A time, energy and security coordination approach

Benjamin Rouxel
University of Amsterdam
benjamin.rouxel@uva.nl

Julius Roeder
University of Amsterdam
j.roeder@uva.nl

Sebastian Altmeyer
University of Amsterdam
altmeyer@uva.nl

Clemens Grelck
University of Amsterdam
c.grelck@uva.nl

Coordination is a well established computing paradigm with a plethora of languages, abstractions and approaches surveyed in [1]. Yet, we are neither aware of any adoption of the principle in the broader domain of low-powered safety-critical embedded systems, nor are we aware of time, energy and security aware approaches to coordination.

We aim at contributing to the community by bringing a coordination approach to real-time applications with the establishment of a Domain Specific Language (DSL). Our coordination workflow considers time and other non-functional properties, such as energy and security, as first-class citizens in application designs. We therefore aim at building a complete toolchain and workflow to compile a coordinate application to a final executable as presented by Figure 1.

An application organised according to the coordination paradigm consists of a collection of interacting components. Components, also known as actors or tasks, represent application features, sequential building blocks of application, implemented in a general-purpose programming language [2]. Given the focus of the safety-critical embedded systems domain, we exclusively work with the system-level programming language *C*. Hence, a component is technically a callable *C* function with certain restrictions on its functional behaviour, together with a set of non-functional properties, i.e. timing, energy and security.

The coordination language emphasizes communication, concurrency and synchronisation (referred to by the term *coordination*). It aims at describing component interactions in terms of precedences and data exchange. In contradiction with streaming languages such as StreamIT [3], a coordination language is independent from the actual code, but it dictates, to the scheduler, how this code should be executed. An example is the coordination language S-Net [2]. However, like other coordination approaches S-Net merely addresses the functional aspects of coordination programming and has left out any non-functional requirements, not to mention time and security, in particular.

We target CPS platforms executing on COTS multi-core heterogeneous processors where time, security and energy are safety and mission-critical. Our approach is not yet to improve timing analyses or scheduling policies themselves, but to make those secure and efficient by controlling the whole design path from specification to code generation. Hence, we describe how to exploit our coordination language and how to integrate its semantic into state-of-the-art scheduling techniques, e.g. [4].

This work is funded by the European Union Horizon2020 research and innovation programme under grant agreement No. 779882 (TeamPlay).

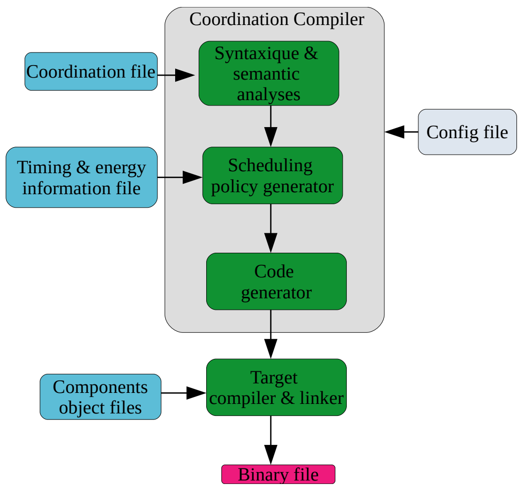

Figure 1: Coordination workflow

For this session, we will describe our DSL design and implementation. We will show how flexible and helpful our DSL is to build secure and safe CPS with real-time, energy and security constraints. We will present how we integrate both time, energy and security as primary properties into our application modelling. Then we will present the different tools and technologies which are part of our toolchain, and motivate our choices among the vast possibilities of existing tools. We will finally demonstrate the viability of our approach on a Drone use-case, given by the University of Southern Denmark [5], while targeting a BIG.little ARM[1] based platform.

References

[1] G. Ciatto, S. Mariani, M. Louvel, A. Omicini, and F. Zambonelli, "Twenty years of coordination technologies: State-of-the-art and perspectives," in *International Conference on Coordination Languages and Models*. Springer, 2018, pp. 51–80.

[2] C. Grelck, S.-B. Scholz, and A. Shafarenko, "Asynchronous stream processing with s-net," *International Journal of Parallel Programming*, vol. 38, no. 1, pp. 38–67, 2010.

[3] W. Thies, M. Karczmarek, and S. Amarasinghe, "Streamit: A language for streaming applications," in *Compiler Construction*. Springer, 2002, pp. 179–196.

[4] B. Rouxel, S. Skalistis, S. Derrien, and I. Puaut, "Hiding communication delays in contention-free execution for spm-based multi-core architectures," in *31th Euromicro Conference on Real-Time Systems (ECRTS19)*, 2019.

[5] EU H2020, "TeamPlay Project," https://teamplay-h2020.eu/.

[1]https://www.hardkernel.com/shop/odroid-xu4/