Model-Driven Allocation Engineering – Abridged Version

Uwe Pohlmann[1] Marcus Hüwe[2]

Keywords: Allocation, Deployment, Constraint Programming, Automotive, Model-Driven Software Engineering, Cyber-physical Systems

Cyber-physical systems provide sophisticated functionality and are controlled by networked electronic control units (ECUs). Nowadays, software engineers use component-based development approaches for developing the software. Moreover, software components have to be allocated to an ECU in order to be executed. Engineers have to cope with topology, software, and timing dependencies and memory, scheduling, and routing constraints [Al13]. Currently, engineers use techniques like integer linear programming or SAT-based encodings to specify allocation constraints and to derive a feasible allocation automatically. However, encoding constraints manually is a complex task [ZP13].

This paper is an abridged version of our paper [PH15] that introduces our model-driven, allocation engineering approach. The original paper contributes an approach for specifying allocation constraints in an easy, expressive, and more compact way and for deriving a feasible allocation automatically. In particular, it helps engineers if constraint satisfaction is a crucial, safety-critical aspect.

Fig. 1 depicts the process of our approach. We provide a new domain-specific *Allocation Specification Language (ASL)* that embeds the Object Constraint Language (OCL). The OCL enables to specify object query expressions in the context of models. We generate a system of linear inequalities automatically using the specified allocation constraints on the basis of a software architecture model and hardware platform model. The generated linear inequalities are independent from concrete solvers. Hence, different existing solving methods, like integer linear programming, SAT solvers, or metaheuristics can be used to compute feasible solutions automatically. Afterward, we generate an allocation model from a computed solution of the system of inequalities that maps the component instances from the software architecture to ECUs from the hardware platform.

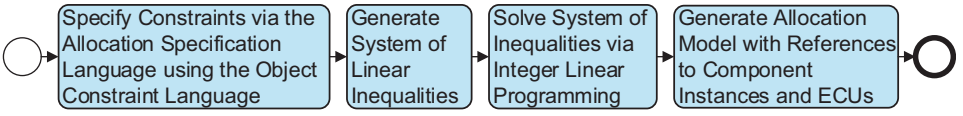

Fig. 1: Model-Driven Allocation Engineering Process

[1] Software Engineering, Fraunhofer IEM, Zukunftsmeile 1, 33102 Paderborn, Germany, uwe.pohlmann@iem.fraunhofer.de
[2] Software Engineering, Fraunhofer IEM, Zukunftsmeile 1, 33102 Paderborn, Germany, marcus.huewe@iem.fraunhofer.de

The ASL provides the four constraint kinds: *collocation*, *separateLocation*, *requiredLocation*, and *requiredResource*. Each constraint kind has its own formal semantics definition and is transformed to a different system of linear inequalities. Fig. 2 shows an example of a constraint specification that uses all supported allocation constraint kinds.

```
constraint collocation collocateSc11AndSc12
 descriptors (first:instance::ComponentInsta
 second:instance::ComponentInstance);
 ocl Tuple{first=self.getSWInstance('sc11'),
  second=self.getSWInstance('sc12')}
  .oclAsSet();
 }
constraint separateLocation
 redundantComponents {
 descriptors (first:instance::ComponentInsta
  second:instance::ComponentInstance);
 ocl Tuple{first=self.getSWInstance('sc7a'),
  second=self.getSWInstance('sc7b')
  }.oclAsSet();
```

```
constraint requiredLocation
 sc1ToECUsWithSensorAccess {
 descriptors (first:instance::ComponentIn
  second:hwresourceinstance::ResourceInst
 ocl self.allocateToECU('sc1','hw4')->
  union(self.allocateToECU('sc1','hw5'));
}
constraint requiredResource maxMemory {
 weight requiredMemory;
 bound maxMemory;
 descriptors (componentInstance:instance:
  ComponentInstance, resourceInstance:hwr
 ocl self.maxMemoryConsumption();
}
```

Fig. 2: Examples of Supported Constraint Kinds of the Allocation Specification Language

We evaluate our approach with an automotive case study modeled with MechatronicUML [Be14] based on the Brake-by-wire case study by Aleti [Al13]. Our validation shows that we can specify allocation constraints with less engineering effort and are able to derive feasible allocations automatically for a complex component-based software model and hardware platform model. As a result of using our approach, the solving of the whole allocation problem becomes transparent for engineers. They specify the allocation constraints via the ASL and get a feasible solution automatically without having to know how to encode and solve the allocation problem as a system of inequalities like an integer linear program.

Acknowledgments: This work was developed in the Leading-Edge Cluster 'Intelligent Technical Systems OstWestfalenLippe' (IT'S OWL). The IT'S OWL project is funded by the German Federal Ministry of Education and Research.

References

[Al13] Aleti, A.; Buhnova, B.; Grunske, L.; Koziolek, A.; Meedeniya, I.: Software Architecture Optimization Methods: A Systematic Literature Review. IEEE Transactions on Software Engineering, 39(5):658–683, May 2013.

[Be14] Becker, Steffen; Dziwok, Stefan; Gerking, Christopher; Heinzemann, Christian; Schäfer, Wilhelm; Meyer, Matthias; Pohlmann, Uwe: The MechatronicUML method: model-driven software engineering of self-adaptive mechatronic systems. In: Companion Proceedings of the 36th International Conference on Software Engineering. ICSE Companion '14, ACM, New York, USA, pp. 614–615, 2014.

[PH15] Pohlmann, Uwe; Hüwe, Marcus: Model-Driven Allocation Engineering. In: Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering. ASE '15. IEEE, pp. 374–384, Nov 2015.

[ZP13] Zeller, Marc; Prehofer, Christian: Modeling and efficient solving of extra-functional properties for adaptation in networked embedded real-time systems. Journal of Systems Architecture, 59(10, Part C):1067–1082, 2013. Embedded Systems Software Architecture.