ICAI 2023
12th International Conference on Applied Informatics
Eger, Hungary, 2023

# Static analysis for safe software upgrade*

**Dániel Ferenci**[a], **Melinda Tóth**[a]

[a]ELTE, Eötvös Loránd University, Budapest, Hungary
{danielf,toth_m}@inf.elte.hu

## Abstract

Having applications accessible without downtime is no longer an exclusive requirement of mission-critical applications or traditional domains like communications. Running applications also require changes in the source code and upgrading the live systems. Different approaches exist depending on the used technology. Systems implemented in Erlang can take the advantage of the underlying BEAM virtual machine and can be upgraded easily. However, the source code has to be developed carefully once upgrade is needed to not introduce run-time errors during the upgrade. We are developing a method to statically check the source code for constructs that may lead to upgrade issues.

### Problem demonstration

With the ever-increasing use of e-commerce, even the owner of a simple webshop expects his site to operate without incidents all year round. Running applications also require changes, however - the need to fix security issues or add new features and changes may happen at any time. An outage while a change is applied thus results in customer dissatisfaction, or even broken SLAs, and in the end, lost revenue. This presents a demand for seamless, "zero downtime" upgrades. The facilities for such upgrades depend on the stack chosen for the development and operation of the application. These choices also determine what is possible during such an upgrade. Some tools will launch new containers running a new version of the application in question, while slowly removing the previous release and possibly leading to a lost state. Other tools allow for a more fine-grained approach, upgrading only

*Application Domain Specific Highly Reliable IT Solutions project has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the Thematic Excellence Programme TKP2020-NKA-06 (National Challenges Subprogramme) funding scheme.

the changed modules, and making state preservation possible. Errors in the use of these tools will however lead to unexpected behaviour or even downtime. This leads to the need to statically analyse the code responsible for the upgrade.

In this work, we demonstrate such a static analysis made for the language Erlang [3]. Erlang is a general-purpose, dynamically typed, functional programming language. It is designed to build distributed, concurrent, fault-tolerant computer systems. Erlang is distributed alongside the BEAM virtual machine, which compiles and runs Erlang code. Support for doing zero-downtime upgrades (in Erlang terms, "hot code loads") is built into the BEAM environment. It allows for state-preserving code changes on a module basis, in contrast to tools, like Kubernetes [5] that typically route connections to containers running new software versions. As BEAM can run only two versions of the same module simultaneously the developer has to take care during development.

Bugs in the code may lead to calls to versions no longer present in the virtual machine. The code snippet below presents a typical example of problematic code that cannot be upgraded. It shows an initialisation of a tail-recursive loop function, a standard way to develop a server in Erlang. Having function references in the loop's state is dangerous, however, as these references might become outdated during module upgrades. A careful developer will ensure that these references are fully qualified - which results in function calls using the implementations in the latest version of the module. On the left-hand side snippet, the reference at line 3 will keep its original value of the Adder function through module upgrades, eventually resulting in an error, when it references a beam code version no longer present in the virtual machine. The right-hand snippet presents a fix to this, by making a fully qualified call to the function. This, when called will reference the latest version of the adder function.

```
1 init(InitNum) ->
2   Adder = fun adder/1,
3   srv:loop({InitNum, Adder}).
```

```
1 init(InitNum) ->
2   Adder = fun server:adder/1,
3   srv:loop({InitNum, Adder}).
```

## Static analysis

The mentioned upgrade related issues can be detected before the execution of the code, during the development phase using static analysis techniques. In our particular case, we want to analyse the contents of the state passed to server loops. If the state contains function calls, they should be fully qualified, otherwise, code upgrades during the operation of the program may lead to errors. For this analysis, we are using RefactorErl [2] as a framework. RefactorErl allows for analysing source code represented and stored in its Semantic Program Graph. Once the source code is stored, the tool offers different options for analysis, through its querying interfaces. The semantic query language provides an expressive language for the programmer to analyse her code [8], but RefactorErl also provides the option to

run queries through the graph representation of the program.

During our analysis, we rely on the data flow reaching [7] of RefactorErl to determine the possible values of the state and we analyse the result.

## Related work

Apart from safe upgrades, code can be analysed for other properties that can present issues during operation. RefactorErl itself can be used to check the code for common vulnerabilities [1]. Ensuring safe upgrades is however a general problem, present across technologies. For example, Kustomize [6] includes tools for ensuring correct configurations for the popular orchestration system, Kubernetes [4].

## Conclusion

In this work, we have demonstrated a checker for unsafe local fun references in server loops in Erlang. Apart from local references in the state, other problems in the code might also impede safe upgrades. Investigating further unsafe patterns could be the topic of further studies.

# References


[1] B. Baranyai, I. Bozó, M. Tóth: *Supporting Secure Coding with RefactorErl*, Submitted to the ANNALES Universitatis Scientiarum Budapestinensis de Rolando Eötvös Nominatae Sectio Computatorica (2020).

[2] I. Bozó, D. Horpácsi, Z. Horváth, R. Kitlei, J. Köszegi, T. M., M. Tóth: *RefactorErl - Source Code Analysis and Refactoring in Erlang*, in: Proceedings of the 12th Symposium on Programming Languages and Software Tools, ISBN 978-9949-23-178-2, Tallin, Estonia, Oct. 2011, pp. 138–148.

[3] F. Cesarini, S. Thompson: *Erlang Programming: A Concurrent Approach to Software Development*, O'Reilly Media, 2009, isbn: 9780596555856.

[4] B. Copy, M. Bräger, A. P. Koufidis, E. Piselli, I. P. Barreiro: *Integrating IoT Devices Into the CERN Control and Monitoring Platform*, in: Proc. ICALEPCS'19 (New York, NY, USA), International Conference on Accelerator and Large Experimental Physics Control Systems 17, JACoW Publishing, Geneva, Switzerland, Aug. 2020, pp. 1385–1388, isbn: 978-3-95450-209-7, doi: 10.18429/JACoW-ICALEPCS2019-WEPHA125.

[5] *Kubernetes Documentation*, Accessed: 2023-01-12, url: https://kubernetes.io/docs/home/.

[6] *Kustomize Documentation*, Accessed: 2023-01-12, url: https://kubectl.docs.kubernetes.io/references/kustomize/.

[7] M. Tóth, I. Bozó: *Static Analysis of Complex Software Systems Implemented in Erlang*, Central European Functional Programming Summer School – Fourth Summer School, CEFP 2011, Revisited Selected Lectures, Lecture Notes in Computer Science (LNCS), Vol. 7241, pp. 451-514, Springer-Verlag, ISSN: 0302-9743, 2012.

[8] M. Tóth, I. Bozó, J. Kőszegi, Z. Horváth: *Static Analysis Based Support for Program Comprehension in Erlang,* In Acta Electrotechnica et Informatica, Volume 11, Number 03, October 2011. Publisher: Versita, Warsaw, ISSN 1335-8243 (print), pages 3-10.