# Modeling Dynamic Type Systems in Statically Typed Languages

**András Németh, Melinda Tóth**

Faculty of Informatics, Eötvös Loránd University

{neataai, tothmelinda}@caesar.elte.hu

Each general-purpose programming language has its unique properties which make some of them more favorable compared to others in the respect of solving specific problems. We often write big programs using one general-purpose programming language that is good enough for the most of our intentions, but often might not be the best choice on a few but essential areas. To overcome such complications, we can piece our program together using components written in different programming languages, resulting in a mixed construction that can be an optimal solution for that problem domain.

In this paper, we choose two programming languages: Erlang [2] and C++ [3]. On one hand, Erlang is a good choice to achieve greater productivity and easy cluster communication, or to write soft real-time applications. On the other hand, with C++, one can produce more efficient programs, since the emitted machine code is much more optimized and performs better than interpreted and platform-independent bytecode. Putting these languages together can be a good choice if the computation-intensive parts of a distributed application is written in C++ while the rest in Erlang. However there is one fundamental difference between the two languages: these have completely different type systems [1]; while Erlang is dynamically typed, C++ is statically typed. In order to implement parts of our program in C++, modeling the type system of Erlang is required since the algorithms implemented with both languages have to operate on the same data. In order to ensure seamless transitions from one to the other and vice versa, we should be able to specify the type of data even when we do not know in advance what the data is.

Here, a definition of a dynamic type system is presented that preserves strong typing rules in respect of its semantics along with the programming techniques needed to achieve the same runtime behavior that it has in its original language environment. A reference implementation is also introduced using the language constructs of the statically typed language to demonstrate the feasibility of the results presented by our research.

## References

[1] Benjamin C. Pierce: Types and Programming Languages. MIT Press, 2002. ISBN 0-262-16209-1.

[2] Joe Armstrong: Programming Erlang: Software for a Concurrent World, Pragmatic Bookshelf, 2007. ISBN: 978-1934356005.

[3] Bjarne Stroustrup: The C++ Programming Language. Addison-Wesley, ISBN 978-0321563842, May 2013, 4th edition.

[4] András Németh: Data Access Optimization. On the $31^{th}$ National Scientific Students Associations Conference, Budapest, Hungary, 2013.

[5] András Németh: Processable Erlang Data in C++. Talk at the Middle-European Conference on Applied Theoretical Computer Science. Koper, Slovenia, 2013.