


# Contract Systems Need Domain-Specific Notations (Artifact)

Cameron Moy ✉ 

PLT @ Northeastern University, Boston, MA, USA

Ryan Jung<sup>1</sup> ✉ 

Northeastern University, Boston, MA, USA

Matthias Felleisen ✉ 

PLT @ Northeastern University, Boston, MA, USA

## Abstract

Contract systems enable programmers to state specifications and have them enforced at run time. First-order contracts are expressed using ordinary code, while higher-order contracts are expressed using the notation familiar from type systems. Most interface descriptions, though, come with properties that involve not just assertions about single method calls, but entire call chains. Typical contract systems

cannot express these specifications concisely. Such specifications demand domain-specific notations. In response, the related article proposes that contract systems abstract over the notation used for stating specifications. It presents an architecture for such a system, some illustrative examples, and an evaluation in terms of common notations from the literature.

**2012 ACM Subject Classification** Software and its engineering → Domain specific languages

**Keywords and phrases** software contracts, domain-specific languages

**Digital Object Identifier** 10.4230/DARTS.11.2.4

**Funding** This work was supported by NSF grant SHF 2116372.

**Related Article** Cameron Moy, Ryan Jung, and Matthias Felleisen, “Contract Systems Need Domain-Specific Notations”, in 39th European Conference on Object-Oriented Programming (ECOOP 2025), LIPIcs, Vol. 333, pp. 42:1–42:24, 2025. <https://doi.org/10.4230/LIPIcs.ECOOP.2025.42>

**Related Conference** 39th European Conference on Object-Oriented Programming (ECOOP 2025), June 30–July 2, 2025, Bergen, Norway

**Evaluation Policy** The artifact has been evaluated as described in the ECOOP 2025 Call for Artifacts and the ACM Artifact Review and Badging Policy.

## 1 Scope

The article proposes a new architecture for contract systems that are abstracted over domain-specific notations for specifications. This artifact is intended to provide the software necessary to validate that the examples given in the paper run and that the libraries provided work properly. There is no experimental evaluation.

## 2 Content

The Docker image is based on Guix System and contains only a very limited number of tools: bash, coreutils, nano, and vim. The artifact package includes two Racket libraries:

- `trace-contract` that provides the trace contract library, and
- `logic` that provides several logic DSLs.

<sup>1</sup> Currently working in industry.



### **3 Getting the artifact**

Run the follow commands to get the artifact:

```
$ wget https://zenodo.org/records/15285740/files/dsc-artifact.tar.gz
$ docker load --input dsc-artifact.tar.gz
$ docker run -p 8080:8080 -ti dsc-artifact
```

You can validate that the tests run correctly for both the trace contract library and the logic library:

```
$ raco test trace-contract/trace-contract
$ raco test logic/logic
```

To read the documentation, please run the following:

```
$ racket serve.rkt
```

From your host operating system, you should be able to navigate to the following URLs to view the documentation:

- <http://localhost:8080/trace-contract>
- <http://localhost:8080/logic>

The TCP example from the paper can be viewed and run as such:

```
$ cat trace-contract/trace-contract/example/listener.rkt
$ raco test trace-contract/trace-contract/example/listener.rkt
```

For the artifact assessment phase, reviewers should check that documentation is sufficient for reusability.

### **4 Tested platforms**

The artifact has been tested on an x86-based Linux machine. No specific resources are required other than conventional hardware.

### **5 License**

The artifact is available under the Apache 2.0 license.

### **6 MD5 sum of the artifact**

2544798173d4e73a99d1891d7b1e34b7

### **7 Size of the artifact**

820M