




CoVeriTeam Service: Verification as a Service

Dirk Beyer 
LMU Munich, Germany

Sudeep Kanav 
LMU Munich, Germany

Henrik Wachowitz 
LMU Munich, Germany

Abstract—The research community has developed numerous tools for solving verification problems, but we are missing a common web interface for executing them. This means, users have to commit to install and execute each new tool (version) on their local machine. We propose to use CoVeriTeam Service to make it easy for verification researchers to experiment with new verification tools. CoVeriTeam has already unified the command-line interface, and reduced the burden by taking care of tool installation and isolated execution. The new web service in addition enables tool developers to make their tools accessible on the web and users to include verification tools in their work flow. There are already further applications of our service: The 2023 competitions on software verification and testing used the service for their integration testing, and we propose to use CoVeriTeam Service for incremental verification as part of a continuous-integration process.

Demonstration video: <https://youtu.be/0Ao0ZogSu1U>

Demonstration service: <https://coveriteam-service.sosy-lab.org>

Index Terms—Cooperative Verification, Tool Development, Incremental Verification, Software Verification, Automatic Verification, Verification Tools, Web Service, API, Continuous Integration

I. INTRODUCTION

Numerous automated verification and testing tools have been developed in the last few decades. This is attested by the growing number of participants in the competitions on software-verification (SV-COMP) [5] and testing (Test-Comp) [4].

Consider a verification researcher or student who wants to experiment with a verification tool. On the one hand, a larger number of tools provides more opportunities to such a user. On the other hand, it requires considerable effort to figure out how to execute each tool and interpret the results. Additionally, there might be a mismatch between the configuration of the user’s system and the required configuration to execute the tool. Or, the user might not want to execute an untrusted verification tool locally due to security concerns. Even with numerous choices available for the verification tools, it is not straightforward to use them.

There are arguments regarding the developers of verification tools pointing in the same direction. A lot of effort goes in developing the tools; but even after spending so much effort in developing excellent tools, the tools are not easily accessible.

This work aims at improving the accessibility of automated verification tools. We propose a web service that enables remote execution of verification tools. The solution is based on CoVeriTeam [7], which provides already a common command-line interface to verification tools. A common web interface makes it easier for users to experiment with arbitrary verification tools, as well as for tool developers to benefit from making their tools more accessible to new users. The web service

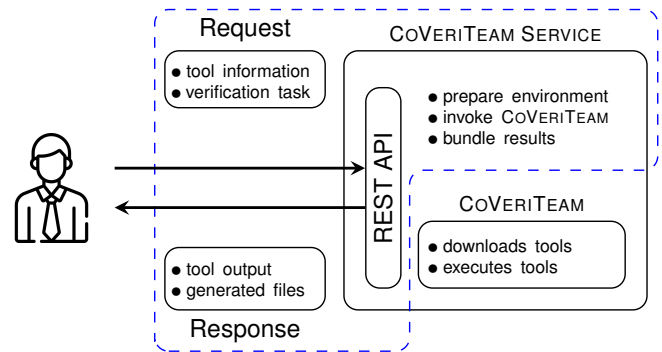


Fig. 1: Abstract view of the CoVeriTeam SERVICE

liberates users from the concerns of configuring the local system. Additionally, running tools remotely eliminates security concerns arising from local execution of untrusted code.

CoVeriTeam [7] provides a common interface to verification tools through the means of verification actors and artifacts. It manages download, execution, and processing input and output of these tools. It provides a library of actors for many publicly available verification and testing tools. Also, it provides means to integrate a new tool easily via BENCHEXEC [16].

Developing a web service for CoVeriTeam allows us to reuse the infrastructure that was developed for CoVeriTeam. If users want to execute a verification tool remotely, they can simply invoke CoVeriTeam with the required inputs and the option for remote execution, and CoVeriTeam manages the rest. Alternatively, the service can be called directly via its REST API, making it suitable for integration into other applications. Figure 1 shows the abstract view of our solution. A user sends the information about the tool and the verification task to the service; the service prepares the environment, invokes CoVeriTeam, and bundles the results of the execution; and CoVeriTeam manages tool download and its execution.

Contributions. We make the following contributions:

- 1) *Remote execution*: a web service to remotely execute publicly available verification tools and their combinations,
- 2) *Incremental verifier*: a web service for incremental verification using CoVeriTeam SERVICE as a micro-service,
- 3) *Easy access to verification tools*: a solution allowing tool developers to provide easy access to their verification tools,
- 4) *Reuse*: an open-source Python implementation and a reproduction package [11, 12].

Impact. This service was part of the continuous-integration pipeline of the competitions SV-COMP and Test-Comp 2023. The service was used to make sure that a (new version of a)

tool is only integrated if it can be successfully executed in the specified competition environment. This makes it possible to find issues with the participating tool archive early on, thus saving the effort required by the competition organizer and participants to locate and solve the issues by manual inspection.

Related Work. *Electronic Tools Integration* (ETI) [23, 25] was conceived as a platform to allow users to access tools through the internet. It was intended to serve as a site for testing, presenting, evaluation, and benchmarking tools.

Model Checking as a Service (MCaaS) [21] aims to hide the model checking entirely from engineers. The authors aim to provide a ready made solution for a specific engineering workflow with limited configuration.

The (deprecated) RiSE4Fun [3] service is a web front-end to web services on formal methods. CoVeriTeam Service is not a web front-end, but a backend service. It could have been integrated as one of the backend services in RiSE4Fun, acting as a generic interface to many verification tools, similar to our own web UI.

Unite [26] aims to provide easier access to analysis tools by converting them to web services. It uses *open services for lifecycle collaboration* (OSLC) to create web services from software-analysis tools. While ETI, MCaaS, and our service provide a remote interface to analysis tools, the focus of Unite is to provide a framework to setup a web service for *one tool at a time*. This requires manual setup and configuration of the tool and its arguments. Once configured, the OSLC endpoint integrates with respective OSLC clients that are able to mirror the tool’s interface.

Our approach decouples (a) integrating a new tool from (b) providing a web service: (a) COVERITEAM integrates the analysis tools and provides the interface to the tools; (b) COVERITEAM SERVICE provides a web-service interface using a REST API. The interface to our web service is uniform, regardless of the tool a user wants to execute.

Many verification tools have implemented some form of web front-end [6, 18, 20, 22], stating a clear demand for web-accessibility. With COVERITEAM SERVICE, we provide one uniform interface for all tools. We present a web UI in Sect. IV, where users can already select from 21 verification tools and run them remotely.

Running Example. Users want to verify a program. They want to use a verification tool to achieve it; but they do not want to execute the tool locally. There could be multiple reasons for the aversion to the local execution of the tools, e.g., ease of use: the user does not want to install the tool and its dependencies, or the tool might require different packages than available on the local machine, or security concerns: the user does not want to run an untrusted tool on the local machine. Our running example considers the case that our user wants to execute the verification tool CPACHECKER remotely.

II. BACKGROUND: COVERITEAM

COVERITEAM SERVICE uses COVERITEAM as execution backend. COVERITEAM [7, 10] provides a common interface to verification

```
// Create verifier from an actor definition
verifier = ActorFactory.create(ProgramVerifier,
    "cpachecker.yml", "2.1");
// Prepare inputs
prog = ArtifactFactory.create(Program,
    "test02.c", ILP32);
spec =
    ArtifactFactory.create(BehaviorSpecification,
        "unreach-call.prp");
inputs = {"program":prog, "spec":spec};

// Execute the verifier on the inputs
result = execute(verifier, inputs);
```

Listing 1: A COVERITEAM program to execute a verifier

```
actor_name: "cpachecker"
toolinfo_module: "cpachecker.py"
archives:
  - version: "2.1"
    doi: "10.5281/zenodo.5720557"
    options: ["-svcomp22", "-timelimit", "900_s"]
```

Listing 2: An example verification actor in YAML format

tools. It considers verification tools as *verification actors*, and the input and output of these tools as *verification artifacts*. The behavior of COVERITEAM is defined by a COVERITEAM program, written in a simple domain-specific language for the construction and execution of tool combinations.

Listing 1 shows an example COVERITEAM program (.cvt file), which constructs first an actor of type ProgramVerifier from an external-actor definition that is provided in file cpachecker.yml. Then it creates two artifacts of types Program and BehaviorSpecification that are prepared as input mapping for the execution of the verifier on these inputs. COVERITEAM downloads the tool, assembles the command for the tool execution, executes the tool (in a controlled and isolated BENCHEXEC [16] container), and processes the output generated by the tool to extract the output artifacts.

Listing 2 shows the external-actor definition (.yml file) that was used in the above COVERITEAM program. (Internal actors are not considered here: they are the combinations constructed in the COVERITEAM program.) An external-actor definition specifies the name of the actor, the BENCHEXEC [16] tool-info module that is used to assemble the tool command line and parse the output produced by the tool, the version, the tool archive (by DOI or URL), command-line options, and resource limits to enforce during the execution.

External-actor definitions can be defined by the developers of the external verification tools and serve as the interface to their verification tools (see COVERITEAM’s [library of external actors](#)).

III. APPROACH: COVERITEAM SERVICE

COVERITEAM SERVICE and the service for incremental verification are based on a REST API. The client communicates with the services using HTTP.

COVERITEAM SERVICE. There are three ways to execute a job using COVERITEAM SERVICE: (1) via COVERITEAM, by using the same command line, just with an additional option `--remote` appended, (2) via HTTP, by writing a job specification in a JSON file (Listing 3) and send a POST request to COVERITEAM

```

{
  "cvt_program": "verifier.cvt",
  "coveriteam_inputs": {
    "verifier_path": "cpachecker.yml",
    "program_path": "test02.c",
    "specification_path": "unreach-call.prp",
  },
  "working_directory": "coveriteam/examples",
}

```

Listing 3: Example JSON showing the input for the service

```

curl \
  --form "args=<listing3.json" \
  --form cpachecker.yml=@cpachecker.yml \
  --form test02.c=@test02.c \
  --form verifier.cvt=@verifier.cvt \
  --form unreach-call.prp=@unreach-call.prp \
  --output cvt_remote_output.zip \
  https://coveriteam-service.sosy-lab.org/execute

```

Listing 4: Example cURL request for the service

SERVICE, e.g., via cURL (Listing 4), or (3) via the web UI, by visiting the service using a web browser and completing the form.

COVERITEAM SERVICE receives the input via one of the three ways, performs consistency checks on the input, assembles the command for COVERITEAM, executes the given COVERITEAM program on the server, and sends back the output artifacts.

Service for Incremental Verification. While tests and lightweight linters are regularly used in continuous integration (CI), verification tools are more difficult to integrate, due to their resource requirements. Our service makes it possible to delegate the verification load from the CI machine to a dedicated machine for the verification service via COVERITEAM SERVICE. Furthermore, incremental verification (IV) aims at speeding up the verification of the current program version by reusing knowledge from verifying a previous version [15, 24].

We offer a solution that combines the two approaches: a *service* for *incremental* verification. The solution is based on the following components: (1) a verifier that can export and import the knowledge learned during a verification run, (2) a store to save and retrieve this information, and (3) a *manager* to connect the above two. In our solution, COVERITEAM SERVICE is used as a micro-service to execute the verifier; and the service for IV manages the store and interacts with COVERITEAM SERVICE.

Figure 2 illustrates the data flow of our solution. The service for incremental verification stores and retrieves the knowledge

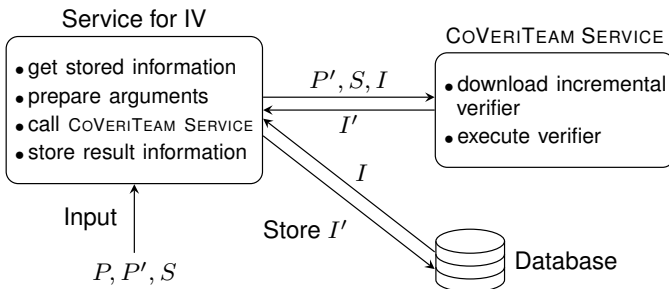


Fig. 2: Architecture of the service for incremental verification

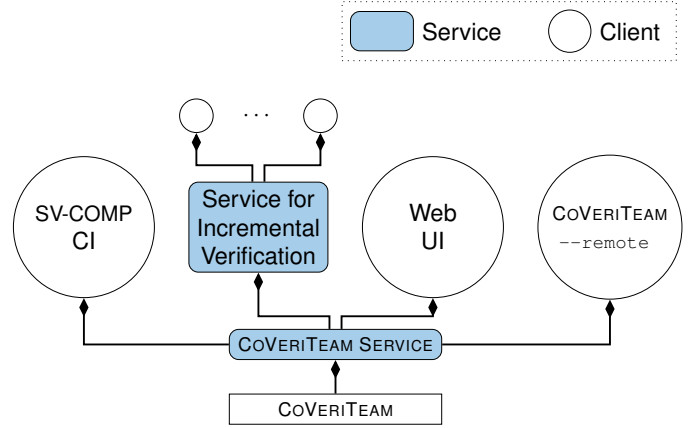


Fig. 3: Various clients of the COVERITEAM service

exported by the verification tool so that it can be reused later to assist in verification. The service takes as input a current program P' , its previous version P , and the specification S . It first checks if some reusable verification information I for P and S is stored in the database. If available, it retrieves the stored information and calls COVERITEAM SERVICE passing it the current program P' , the specification S , and the verification information I . If the information is absent, then the task is treated as a fresh verification task. COVERITEAM SERVICE then executes the verifier, and returns the result. Before returning this result to the user, the service for incremental verification extracts and stores relevant information obtained from the result.

We explain an instantiation of this construction in Sect. IV using CPACHECKER as the verifier. It is straightforward to adapt this to another verifier that supports export and import of any information reusable for verification.

IV. APPLICATIONS AND USE CASES

A detailed evaluation of COVERITEAM through case studies and experiments has been presented in our previous work [7, 8]. Here we present some applications and use cases of the service.

Remote Execution of Verification Tools. Using our publicly available installation of a COVERITEAM SERVICE instance at <https://coveriteam-service.sosy-lab.org>, a user can execute verification tools and their combinations remotely without the need to install and execute them locally. The service can be called using (1) the client integrated in COVERITEAM, by simply adding the option `--remote` to a command (this option instructs COVERITEAM to execute *remotely* via the service, whereas without this option execution happens *locally*), (2) a cURL command (Listing 4), or (3) a web page allowing users to execute tools that are included in COVERITEAM's library of external actors. Figure 3 shows the currently implemented clients of the service.

This service has been hit more than 930 times within a month of going online. Many of these hits were from the participants of SV-COMP and Test-Comp who wanted to test their tools on a machine similar to the ones used in the competition.

Continuous Integration for Competitions. The service has been used in SV-COMP and Test-Comp this year as part of the continuous-integration (CI) pipeline of the competition

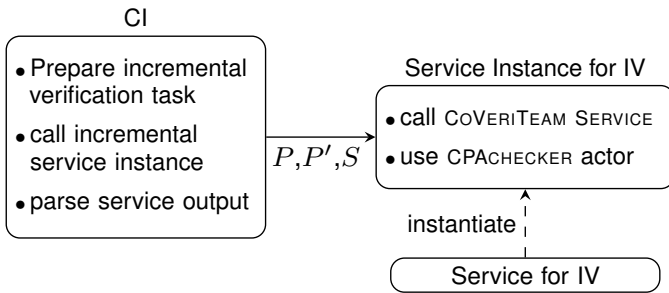


Fig. 4: Workflow of CI for incremental verification

repositories. Submitting a zipped tool archive triggers the CI pipeline, which calls COVERTTEAM SERVICE to execute the tool on test programs. The feedback from the CI enables the participants to fix issues with their tool archives without asking the competition organizer for test runs. This CI has helped reducing the workload of the competition organizer and participants for debugging issues with tool executions.

Additionally, tool developers can integrate a call to this service in their own CI. This ensures that the tool archive of the latest version is readily executable on a different machine (e.g., a competition environment), thus reducing the effort required for packaging and testing the tool for reproducibility.

CI for Incremental Verification. Software is developed incrementally. It is a common practice to integrate tests and linters in CI scripts that are triggered either on each commit to the source code or in regular intervals, to control software quality.

Formal verification is a candidate for inclusion in the CI as well, but generally it is too resource-intensive to be executed frequently. Moreover, many open-source projects execute their CI using free service plans, which provide limited computational resources. COVERTTEAM SERVICE and our service for incremental verification can address this problem by delegating resource-intensive computations to a separate service.

To evaluate our approach, we used CPACHECKER as a backend verifier in the service for incremental verification. Figure 4 illustrates the workflow of this use case. A user commits some changes to program P , yielding the modified program P' . The CI script first prepares the incremental verification task, i.e., assembles a request containing P , P' , and the specification S , and second calls the service for incremental verification to solve this incremental-verification task.

V. INTEGRATION OF TOOLS

Easy Integration of Verification Tools. To integrate a new verification tool into COVERTTEAM SERVICE, one only needs to integrate it into COVERTTEAM, i.e., create a tool-info module and an external-actor definition. Many tools are already integrated because they are readily available in COVERTTEAM [7].

Construct Service for Verification Tools. We envision COVERTTEAM SERVICE to be instantiated by tool developers to provide easy access to their own tools, on their own server, independently from our instance of COVERTTEAM SERVICE. To achieve this, one needs to create a container image or a virtual

machine with the required operating system and packages, and host COVERTTEAM SERVICE in it. This would allow an arbitrary user to connect to their new instance of COVERTTEAM SERVICE to execute the verification tool under consideration. We have tried to make hosting the service straightforward, and provide instructions and the required configuration files (see Sect. VI).

Limitations. Our service is based on COVERTTEAM [7] and BENCHEXEC [16], which require a GNU/Linux-based system providing access to cgroups. The tools from SV-COMP and Test-Comp are only a few examples of the supported tools.

VI. HOSTING AN INSTANCE

We provide three different options to create and host an instance of COVERTTEAM SERVICE:

- 1) Launching a virtual machine (VM): The repository includes a Vagrantfile, which can be used by Vagrant [19] to create a fresh VM with the service installed. Additionally, we provide a ready-to-use VM as artifact.
- 2) Launching a container: We provide a Containerfile and launch scripts in the repository. Using a container manager like PODMAN [2] (or DOCKER [1]), an image containing COVERTTEAM SERVICE can quickly be created and started.
- 3) Integrating with an existing web server: The service can be integrated with WSGI-compatible web servers [17]. The repository contains an example integrating the service with an Apache web server.

VII. CONCLUSION

Many excellent verification tools are publicly available, but installation requirements as well as resource limitations hinder the integration of verification tools into development processes. We presented COVERTTEAM SERVICE, a web service for software verification. Our solution aims to ease the effort required by a user to start working with a verification tool, supports continuous integration by enabling delegation to a service, and provides a mechanism to tool developers to make their tools easily accessible. The service has already found a user group: 930 hits within the first month of service. The continuous-integration pipeline of the competitions SV-COMP and Test-Comp is based on our new web service.

DECLARATIONS

Data-Availability Statement. We used COVERTTEAM [10, 13] version 1.0 and COVERTTEAM SERVICE [11, 14] version 1.1, which are both open source and released under the Apache 2 license. We also host an instance of the service publicly available for experimentation: <https://coveriteam-service.sosy-lab.org>. A demo of the service is available on YouTube [9]. A ready to use reproduction package containing a virtual machine hosting the service is available at Zenodo [12].

Funding Statement. This work was funded by the Deutsche Forschungsgesellschaft (DFG) — 378803395 (ConVeY).

Acknowledgement. We thank Nian-Ze Lee for the valuable feedback on this article, Klara Cimbalnik for the implementation of the Web UI, and the SV-COMP community for their valuable feedback on experimenting with COVERTTEAM SERVICE.

REFERENCES

- [1] Docker. <https://www.docker.com/>, accessed: 2023-02-09
- [2] Podman. <https://github.com/containers/podman>, accessed: 2023-02-09
- [3] Ball, T., de Halleux, P., Swamy, N., Leijen, D.: Increasing human-tool interaction via the web. In: Proc. PASTE. pp. 49–52. ACM (2013). doi:10.1145/2462029.2462031
- [4] Beyer, D.: Advances in automatic software testing: Test-Comp 2022. In: Proc. FASE. pp. 321–335. LNCS 13241, Springer (2022). doi:10.1007/978-3-030-99429-7_18
- [5] Beyer, D.: Progress on software verification: SV-COMP 2022. In: Proc. TACAS (2). pp. 375–402. LNCS 13244, Springer (2022). doi:10.1007/978-3-030-99527-0_20
- [6] Beyer, D., Dresler, G., Wendler, P.: Software verification in the Google App-Engine cloud. In: Proc. CAV. pp. 327–333. LNCS 8559, Springer (2014). doi:10.1007/978-3-319-08867-9_21
- [7] Beyer, D., Kanav, S.: CoVeriTEAM: On-demand composition of cooperative verification systems. In: Proc. TACAS. pp. 561–579. LNCS 13243, Springer (2022). doi:10.1007/978-3-030-99524-9_31
- [8] Beyer, D., Kanav, S., Richter, C.: Construction of verifier combinations based on off-the-shelf verifiers. In: Proc. FASE. pp. 49–70. Springer (2022). doi:10.1007/978-3-030-99429-7_3
- [9] Beyer, D., Kanav, S., Wachowitz, H.: Demonstration video of CoVeriTEAM SERVICE. <https://youtu.be/0Ao0ZogSu1U>, accessed: 2023-02-16
- [10] Beyer, D., Kanav, S., Wachowitz, H.: Source-code repository of CoVeriTEAM. <https://gitlab.com/sosy-lab/software/coveriteam>, accessed: 2023-02-09
- [11] Beyer, D., Kanav, S., Wachowitz, H.: Source-code repository of CoVeriTEAM SERVICE. <https://gitlab.com/sosy-lab/software/coveriteam-service>, accessed: 2023-02-09
- [12] Beyer, D., Kanav, S., Wachowitz, H.: Reproduction package for the ICSE2023 article ‘CoVeriTeam service: Verification as a service’. Zenodo (2023). doi:10.5281/zenodo.7635848
- [13] Beyer, D., Kanav, S., Wachowitz, H.: CoVeriTEAM release 1.0. Zenodo (2023). doi:10.5281/zenodo.7635975
- [14] Beyer, D., Kanav, S., Wachowitz, H.: CoVeriTEAM Service release 1.1. Zenodo (2023). doi:10.5281/zenodo.7635969
- [15] Beyer, D., Löwe, S., Novikov, E., Stahlbauer, A., Wendler, P.: Precision reuse for efficient regression verification. In: Proc. FSE. pp. 389–399. ACM (2013). doi:10.1145/2491411.2491429
- [16] Beyer, D., Löwe, S., Wendler, P.: Reliable benchmarking: Requirements and solutions. Int. J. Softw. Technol. Transfer **21**(1), 1–29 (2019). doi:10.1007/s10009-017-0469-y
- [17] Eby, P.J.: PEP 3333 – Python web-server gateway interface v1.0.1. Tech. rep., <https://peps.python.org/pep-3333/>, accessed: 2023-02-09
- [18] Esen, Z., Rümmer, P.: TRICERA: Verifying C programs using the theory of heaps. In: Proc. FMCAD. pp. 360–391. TU Wien Academic Press (2022). doi:10.34727/2022/isbn.978-3-85448-053-2
- [19] Hashimoto, M.: Vagrant. <https://github.com/hashicorp/vagrant>, accessed: 2023-02-09
- [20] Heizmann, M., Christ, J., Dietsch, D., Ermis, E., Hoenicke, J., Lindemann, M., Nutz, A., Schilling, C., Podelski, A.: ULTIMATE AUTOMIZER with SMTInterpol (competition contribution). In: Proc. TACAS. pp. 641–643. LNCS 7795, Springer (2013). doi:10.1007/978-3-642-36742-7_53
- [21] Horváth, B., Graics, B., Hajdu, Á., Micskei, Z., Molnár, V., Ráth, I., Andolfato, L., v. Gomes, I., Karban, R.: Model checking as a service: Towards pragmatic hidden formal methods. In: Proc. MODELS Companion. pp. 37:1–37:5. ACM (2020). doi:10.1145/3417990.3421407
- [22] Macedo, N., Cunha, A., Pereira, J., Carvalho, R., Silva, R., Paiva, A.C.R., Ramalho, M.S., Silva, D.: Experiences on teaching ALLOY with an automated assessment platform. Science of Computer Programming **211**, 102690 (2021). doi:10.1016/j.scico.2021.102690
- [23] Margaria, T., Nagel, R., Steffen, B.: Remote integration and coordination of verification tools in JETI. In: Proc. ECBS. pp. 431–436 (2005). doi:10.1109/ECBS.2005.59
- [24] Rothenberg, B., Dietsch, D., Heizmann, M.: Incremental verification using trace abstraction. In: Proc. SAS. pp. 364–382. LNCS 11002, Springer (2018). doi:10.1007/978-3-319-99725-4_22
- [25] Steffen, B., Margaria, T., Braun, V.: The Electronic Tool Integration platform: Concepts and design. STTT **1**(1-2), 9–30 (1997). doi:10.1007/s100090050003
- [26] Vašíček, O., Fiedor, J., Kratochvíla, T., Bohuslav, K., Smrčka, A., Vojnar, T.: Unite: An Adapter for Transforming Analysis Tools to Web Services via OSLC. In: Proc. ESEC/FSE. ACM (2022). doi:10.1145/3540250.3558939