click on Add in the Medium Dialog
11. Select the `ecoop_coco.iso` image, click on "Choose" and then on "Ok"

12. Adjust the other machine settings according to your Host Hardware (e.g. adding more cores and activating acceleration)

Similar instructions apply for other virtualization technologies.

Depending on your computer it should also be possible to burn the image on a DVD or flash it on an USB-drive and directly boot it. Using Linux or similar systems the image can be copied to an external drive by

```
dd if=ecoop_coco.iso of=/dev/drive
```

where drive has to be replaced by the device file of your drive (e.g. `sdb`). Note that this will erase all data on the target device!

5 License

License information and sources are included in the image, or available on via the Devuan webpage [3], Microsoft packages [5], and the VSCodium repository [7]. The code to illustrate CoCo is available under the APACHE 2.0 license, which is included along side of it.

6 MD5 sum of the artifact

ba86f1ea3fbe5ea6d7af023f744d8288

7 Size of the artifact

1.79 GiB

A Important Notes

Note that the image is intended for archival purposes:

- It will not receive any security updates.
- It should only be booted on machines that are isolated (e.g. by a VM or physically) from any valuable data.
- It is not meant to provide a production-grade user experience.
- It does not contain translations.
- It does not have support for assistive technologies.

While you can read and write files, the image is immutable and will not store anything you do with it across reboots, unless you add and mount an additional medium.

B Further Information

The image contains a minimal set of software and is based on Devuan (ceres/unstable), which is a Debian fork. It is running with a Linux 5.10.6 kernel and boots into a graphical environment (XServer). If the XServer does not start, you might try selecting “no modesetting” at the boot loader. When still dropped to the console, you can login using username “coco” with password “coco” or “root” with the same password. The XServer can be started manually by typing in “startx” as user “coco”. This can help to debug error messages.

References

1. Jan Bessai, George T. Heineman, and Boris Düdder. Covariant conversions (coco): A design pattern for type-safe modular software evolution in object-oriented systems (artifact), 2021. doi:10.5281/zenodo.4756838.
2. Jan Bessai, George T. Heineman, and Boris Düdder. Github repository for this artifact, 2021. URL: https://github.com/JanBessai/ecoop2021artifacts.
3. Devuan GNU+Linux, 2021. URL: https://www.devuan.org/.
4. Will Haley. Create a custom debian live environment (CD or USB), 2020. URL: https://willhaley.com/blog/custom-debian-live-environment/.
5. Linux software repository for microsoft products, 2021. URL: https://docs.microsoft.com/en-us/windows-server/administration/linux-package-repository-for-microsoft-software.
6. Benjamin C. Pierce. *Types and programming languages*. MIT Press, 2002. URL: https://www.cis.upenn.edu/~bcpierce/tapl/.
7. Pavlo Rudy. VSCodium Repository, 2021. URL: https://gitlab.com/paulcarroty/vscodium-deb-rpm-repo.
8. Philip Wadler. The expression problem, 1998. E-Mail to the Java Genericity Mailing List. URL: http://homepages.inf.ed.ac.uk/wadler/papers/expression/expression.txt.
9. Weixin Zhang and Bruno C. d. S. Oliveira. EVF: an extensible and expressive visitor framework for programming language reuse. In Peter Müller, editor, *31st European Conference on Object-Oriented Programming, ECOOP 2017, June 19-23, 2017, Barcelona, Spain*, volume 74 of *LIPIcs*, pages 29:1–29:32. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ECOOP.2017.29.
10. Weixin Zhang and Bruno C. d. S. Oliveira. CASTOR: Programming with extensible generative visitors. *Sci. Comput. Program.*, 193:102449, 2020. doi:10.1016/j.scico.2020.102449.