(Hrsg.): SE'18: Software Engineering-Tagung der Gesellschaft für Informatik (GI),
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2018 1

Improving the Efficiency of Dislocality Constraints for an Automated Software Mapping in Safety-Critical Systems

Robert Hilbrich[1], Michael Behrisch[2]

Abstract: This is a brief overview of the paper, which should be 70 to 150 words long and include the most relevant points. This has to be a single paragraph.

Keywords: Keyword1; Keyword2

1 Introduction

Engineering complex and safety-critical systems, such as flight control systems aboard an airplane, is still challenging and costly. Despite recent advancements in our model-based tool suites and engineering methods, the design of these systems still bears risk and uncertainties with regard to its outcome. The formalization and automation of crucial engineering tasks appears to be a promising approach to tackle these challenges [Ch07]. Systems in these areas are engineered to implement a complex interplay between mechanical elements, electronic components, as well as (embedded) software. Therefore, their design has to mirror this interplay and requires the development of a hardware and software architecture.

In practice, these architectures can often be developed independent of each other, but their integration in the final system requires a *link* between the software components and their hardware resources. This *link* is referred to as a *deployment* of the software components. Constructing a deployment requires the systems engineer to *map* software components to resources and to *schedule* the access to shared resources. Therefore, mapping refers to a *spatial allocation*, while scheduling refers to a *temporal allocation* of software components.

The construction of a deployment is an engineering task, which not only affects the fulfillment of functional requirements by providing the necessary, but also affects the satisfaction of non-functional requirements, such as safety and reliability. Redundancy and fault tolerance can only be achieved, if critical software components are deployed accordingly.

The construction of a deployment is a very elaborate task with zero tolerance for errors as they may jeopardize the correctness of the system. At the same time, it requires a detailed

[1] German Aerospace Center (DLR), Rutherfordstr. 2, 12489 Berlin, Germany robert.hilbrich@dlr.de
[2] German Aerospace Center (DLR), Rutherfordstr. 2, 12489 Berlin, Germany michael.behrisch@dlr.de

understanding of the requirements of all software components and the capabilities of all hardware resources in the system. Due to the sensitivity and complexity of this task, its formalization and automation is a valuable research goal.

2 Automated Construction of Deployments

In order to achieve an automated construction of a deployment and to argue its correctness, a formalization of the mapping problem is required. For smaller mapping problems, this has been successfully achieved based on Linear Integer Programming [Da06; Ku09], SMT-based solvers [VS13] or evolutionary algorithms [Wh11]. However, these approaches reach their limits when larger, real-world mapping problems with limited gradient information to guide a search process are considered.

The authors instead chose to transform a mapping problem into a semantically equivalent *Constraint Satisfaction Problem (CSP)* [Ap03; De03] and solve this CSP with *Constraint Programming* techniques [PFL16; RBW06]. The advantages of using Constraint Programming in comparison to other techniques lie in the availability of powerful modeling elements, such as an ALLDIFFERENT constraint, and the ease with which custom search heuristics can be implemented.

2.1 Constraint Satisfaction Problems

Constraint Programming refers to a set of techniques in artificial intelligence and operations research. These techniques assist in finding solutions for problems based on variables, which are affected by constraints. Each constraint defines valid or invalid solutions for a subset of these variables. In this paper, a subclass of constraint satisfaction problems is used to express mapping problems: *finite domain integer constraint satisfaction problems* in which each variable has a finite integer domain. Solutions for this problem class can be obtained by applying a combination of *search* techniques – including backtracking – and constraint *propagation* techniques for value elimination.

To illustrate the modeling approach of Constraint Satisfaction Problems, consider the well-known *Map Coloring* problem as an example. This problem asks, whether it is possible to color a map with only four colors in such a way, that neighboring countries have different colors. It can be formulated as a CSP by assigning an integer variable x_i for each country with the index i. The domain of each variable corresponds to the four colors: $D_{x_i} = \{0, 1, 2, 3\}$. In order to model the restrictions of this problem, a constraint is added for each pair of adjacent countries. If country x_i is adjacent to country x_j, then $x_i \neq x_j$ is required. The search algorithm is now responsible to select a variable and test a value of its domain. Assuming a simple "first variable, first value" strategy, the variable x_0 would be chosen and set to the value 0 as a test. This would be *propagated* to all variables which are directly

linked to x_0 by a constraint, so that the value 0 gets removed from their domains. This removal may lead to other value removals in indirectly linked variables and is processed until a fix point is reached. If a contradiction is encountered or the domain of a variable becomes empty, *backtracking* is initiated, so that the next value of the variable x_0 is tested. Otherwise, the search algorithm continues with the next uninstantiated variable.

This example also shows, that the propagation of the NotEqual constraint is *weak*, because it affects only two variables and invalidates only 4 out of the 16 possible value combinations between two variables.

2.2 Toolsuite ASSIST

As a proof of concept for the ongoing research toward an automated construction of deployments based on Constraint Satisfaction Problems, the toolsuite *Architecture Synthesis for Safety-Critical Systems (ASSIST)* [Hi14] was developed by the authors. It is publicly available under the Eclipse Public License 2.0 and uses the constraint solver *Choco 4* [PFL16] internally.

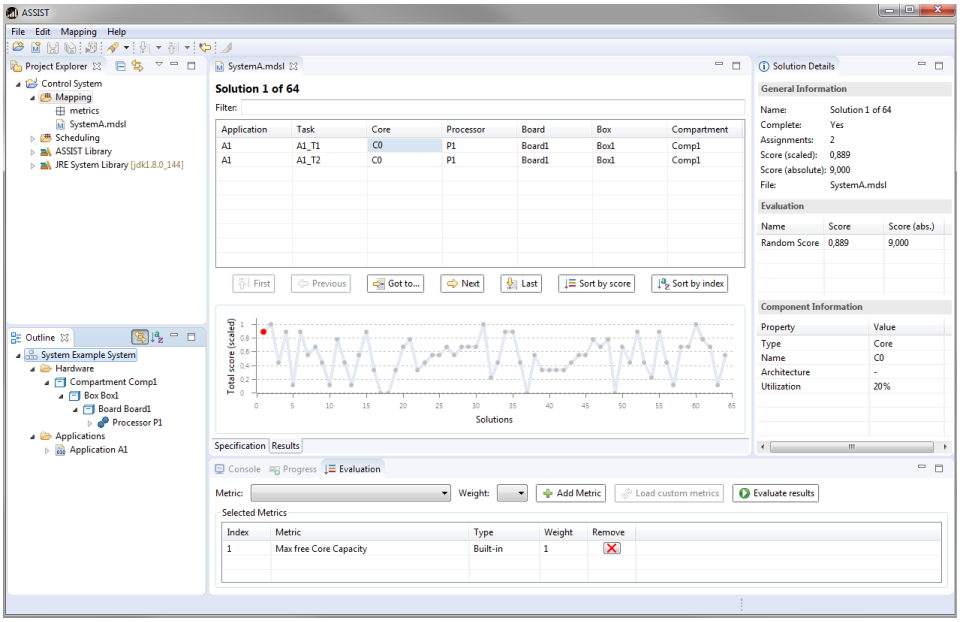


Fig. 1: Screenshot of ASSIST with a specification for a control system

ASSIST (see Figure 1) allows a systems engineer to automatically construct and optimize mappings and schedules based on textual specifications of the

- software components and hardware resources,

- *dislocality*, *dissimilarity* and *colocality* requirements,
- optimization goals.

The textual specifications in ASSIST conform to a domain-specific language which allows to hide the intricacies of a formal specification. Using a domain-specific language is expedient to enable systems engineers without a formal education in computer science to precisely specify a deployment problem.

3 Ensuring Fault Tolerance by Requiring Dislocality

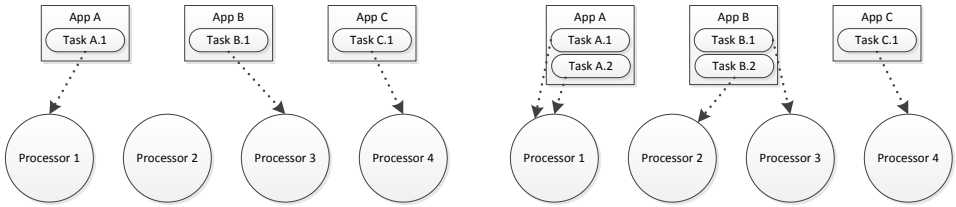
In order to achieve fault tolerance and reliability, it is essential to support "significant differences" in the choice of resources to which critical software components are deployed to. For instance, a simple redundancy requirement between two software components, may force the systems engineer to allocate these software components to different processing boards in different locations aboard an airplane. Furthermore, systematic errors and undetected design flaws in hardware components may be addressed by choosing dissimilar hardware resources, for example processors or memory blocks from different vendors.

Due to the importance of choosing "different" resources for fault tolerance and reliability in safety-critical systems, engineering tools for an automated construction of deployments need to be able to fully support these choices. ASSIST supports the engineer by offering *dislocality* and *dissimilarity* requirements as part of the domain specific language. They can be used to enforce "differences" for the choice of resources for the mapping of software components. Finding an *efficient* formulation to express the semantics of each requirement as a Constraint Satisfaction Problem is challenging, but also essential in order to provide an effective toolsuite for the engineering of safety-critical systems.

In order to illustrate the challenges and to present specific modeling improvements, the *dislocality* requirement is used as an central example in this paper. Please note, that the concepts developed for *dislocality* requirements can also be reused for *dissimilarity* requirements.

The semantics of a dislocality requirement can be illustrated with the following deployment problem (see Figure 2). In the example system, there are three *applications* consisting of one or more *tasks*. Each task has to be mapped to exactly one of the *processors* in the system. It is assumed, that the processor contains multiple cores, so that multiple tasks can be mapped to a single processor. However, in order to ensure fault tolerance, a *dislocality* requirement is added for all applications. This means, that the applications must not share a processor, so that a faulty processor affects only one application.

Expressing the basic deployment problem with constraints is straight forward. Each task i in the system is represented by an integer variable $X_i \in \{0, 1, \ldots, n-1\}$. The domain of each



(a) Simple Case (b) Complex Case

Fig. 2: An Example Deployment Problem - Mapping Tasks to Processors

variable X_i corresponds to the indices of the n processors in the system. This formulation is still missing the constraints for the dislocality requirements.

In the simple case, if all applications consist of only one task, this can be achieved with a single `allDifferent` constraint (see [BCR14]) over all variables X_i. This situation is depicted in Figure 2 (a). Fortunately, Choco 4 already contains an implementation for an `allDifferent` constraint based on the algorithm of Régin [Ré94].

However, in reality, applications usually consist of more than just one task and tasks of the same application *can* typically share the same processor. This situation is more complex and depicted in Figure 2 (b). Unfortunately, in this case, the previous approach of applying a single `allDifferent` constraint over all variables X_i can no longer be used. It would prevent solutions in which a processor is shared by multiple tasks of the same application. The existing `allDifferent` constraint only ensures, that values for a list of variables are different. However, the complex case requires an advanced `allDifferent` constraint operating with a list of a list of variables and ensuring that the combined values for each list of variables are different. Unfortunately, such a constraint is neither part of the Global Constraint Catalog [BCR14] nor available in Choco 4.

4 Modeling Complex Dislocality Requirements

- Based on available tools - one option is the recursive approach
- Implementation is straight forward, but it requires a lot of constraints - slow in practice
- Another option is to react on instantiation only and remove unwanted solutions - weak in propagation
- Last Option is to combine inst-only approach with int-values union and alldifferent

5 Experiments

- Synthetic example generator

- 10/20 Examples were generated with a parameter domain ...
- Search with the same strategies DomWD, minValueFirst
- iMac 5k, 64 GB RAM, Choco 4.0.6

6 Results

- Show results for var count, constraint count
- Show results for resolution time
- Show results for backtracks and fails

7 Conclusions

I am a summary - what shall i put here?

References

[Ap03] Apt, K. R.: Principles of constraint programming. Cambridge University Press, 2003.

[BCR14] Beldiceanu, N.; Carlsson, M.; Rampo, J.-X.: Global Constraint Catalog, Technical Report T2012:03, ISSN: 1100-3154, SICS, Feb. 2014, URL: http://www.emn.fr/z-info/sdemasse/gccat/.

[Ch07] Chapman, R.: Correctness by construction: putting engineering (back) into software. In: Proceedings of the 2007 ACM international conference on SIGAda annual international conference. SIGAda '07, ACM, Fairfax, Virginia, USA, pp. 100–100, 2007, ISBN: 978-1-59593-876-3, URL: http://doi.acm.org/10.1145/1315580.1315605.

[Da06] Damm, W.; Metzner, A.; Eisenbrand, F.; Shmonin, G.; Wilhelm, R.; Winkel, S.: Mapping Task-Graphs on Distributed ECU Networks: Efficient Algorithms for Feasibility and Optimality. In: Embedded and Real-Time Computing Systems and Applications, 2006. Proceedings. 12th IEEE International Conference on. Pp. 87–90, 2006.

[De03] Dechter, R.: Constraint Processing. Elsevier Science & Technology, 2003.

[Hi14] Hilbrich, R.: Architecture Synthesis for Safety Critical Systems - ASSIST, online, 2014, URL: http://github.com/roberthilbrich/assist-public.

[Ku09] Kugele, S.; Haberl, W.; Tautschnig, M.; Wechs, M.: Optimizing Automatic Deployment Using Non-functional Requirement Annotations. In (Margaria, T.; Steffen, B., eds.): Leveraging Applications of Formal Methods, Verification and Validation. Vol. 17, Communications in Computer and Information Science, Springer Berlin Heidelberg, pp. 400–414, 2009, ISBN: 978-3-540-88478-1, URL: http://dx.doi.org/10.1007/978-3-540-88479-8_28.

[PFL16] Prud'homme, C.; Fages, J.-G.; Lorca, X.: Choco Documentation, TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2016, URL: http://www.choco-solver.org.

[RBW06] Rossi, F.; van Beek, P.; Walsh, T., eds.: Handbook of Constraint Programming. ELSEVIER SCIENCE & TECHNOLOGY, 2006.

[Ré94] Régin, J.-C.: A Filtering Algorithm for Constraints of Difference in CSPs. In: Proceedings of the Twelfth National Conference on Artificial Intelligence (Vol. 1). AAAI '94, American Association for Artificial Intelligence, Seattle, Washington, USA, pp. 362–367, 1994, ISBN: 0-262-61102-3, URL: http://dl.acm.org/citation.cfm?id=199288.178024.

[VS13] Voss, S.; Schatz, B.: Deployment and Scheduling Synthesis for Mixed-Critical Shared-Memory Applications. In: Engineering of Computer Based Systems (ECBS), 2013 20th IEEE International Conference and Workshops on the. Pp. 100–109, 2013.

[Wh11] White, J.; Dougherty, B.; Thompson, C.; Schmidt, D. C.: ScatterD: Spatial deployment optimization with hybrid heuristic/evolutionary algorithms. TAAS 6/3, p. 18, 2011.