Towards Facilities for Modeling and Synthesis of Architectures for Resource Allocation Problem in Systems Engineering

Stephen Creff, Jérome Le Noir, Eric Lenormand, Sébastien Madelénat

Towards Facilities for Modeling and Synthesis of Architectures for Resource Allocation Problem in Systems Engineering

Stephen Creff
IRT SystemX
Palaiseau, France
stephen.creff@irt-systemx.fr

Jérôme Le Noir, Eric Lenormand, Sébastien Madelénat
Thales Research & Technology
Palaiseau, France
firstname.lastname@thalesgroup.com

ABSTRACT

Exploring architectural design space is often beyond human capacity and makes architectural design a difficult task. Model-based systems engineering must include assistance to the system designer in identifying candidate architectures to subsequently analyze trade-offs. Unfortunately, existing languages and approaches do not incorporate this concern, generally favoring solution analysis over exploring a set of candidate architectures.

In this paper, we explore the advantages of designing and configuring the variability problem to solve one of the problems of exploring (synthesizing) candidate architectures in systems engineering: the resource allocation problem. More specifically, this work reports on the use of the Clafer modeling language and its gateway to the CSP Choco Solver, on an industrial case study of heterogeneous hardware resource allocation (GPP-GPGPU-FPGA).

Based on experiments on the modeling in Clafer, and the impact of its translation into the constraint programming paradigm (performance studies), discussions highlight some issues concerning facilities for modeling and synthesis of architectures and recommendations are proposed towards the use of this variability approach.

KEYWORDS

Architecture Synthesis, Variability Modeling, Constraint Solving, Empirical Study, Allocation Problem

ACM Reference Format:
Stephen Creff and Jérôme Le Noir, Eric Lenormand, Sébastien Madelénat. 2020. Towards Facilities for Modeling and Synthesis of Architectures for Resource Allocation Problem in Systems Engineering. In *24th ACM International Systems and Software Product Line Conference (SPLC '20), October 19–23, 2020, MONTREAL, QC, Canada.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3382025.3414963

1 INTRODUCTION

The *conceptualization*, a core process according to the ISO 42020 [15], aims at characterizing the problem space and determine (by synthesis) suitable solutions in the solution space that address stakeholder concerns, achieve architecture objectives and meet relevant requirements. Due to the increasing system complexity, architects have to choose among a combinatorially growing number of design options: exploring architectural design spaces, and bringing the best alternatives out of the solution space has often become beyond human capacity [12]. The need for automated design space exploration that improves an existing architecture specification has been recognized [2, 5, 16, 24, 26], but searching through a large number of possibilities is often time- and cost-consuming [9], and error-prone [6].

Model-Based Systems Engineering (MBSE) must include assistance to the system designers in identifying candidate architectures to subsequently analyze trade-offs. Unfortunately, efforts in the systems engineering community generally favors solution analysis, e.g. in providing system modeling languages and tools (e.g. SysML, Capella), over exploring a set of candidate architectures (architectures synthesis), a feature needed in the early stages.

Kang *et al.* [17] pointed out the wish of some effective frameworks, and [28], emphasizes the one to have declarative solving tools to tackle design problems like allocation, configuration, sizing, and architectural design. The need goes towards the modeling and solving of sub-defined systems, offering variants in the design and free-ranged parameters, something close to the product-line community concerns. Therefore, in this paper, we explore the advantages of designing and configuring variability problems to solve one of the problems of exploring (synthesizing) candidate architectures in systems engineering: the resource allocation one.

This article follows a report on the use of the Clafer modeling language [5] and its gateway to the Choco Solver[8] Constraint Solver Programming (CSP), on an implemented industrial case study of heterogeneous hardware resources allocation which involves General Purpose Processors (GPP), General-Purpose Computing on Graphics Processing Units (GPGPU, or GPU for short), and Field Programmable Gate Arrays (FPGA). The size of the case study highlights the problem of scaling up (over 4 billions possibilities). The article puts forwards some observations from modeling activities and performance studies on the impact of the ways problems can be modeled in Clafer and their translation into the constraint programming paradigm for an efficient resolution. With the objective to outline some characteristics for architecture exploration modeling and for an associated resolution tooling, the discussions highlight some issues concerning facilities for modeling and synthesis of architectures, and recommendations are proposed towards the use of the Clafer approach.

The remainder of the paper is organized as follows. Section 2 highlights the motivations towards the needs concerning facilities

SPLC '20, October 19–23, 2020, MONTREAL, QC, Canada

ACM ISBN 978-1-4503-7569-6/20/10…$15.00
https://doi.org/10.1145/3382025.3414963

for modeling and synthesis of architectures in Systems Engineering, the motivations regarding this empirical study and positions related work in the domain. Then, in Section 3, the allocation resource case study is introduced. Section 4 describes the experiments conducted to encode the model into Clafer, and summarizes the observations on the impact of alternative modeling choices on the resolution. Later, Section 5 highlights generic issues and provides recommendations towards the use of the approach. Finally, the last Section draws our conclusion and narrows down possible future works.

2 MOTIVATIONS

Motivations highlight needs for synthesis in MBSE, related works, and introduce our approach.

2.1 MBSE needs for architecture synthesis

Classic MBSE activities, like function analysis and structural design, are not adapted to early design phases where many parameters are still free/undefined, and it is expensive to build analysis models to explore the space of possible solutions. Indeed, as summed-up in [21], some design parameters may not be fixed, some components may not be selected, some resources may not be allocated, and the structure of the architecture may not be fixed, resulting in many try-and-test iterations. Some facilities for modeling and synthesis of architectures in systems engineering is required.
More generally, an effective framework is depicted in [16] as composed of a suitable representation, an efficient analysis tooling, and an adequate exploration method. Besides, we highlight two essential needs in terms of modeling language and associated solving capabilities:

- Providing to the architect a language capable of representing a complex system, with degrees of freedom (as called in [2]) like: free parameters, sub-defined allocations, free selection of alternatives;
- Having resolution facilities to explore automatically the design space to find eligible architectures, i.e. those compatible with the various system requirements. Considering parameter values, types to consider can be integer (discrete problem), real (continuous problem), or both (mixed problem).

2.2 Related work

The research activities are scattered across many research communities, and a plethora of architecture optimization approaches based on formal architecture specifications have been developed.

First, and obvious, a body-of-knowledge exists on dedicated software optimization, which consider heterogeneous computation nodes that use combination of CPUs, GPUs and FPGAs in allocation problems. For example, [7, 10], where an optimization model is defined and a mixed-integer programming (MIP) solver and a framework for constraint integer programming. A similar problem is addressed by Svogor et al. by using a genetic algorithm based method [29]. This work discusses a possible approach to compute allocation schemes for hardware platforms with CPUs, GPUs and FPGAs nodes. Although it deals with allocation, our concerns do not go towards a specific resolution (direct CSP encoding), but towards the search for a higher level design language, allowing an easier formalization a non-optimal but satisfying resolution.

Then, in considering more generic approaches and problems, some related work concerns approaches to extend MBSE practices. In [1], Min *et al.* propose a multi-objective optimization from SysML model, with a process integration and design optimization (PIDO) framework (using the ModelCenter tool). An analytic and non-synthetic approach. Besides, the approach from [21] extends SysML and the OOSEM methodology [11] to allow modeling a large number of variants by using "decision points" (e.g. in allocation). Variability is not native. The generation is based on Choco [8] or PyOpt[1], for discrete or continuous problems, missing performance details.

In product-line engineering, variability models have been extensively researched and are now widely used to represent configuration spaces. Variability is commonly addressed by modeling features and their inter-dependencies as a feature model (FM) [17]. Considering open range properties, an extension of FMs exists where features have quality attributes, e.g. [23, 27]. Considering allocation problems, in [14], the designer creates the various alternatives of allocations with the MARTE profile and annotations and propose a transformation from a FM to a mathematical representation of an optimization problem. Besides, in [19, 20], the authors propose an approach to deal with the variability (with FM) at the application level (data-flow), resource (hardware) and mapping. They also show that variability-aware methods are convenient to analyse the behaviour of alternative designs because they can exploit the commonalities between the design variants to reduce the computation involved. Making explicit the allocation mapping into FMs could result in huge FMs, not our choice in the following.

No standard is defined to model variability. Among the variability language initiatives is Clafer [5], that combines variability and metamodeling facilities, for architecture description: an interesting characteristic. A set of works binding the architectures exploration need and this variability language and tool-chain was proposed [18, 22, 25, 26]. The authors define modeling methodologies for architectural modeling, as well as Clafer modeling patterns usage. They provide an approach to assess non-functional properties and some optimizations.

2.3 Motivations of the empirical study

In order to model multiple candidates, variability (degrees of freedom [2]) has to be expressed in the model. It can be expressed as features, free instances generation, and free property ranges.

Based on the works in [18, 22, 25, 26], the basic idea behind our study is to experiment the approach on an industrial case study of heterogeneous hardware resource allocation. Some modeling alternatives in Clafer are described to understand the capabilities and analyze the impact the translation into the constraint programming paradigm. We consider the tool-set as it is, without making any changes, in the 0.4.3 version, which is not the latest one but seems more stable (the 0.4.4 seems to offers less performance resolution, the 0.4.5 less features). Besides, contrary to [22, 26] that restrict their reasoning performance perspectives to the number of Clafer elements compiled (i.e. All clafers, Abstract, Concrete, Reference, and Constraints) and execution time, we went a step further, looking at the *resolution variables* at the CSP level, and *resolution branching priorities*.

[1] http://www.pyopt.org/

3 ALLOCATION PROBLEM & CASE-STUDY

In many domains of embedded systems, task allocation is typically addressed as a sub-problem of scheduling real-time multi-core systems. The idea is to use models of the system under development to obtain performance predictions with sufficient accuracy, already prior to the implementation, and thus get an indication on whether a particular allocation is good or bad in terms of performance. Before analyzing architecture candidates performances for trade-offs (e.g. end-to-end response times, deadline misses), these architecture candidates must be synthesized. In our case, the modeling consists of i) a data processing application (i.e., the set of necessary functions represented as a data-flow oriented graph), and ii) a resource-constrained hardware platform (i.e., set of available heterogeneous hardware components like non-programmable processors, generic or specialized processors and communication connections).

3.1 Data-flow oriented software modeling

The architecture is characterized by an oriented graph $(\mathcal{F}, f\_cnx)$ for which the set of nodes $\mathcal{F} = \{f_1, \ldots, f_m\}$ designates the functions (*ProcFunction*) and the set of edges $f\_cnx \subseteq \mathcal{F} \times \mathcal{F}$, representing the communications between the functions (*AFConnector*). An edge $f_cnx(f_i, f_j)$ models the sending of a message from f_i to f_j.

The functions to be deployed on hardware computing units may differ in terms of resource usage. This type of heterogeneity presents a challenge for designers when deciding where to place software components on the compute units. *Functions* and communication *AFConnectors* are annotated with properties. First, considering functions, from experimental results execution time (*execTime* performance property) on specific processor are taken (expressed in *ms*); $costF : \mathcal{F} \rightarrow \mathbb{R}_{>0}$.

Then, the specification of interactions between functions are described by the *tokenSize* property which is the length of data transfer between the two connected processing functions (expressed in Kb); $costC : f\_cnx \rightarrow \mathbb{R}_{>0}$.

In the model, functions are specialized (in an object oriented modeling meaning) into *FPGA*, *GPP*, and *GPGPU* ones, and as already mentioned, performances pre-calculated (estimations).

As explained later in Section 3.4, the application has a known fixed period (equal to the input frame period), which implies, among other things, that no processor or communication connection should take longer than this period to do its job. Each processor in the architecture may be in charge of one or more functions (nodes), when alone on its processor, is approximated by a known duration (which is necessarily less than the input frame period).

3.2 Hardware modeling

The heterogeneous platform is modeled defining boards, connected to each other via Ethernet. Briefly, a Hardware Architecture is seen as $\langle B, P, capaP, cUseP, Comcon, capaComcon, FPGA, fpga, GPP, gpp, pen, GPU, gpu, PCI, pci, E, cUseE, con, HwC\rangle$, such as: B is a finite set of boards; P is a finite set of abstract processors; $capaP : P \rightarrow \mathbb{R}_{>0}$, the processors capacity; *Comcon* is a finite set of abstract communication connections; $capaComcon : Comcon \rightarrow \mathbb{R}_{>0}$, the communication connections capacity; $FPGA \subseteq P$ is a finite set of FPGA; $fpga : B \rightarrow \{FPGA|\emptyset\}$ indicates the optional containment relation from a board; $GPP \subseteq P$ is a finite set of GPP; $gpp : B \rightarrow GPP$ indicates the containment relation from a board; $pen : GPP \rightarrow \mathbb{R}_{>0}$, represent the communication transfer penalties; $GPU \subseteq P$ is a finite set of GPGPU; $gpu : B \rightarrow GPU$ indicates the containment relation from a board; $PCI \subseteq Comcon$ is a finite set of Peripheral Component Interconnect (PCI) connections; $pci : B \rightarrow PCI$ indicates the containment relation from a board; $E \subseteq Comcon$ is a finite set of Ethernet wire connections; $con \subseteq B \times E$ is the set of vertices relating boards and Ethernet connections; $HwC \in CLC$ is set of domain Hardware Constraints. Note that, our modeling only reifies Ethernet communication equipment allocations, PCI allocations are integrated through capacity penalties in *pen* for the GPP on the given Board. For a given allocation, a processor (resp. Ethernet) current usage ($cUseP : P \rightarrow \mathbb{R}_{>0}$, resp. $cUseE : E \rightarrow \mathbb{R}_{>0}$), is seen as the sum of the nodes (resp. vertices) of the data-flow graph that it is in charge of (*totalExecTimebyP*, resp. *currentBWBytesPerMs*). In the GPP case, different penalties (*pen*) are added to this sum depending of the incoming and outgoing data exchanges between nodes host by heterogeneous processors as described in Table 1.

3.3 Deployment/Allocation schemes

Determining feasible allocation schemes requires knowledge on the functional and non-functional properties of the system, the hardware architecture and the system requirements. Globally, from a mathematical viewpoint, an allocation represents a permutation with repetition which assigns one computational unit to each function or data flow communications[2]. The deployment/allocation are expressed as follows:

- $depl \subseteq F \times P$ is the set of vertices of functions deployment onto processors;
- $deplC \subseteq f\_cnx \times E$ is the set of vertices of communications *AFConnector* deployment onto Ethernet connections;

Deployment/allocation is subject to constraints and costs.

3.3.1 Constraints. The defined allocation constraints are:

- *C1*: Residence constraints: A dedicated function (FPGA, GPGPU or GPP) must be deployed onto its dedicated processor.
- *C2*: Unique deployment: Any function (resp. data flow communications) has to (resp. may) be deployed to one and only one computation resource (resp. connection). Note that for communication connection, only Ethernet allocation is reified, the deployment is therefore optional; other paths, i.e. internal memory or PCI bus are not, but considered in the penalties constraints. One or more functions (resp. data flow communications) can be allocated on the same computational unit, depending on the capacity of the unit.
- *C3*: Exactly one variant of each task must be allocated. All functions must to be deployed; indeed, the modeling of the functional architecture corresponds to the linking of all the functions necessary for the system with all the resources made available. Each of the modeled functions has an important role for the global system, so it is necessary to ensure that no function is orphaned during the allocation process.
- *C4*: Compatible connections: If a data flow communication is deployed on a communication connection then its origin

[2]implying that the search space increases exponentially with the number of data-flow or hardware elements

function should be deployed to the first resource in this communication connection and the destination function should deployed to the last resource in this connection.
- *C5*: The capacity of any processor must not be exceeded: An allocation is feasible if the resources consumed by the computation functions deployed to any computational unit do not exceed the resource capacities that the computational unit provides (given a defined usage ratio), cf. Table 3.
- *C6*: The bandwidth of any communication connection must not be exceeded: The sum of the token size message of all data flow connectors (*AFConnector*) deployed on a network bus (*Ethernet*) must not exceed the bandwidth capacity that the communication connection provides (given a defined usage ratio), more details in Table 3.
- *C7*: The bandwidth of any PCI bus must not be exceeded: The sum of the token size messages of all local data flow communications (*AFConnector*) must not exceed the bandwidth capacity that the PCI bus provides (given a defined usage ratio), more details in Table 3.

3.3.2 Costs. Different penalties (*pen*) are modeled to represent the *Costs* of the data exchanges between nodes host by heterogeneous processors. The different *Costs* are:
- *P1*: On the same board, the use of GPGPU and GPP implies a cost related to the data transfer via the PCI bus;
- *P2*, and *P4*: An incoming communication with its source from another board implies a penalty related to the size of the token of the exchange message;
- *P3*: A communication from a GPGPU or GPP to other boards via Ethernet implies a penalty related to the token size of the exchange message;
- *P5* and P6: A communication between two FPGA on distinct boards doesn't use Ethernet, and thus implies only a penalty of 1 ms on the GPP hosted by the FGPA's board;
- *P7*: A communication between between an FPGA and a GPU or GPP on distinct boards (via Ethernet) implies a penalty of 2.5 ms on the GPP hosted by the FGPA's board and a penalty related to the token size of the exchange message;
- *P8*: On the same board, a communication between an FPGA and a GPGPU or GPP implies a communication cost related to the data transfer via the PCI bus and a penalty of 1 ms on the GPP hosted by this board;
- *P9*: On a board hosting an FPGA, ten percent of the sum of nodes processing time in charge of this FPGA are added to the GPP hosted by the same board;
- *P10*: On a board hosting an FPGA, if the FPGA is allocated to at least a node, the GPP maxload is divided by two.

The different *Costs* taken into account on our resource allocation problem are summarized in Table 1, and illustrated in Figure 1.

Table 1: Penalties

ID	src	tgt	link	TK	GPP	pci
P1	GPU	GPP	local	pen-gpu	-	+
P2	Any	GPU|GPP	in	ethernet	-	-
P3	GPU|GPP	Any	out	ethernet	-	-
P4	Any	FPGA	in	ethernet	-	-
P5	FPGA	FPGA	in	-	1	-
P6	FPGA	FPGA	out	-	1	-
P7	FPGA	GPU|GPP	out	-	2.5	-
P8	FPGA	GPU|GPP	local	-	1	+
P9	FPGA	-	-	-	+10%	-
P10	FPGA	-	-	-	GPP/2	-

3.4 Application description

The application taken as an example is representative of a class of data-flow signal or image processing applications hosted in real-time sensor sub-systems, generally subject to strong constraints on SWaP (Size, Weight and Power) and/or processing latency. In the present case, it implements an image processing chain that takes periodic 2D frames as input and performs a sequence of transformations to detect and process objects of interest. Operations range from systematic high data rate operations such as image corrections in the early stages to progressively less regular operations at lower data rate in the later stages. The former are good candidates for massively parallel architectures like GPGPU and/or dedicated parallel IPs implemented in an FPGA, while the latter, being less regular and partially data-dependent, can only be hosted in a general purpose processor.

Four boards, each containing a GPP, GPGPU and optionally one FPGA as shown in Table 2, are provided to host the application. Processing sites communicate via a PCI bus on-board and via Ethernet connections between boards. Each node of the application graph is pre-allocated to a particular type of processor, but choosing the appropriate board hosting the processor is the objective of the mapping phase. In our experimentation, two nodes (D2 and E2) are kept open to be allocated to a GPP or a GPGPU.

Each node consumes at each firing some processing time on its hosting processor, which is approximated as a fixed value. Similarly, transferring tokens between processing sites consume time on the communication connections on the way, each of which offering a fixed bandwidth budget to share between all its users: the expected time for a communication is then the size of the token divided by this fixed bandwidth. In the specific case of the GPP, which in fact contains 4 individual cores, the processing time of a node is its estimated computation time on one core.

As mentioned above, the scheduling of actors in the application graph follows a data-flow token-based policy (Kahn Process), a given actor producing after each firing a fixed-sized token to each of its sink actors. Tokens are propagated via the relevant communication link(s) to the processing site that holds their destination application node, or are just considered available when both the sender and the receiver are on the same processor. The application is triggered at every period P by the initial token which is the input frame, and all actors, and consequently all necessary tokens communications, fire exactly one time per period P. Mapping estimations are done considering an execution model that determines the conditions and order in which functions are fired. The execution model also assumes that the graph is repeated with a given known period for inputs, which leads to a fixed scheduling scheme for functions, supposed to repeat identically at each period. In the current example, the graph is acyclic and the execution model follows a blocking read policy, where a function becomes eligible when all

Table 2: Boards

Boards	GPP	GPU	FPGA
Board0	X	X	-
Board1	X	X	-
Board2	X	X	X
Board3	X	X	X

Table 3: Processors, Ethernet and PCI capacities

Processors	Maxload usage	Capacity	Max usage
GPP	20.44	4*7	73%
GPGPU	5.6	7	80%
FPGA	6.65	7	95%
WireConnection			
Ethernet	848	893	95%
PCI	5428	5715	95%

its inputs for the period are available in its memory. In the case of an acyclic graph, the same approach can be used, as long as one can find a periodic static scheduling policy.

All processors and communication links, being respectively in charge of potentially several computing actors or tokens that derive from the mapping, must be fast enough to perform their job in less than their time budget. As shown in Table 3, which considers a period P=7, each type of resource has only a percentage of the period as budget, which leaves some useful margin before the actual implementation (GPPs have their budget equal to the percentage of 4*P , i.e. 18.2 ms). Similarly, communication links have only a percentage of their nominal bandwidth available.

The application graph is shown in Figure 1. Nodes of the graph are annotated with their computing cost (in *ms*), and edges between nodes have the size of their token (in *Kb*).

4 CLAFER ENCODING, EXPERIMENTS & RESULTS

The allocation problem and application are encoded in Clafer.

4.1 Clafer and Choco

The Clafer approach binds a high level language to a reasoning paradigm. On the one side, the Clafer constructs are [5]: *Clafer* (unified concept of classes, associations and properties), *Single inheritance*, *Clafer nesting*, *Reference clafer*, *Bag*, *Set*, *Multiplicity*. Clafer supports constraints written in first-order predicate logic. It has both existential and universal quantifiers, sets, and relations, cf. [13].

On the other side, the Choco domain is a typical CSP, which is usually defined by a triplet $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ where $\mathcal{X}$ is a set of variables (*resolution variables*), $\mathcal{D}$ is a set of domains, and $\mathcal{C}$ is a set of constraints: $\forall x_i \in \mathcal{X}$, $\mathcal{D}_i$ is the domain of possible values of x_i; $\forall c_i \in \mathcal{C}$, c_i is a constraint expressed as a relation between $\{x_j\} \subseteq \mathcal{X}$. A relation should be any kind of mathematical linear or nonlinear equations, inequalities, logical formulas, and so forth. Solving a CSP consists in finding one or all the possible solutions (allowed by the constraints). The problem consists in finding an assignment of values to the set of variables (called variable instantiation) such that all constraints are satisfied. Solving is based on three main operations: i) *Domains reduction*: The solver nondeterministically assigns a variable with a value in its domain; ii) *Constraints propagation*: The solver checks and possibly removes values that cannot occur in any solution, and iii) *Backtracking*: The solver restores the domains in case of a dead-end and records the solution in case of a success.

The Clafer tool-chain [3] includes a Clafer compiler, which translates models in Clafer into an intermediate format, for reasoning and processing with back-ends tools. In our case, the *Chocosolver*[3] compiles the intermediate model down to a set of variables and constraints from the Choco constraint programming library [8]. *Chocosolver* extends the Choco library to handle Clafer-specific features, such as reasoning over relational logic [26]. Any solution in the Choco domain gets mapped back to a solution in the Clafer domain. Note that, the *Chocosolver* module defines specific propagators for Clafer constructs. Please also note that, Clafer/Choco has no support of real numbers, so techniques like scaling down and rounding to a nearest integer needed to be used [25].

4.2 Approach & experimental protocol followed

From the description in Section 3, different Clafer models where encoded, defining benchmarks to construct the experiments. We provide the full details of the modeling in a GitHub repository[4]. In the following sections are described some incremental experiments: i) on modeling strategies to encoding the deployment problem in an heterogeneous context (partial modeling **XP#1**); ii) on improving the resolution thanks to customized resolution strategy (partial modeling **XP#2**, and partial modeling **XP#3**, with domain constraints added), iii) on improving the constraints for resolution (**XP#4**, redundancy constraints). The design space size is always the same in these experiments, i.e. includes 17 functions and 4 boards, without capacity constraint taken into account, greater than 1 billion. For **XP#4**, the completeness and soundness of the solution was checked against an internal dedicated algorithm made by our domain experts. The last experiment (**XP#5**) increase the previous one with one initialization function and the variability described in Section 4.3.5 (performance evaluation only).

Performance analysis observed the first instance (10-tries average), and the overall solutions (all instances). Moreover, the *resolution variables* (Choco paradigm) and their initialization domain were observed. Experiments were carried out with the following spec.: Intel® Xeon CPU @ 2.00 GHz, Windows 2012R2 Standard, 64-bit operating system; with a JVM - Java (Oracle), Version 8.

4.3 Encoding the problem in Clafer - Domain modeling

The formalized elements from the software, hardware and allocation problem described in Section 3 are reified into Clafer as *abstract* clafers (more details in Listing 1). The following highlights specific attempts, observations and choices on the Clafer encoding.

[3]https://github.com/gsdlab/chocosolver
[4]https://github.com/IRT-SystemX/allocation-clafer-splc2020

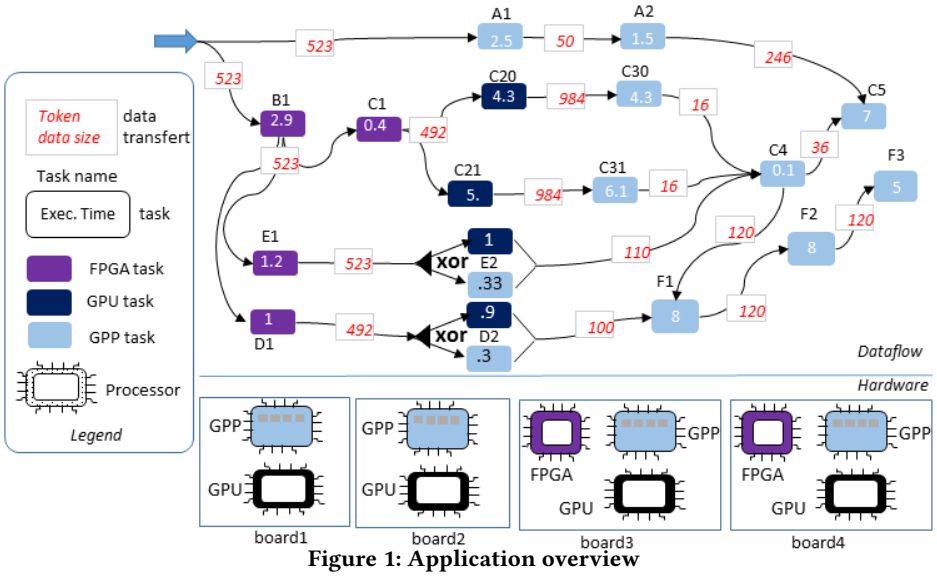

Figure 1: Application overview

4.3.1 Modeling properties and range variability. As noted above, Clafer/Choco has no support of real numbers, so techniques like scaling down and rounding to a nearest integer needed to be used [25], to represent properties like execution time (*costF*), capacities (*capaP*, *cUseP*) introduced in Section 3. Two observations are made:

Obs#1: The CSP performs its calculations on Integers (the original scales have to be transposed according to the truncation - rounding), calculation are limited to basic arithmetic operators, within the range of Big Integers (the original scales have to be transposed to ensure that they do not overflow);

Obs#2: Integer variables management is extremely costly at resolution time (large bounded *resolution variable* domain), modeling properties should be limited to the minimum needs. Therefore, pre-calculations should be done as much as possible, to go from variables to constants (reduce open ranges), and limit operations.

4.3.2 About the DeployedTo/AllocatedTo Free Variability Modeling. The *deployedTo/AllocatedTo* relation between functions and processors raises the question of how to model it in a generic and efficient way. As starting point, we reuse the micro level pattern defined in [22] to model a many-to-one (equivalently, one-to many) relationship between two sets of objects. Indeed, the open problem *deployedTo/AllocatedTo* relation is encoded as a bidirectional (opposite) one-to-many relationship (to respect the following constraints from Section 3.3.1: *C2* and *C3*). As indicated, functions and processors are specialized by dedicated FPGA, GPU, GPP elements. The question is how to encode them, as well as a specialized relationship between them (*C1* constraint).

XP1. In this first experiment, the *maxloadusage* of the GPP must not exceed 12000 (55% *maxusage*). At this stage models do not contain penalty rules. The following patterns are tested to encode the case[5]:

- *deplCase1*: Defining a constraint using a concrete collection of a given type to restrict *allocatedTo* reference as shown in listing 5. Note: the object model is not respected here.
- *deplCase2*: Defining a constraint in the specialized Clafer to restrict the bidirectional *deployedTo/AllocatedTo* relation. Constraint is applied on the many multiplicity side.
- *deplCase3*: Defining a constraint in the specialized Clafer to restrict the bidirectional *deployedTo/AllocatedTo* relation. Constraint is applied on the one multiplicity side as shown in listing 6.
- *deplCase4*: Defining two constraints in the specialized Clafers to restrict the bidirectional *deployedTo/AllocatedTo* relation. Constraints are applied on both sides.
- *deplCase5*: Using the redefinition (e.g. *deployedFTo -> FPGAProcessor* for abstract *FPGAFunction : ProcFunction*). This feature is not available in the version 0.4.3.
- *deplCase6*: Defining specialized relations at the specialization level, i.e. directly between *FPGAFunction* and *FPGAProcessor*, not at a generic level.
- *deplCase7*: Defining a specialization using features, and feature groups on one side as shown in listing 7, and a constraint on the feature selection.
- *deplCase8*: Defining a specialization using features, and feature groups on both sides, and two constraints on the feature selections.

Experimental results shown in Table 4 indicates big differences between cases that provides the 96768 instances in days, and others in more than a week. The start of the resolution (1 instance) is quite similar except for feature modeling patterns which seems inappropriate in our case. Clafer stats on the cases are also provided.

To further investigate the difference between the *deplCase2* and the *deplCase3* cases, let us take a look at the CSP *resolution variables* and their initialization domains, cf. Table 5 (the complete data set is on the GitHub). The free degree of variability in Clafer models are translated into Choco *resolution variables*, e.g. deployedFTo, into an explicit number of instances *c0_deployedFTo@RefX*, associated with an enumerate domain relating other variables, here specific processors. A finer analysis seems to be made by the *Chocosolver*

[5]Models and files are in the GitHub repository, some listings in appendix

Table 4: Performance results - patterns to specialize the allocation (*C1* constraint)

File name	deplCase1	deplCase2	deplCase3	deplCase4	deplCase5	deplCase6	deplCase7	deplCase8
T. Avrg 1 instance	03min16s	03min13s	03min14s	03min14s	not feasible	04min26s	02h25min59s	04h28min20s
T. All instances	>week	>week	1d22h08min10s	2d22h36min23s	not feasible	3d17h32min07s	>week	>week
Abstract	8	11	11	11	11	11	8	5
Concrete	37	34	34	34	40	38	38	42
Constraints	42	39	39	42	36	40	42	59
References	10	7	7	7	13	11	7	7

Table 5: Comparison of domain variable initialization

File name	deplCase2	deplCase3
Choco var domains (extract)	c0_deployedFTo@Ref0 = {0,1,2,3,4,5...,9} c0_deployedFTo@Ref4 = {0,1,2,3,4,5...,9}	c0_deployedFTo@Ref0 = {8,9} c0_deployedFTo@Ref4 = {0,1,2,3}

in the translation with the constraint on the 1 side of the 1-to-many relation. Although it concerns only the initialization of the Choco variables, surprisingly the gain seems important on a global resolution (cf. Table 4), therefore we suggest this way of modeling the relation and constraints. Note that, dealing also with the variable domain translation, there is no difference on the solver side between a reduced cardinality of a reference, e.g. *allocatedTo -> ProcFunction 0..17*, than a 0-to-many one: the domain is calculated during the translation.

Furthermore, the comparison between *deplCase3* and *deplCase4* shows that, in this case, adding constraints does not facilitate the resolution. Finally, the comparison between *deplCase3* and *deplCase6* shows that, considering the low size of the enumerate domain of the *resolution* variables, it seems more efficient to have less variable types with larger domains than more specialized variables with reduced enumerated domains. Thus, we recommend modelling the references that lead to *resolution variables* in abstract Clafers.

From this experiment we kept the *deplCase3* pattern as solution for modeling, and three observations are made at this stage:
Obs#3: Feature modeling seems not adapted to our resolution problem to represent free allocation variability;
Obs#4: Modeling of reference specializations on a 1-to-many relation seems more efficient on the one hand side (Choco domain initialization reduction, and reduced execution time);
Obs#5: Modeling free *resolution variables* in abstract Clafers, which reduces the Choco variables generated (with slightly larger enumerated domains) seems more efficient.

4.3.3 Generic Allocation Problem. From the previous experiments, observations and recommendations, the generic resource allocation problem (i.e. without domain penalties) is encoded as represented in Listing 1. In comments in the listing are the references to the constraints made explicit in Section 3.3.1.

Listing 1: Generic allocation problem

```
1  abstract ProcFunction
2   execTime -> integer
3   deployedFTo -> Processor            //C2,C3
4    [parent in this.allocatedTo]
5  abstract FPGAFunction : ProcFunction
6   [deployedFTo in FPGAProcessor]      //C1
7
8  abstract Processor
9   maxloadusage -> integer
10  totalExecTimebyP -> integer
11  [totalExecTimebyP < maxloadusage]   //C5
12  allocatedTo -> ProcFunction *
13   [this.deployedFTo = parent]
14  hostedBy -> Board
15   [parent in this.processors]
16
17 abstract GPProcessor : Processor
18  maxload -> integer = 280
19  maxusage -> integer = 65
20
21 abstract WireConnection
22  maxloadusage -> integer
23  currentload -> integer
24  [currentload < maxloadusage]    //C6,C7
25
26 abstract PCI  : WireConnection
27  [maxloadusage = 5428]
28
29 abstract Ethernet  : WireConnection
30  src2 -> Board
31  dest2 -> Board
32  [src2.ref != dest2.ref]
33  [maxloadusage = 848]
34  deployedAFConnectors -> AFConnector *
35   [this.deployedTo = parent]
36  [all afc : AFConnector | (afc in deployedAFConnectors)<=>(afc.src
        .deployedFTo.hostedBy.ref = src2.ref &&  afc.dest.
        deployedFTo.hostedBy.ref = dest2.ref)&&(!((afc.src.
        deployedFTo.ref = src2.fpga)&&( afc.dest.deployedFTo.ref =
        dest2.fpga)))]    //C4
37  [currentload = (if deployedAFConnectors then sum
        deployedAFConnectors.tokenSize  else 0)]
```

4.3.4 A Focus on the Penalties. As previously said, incoming and outgoing data exchanges between nodes host by heterogeneous processors induce extra execution time for the GPP, modeled as penalties (cf. Section 3.3.2).

Listing 4 represents the encoding of the *P8* penalty related to a local transfer on the PCI. The calculation requires two steps: a query to isolate a subset, and an operation on it. By queries we assume extraction of relevant clafers from a set of clafers. The encoding requires intermediate variables (e.g. collections like *LocalfpgagpxConnectors*) to perform some operations on the restricted subsets. Queries are represented by constraints with quantifiers, and are very expensive in terms of resolution performance. The *LocalfpgagpxConnectors* variable is used to calculate two penalties (*p8GPP* and *p8Pci*). Observations are made at this stage:
Obs#6: Domain rules requires intermediate Clafer variables, these variables are translated into *resolution* variables and induce i) less understanding of the results, and ii) extra execution time. It is important to limit their use and factorize the variables.
Obs#7: Using quantifiers (like *all*) applied on collection when building queries means repeating the operation during the resolution. These are necessary, however, we observed that the impact can

be reduced limiting the domains on which they apply (i.e. by restricting types and collections, like *FPGAAFConnector* in Listing 4, a specialization of *AFConnector*).

Obs#8: Building queries to implement domain rules (here penalties) is not trivial, setting them at the architecture language level (as opposed to directly in a CSP) make it easier for the designers.

4.3.5 Task Variant Modeling. As described in our resource allocation problem (cf. Section 3.4), the *D2 function* is kept open to be deployed to either a GPP or a GPGPU (with a reduced *execution time*). In the data-flow, *D2* sends a message to *F1*. As the encoding favors the use of Clafer specialisations to reduce sets on which quantifiers are applied (cf. previous section), on the connector side the variability has an impact on the specialized connector concerned (*GPxAFConnector* xor *GPuAFConnector*). Listing 2 represents an extract of the Clafer model. The encoding in Clafer follows the Typecasting micro level pattern defined in [22].

At the end, two sources of extensive variability exist:

- The one in the *allocation problem*: modeling various ways of mapping and deploying on a specific platform;
- The one in the *application modeling*: i) at the application level, multiple data-flows variants can achieve the functional requirements, differing in the choice between functionally-equivalent tasks executed on different processors: GPU or GPP; ii) two different hardware platforms which can differ in including an FPGA or not.

Listing 2: Task variant

```
xor D2 -> ProcFunction
 d2Gpp : GPPFunction
  [parent = this]
  [execTime = 900]
 d2Gpu : GPUFunction
  [parent = this]
  [execTime = 300]

gD2F1 : GPxAFConnector ?
  [src = D2]
  [dest = F1]
uD2F1 : GPuAFConnector ?
  [src = D2]
  [dest = F1]
[d2Gpp <=> gD2F1]
[d2Gpu <=> uD2F1]
[gD2F1 xor uD2F1]
```

4.4 Resolution phase - problem and application specific concerns

In the CSP paradigm, the topology of the search tree depends entirely on the order in which the *resolution variables* are instantiated, as well as the order in which the values are evaluated. The *resolution variables* choice heuristic determines the order in which the variables are instantiated. The value choice heuristic, on the other hand, sets the order in which the values are assigned (branching strategy). As introduced before, the *Chocosolver* binds the Clafer encoding to a set of variables and constraints from the Choco constraint programming library. In our approach, our goal is not to define a dedicated strategy from scratch, but to use the available parameters accessible in the Clafer tools.

From observing *Chocosolver* implementation, we noticed that:

Table 6: Performance results - branching priority effects

File name	deplCase3	deplCase3bD	deplCase3bA
Size	1 073 741 824	1 073 741 824	1 073 741 824
Instances	96768	96768	96768
Time	1d22h08min10s	0d14h27min28s	0d15h19min46s
Diff	0	-69%	-67%

Obs#9: The main ordering *resolution* variables of the *Chocosolver* is an implementation of the Tarjan's strongly connected components algorithm.

Obs#10: The *Chocosolver* takes into account some order in the modeling, order in collections and order in some declarations in the model. The optimallity of the specific orders is out of the concerns of this study.

4.4.1 Branching Priorities. In these experiments, the goal is not to define a dedicated strategy from scratch but to use the available parameters accessible in the Clafer tools. The following parameters are available:

- For the variable selection Clafers tools exhibits the "BranchingPriority" parameter from the Choco Solver library. Note that, branching priority affects the resolution variable order list as well as the solver resolution policy.
- For the value selection, the options are *PreferSmallerInstances* or *PreferLargerInstances* (not investigated in this study).

For our allocation problem, the identified *resolution variables* of first order are the *allocatedTo* and the *deployedFTo* references (allocation schema, bidirectional relations), cf. example in Listing 3.

Listing 3: A branching priority setup via escape

```
[choco|
branchingPriority([[c0_allocatedTo]])
|]
```

XP2. In this experiment, the *maxloadusage* of the GPP must not exceed 12000 (55% *maxusage*). At this stage, models still contain 17 functions (without the variability from Section 4.3.5), and do not contain penalties rules. On such a case, different branching priorities were tested from the *deplCase3* model, mainly:

- *deplCase3:* the reference model, without any branching priority; Ordering of decision variables are computed by Tarjan's strongly connected components algorithm;
- *deplCase3bD:* a branching priority on *deployedFTo*;
- *deplCase3bA:* a branching priority on *allocatedTo*;

Results in Table 6 show that without branching, the resolution time is about two days, and that the gain is up to 69% with the *deployedFTo* priority (down to 14h27min). The gain from the *deployedFTo* priority over the *allocatedTo* one is less important (6%). Other non-significant branching (pair combinations) were tested, providing results even worse than the reference case.

Obs#11: The branching as a significant impact, and therefore the *resolution variables* selection is essential. The Tarjan's algorithm from the *Chocosolver* module, which is a resolution strategy in itself, does not seem to fit our allocation problem and our image processing application (*deployedFTo* or *allocatedTo* never top at the beginning of the list). Depending on the targeted problem, algorithms dedicated to taking into account equations or, conversely, inequalities are to be preferred.

XP3. In this experiment, the GPP max-usage is raised up to 75%, as the domain rules were included (i.e. constraints defining the penalties). At this stage models still contains 17 functions (without the variability from Section 4.3.5). Clafer stats are: Abstract 21; Concrete 105; Reference 46; and Constraints 227. With these parameters, 316 instances are generated in:

- *deplCase10bD file* (branching *deployedFTo*): 05h56min21s
- *deplCase10bA file* (branching *allocatedTo*): 04h24min46s

Note the gain of 25% and the inversion of advantage between *deployedFTo* and *allocatedTo* compared to the previous (less constrained) case.

Obs#12: The partial model (with less constrained) shows good performances when setting a priority onto *deployedFTo*, while the complete model shows good performances when setting a priority onto *allocatedTo*. Intuitively, the branching effectiveness depends on the efficiency of the constraint in the search tree (its ability to contract the search space). One may note that, in the complete case, most of the domain constraints are set up onto the *deployedFTo* relation, which is not the best branching priority. As *allocatedTo* is the opposite of *deployedFTo*, it may implies more constraint. Additional work is needed to improve the understanding of the management of the constraints translated in the solver.

From the experiments, we cannot advice a generic way to choose branching priority set-ups, therefore, we recommend exploring and testing (on the first instances retrieved).

4.4.2 Application Specific Enhancements. Another way to improve the resolution performances is to add redundant constraints dedicated to our application. As indicated by [4], in a general context minimizing the number of constraints (and/or the number of variables) in a CSP does not necessary implies lower solving time in practice. A redundant constraint is a constraint which does not change the set of valuations satisfying a given CSP when adding this constraint to the CSP. In practice, adding well-chosen redundant constraints may speed up the solving process, or obtain a scale-up. In our case, the domain expert analyze of the application emphasizes the following redundancy constraints:

- Functions *C20* and *C21*, GPGPU dedicated, have to be deployed to different processors in order to satisfy the capacity constraint for the GPGPU (see C5);
- Functions *C20* and *C30* have to be deployed on the same board in order to satisfy the bandwidth constraint of the Ethernet connection (see C6);
- Functions *C21* and *C31* have to be deployed on the same board in order to satisfy the bandwidth of the Ethernet connection constraint (C6).

XP4. In this experiment, the above redundancy constraints are added to the model, which includes: 17 functions (without the variability) and domain rules (i.e. constraints defining the penalties). Clafer stats are: Abstract 21; Concrete 105; Reference 46; Constraints 232. The GPP *maxusage* was set up to 74%, and the solutions set are 44 instances, provided in:

- *deplCase11bD file* (branching *deployedFTo*): 04h11min12s
- *deplCase11bA file* (branching *allocatedTo*): **00h07min53s**

Note the extreme gain (97%) obtained with the best branching.

Table 7: Performance results - allocation problem

File name	deplCasebANoRC	deplCasebA	deplCasebD
Size	4 294 967 296	4 294 967 296	4 294 967 296
Instances	704	704	704
Time	9h59min45s	1h00min25s	1d02h12min41s
Constraints	269	272	272

4.4.3 Experiments on the complete version of the case study. Before analyzing architecture candidates performances for trade-offs, e.g. end-to-end response times, deadline misses, these architecture candidates must be synthesized. As quality attributes are not defined on the allocation problem at this time, optimizations strategies are impossible to define, and an exhaustive exploration of all candidate architecture is needed.

XP5. In this last experiment, variability (cf. Section 4.3.5) and an initialization function is added to the 17 functions, redundancy constraints (except for *deplCasebANoRC*) and domain rules set, and the GPP *maxusage* is set up to 73%. Table 7 shows the performance results for the different branchings (comparison of *deplCasebA* - branching *allocatedTo*, *deplCasebANoRC* same branching without redundancy constraints, and *deplCasebD* - *deployedFTo*). Note that, with a dedicated computing machine (Intel® Core™ i9-9900K CPU @ 3.60GHz), the time is reduced to 16 min 00 sec. for *deplCasebA*.

From these results, we can conclude that it is feasible to model and synthesize candidate architectures for this resource allocation problem, and that the Clafer tool-set is able of handling it. However, efforts have to be provided towards resolution to reach such a goal. When creating the case short time (<1 min) is acceptable to find 1 solution. For global final modeling few hours (<5h) is acceptable to find all solutions, which is the case with *deplCasebA* (1h with a basic set-up, or **16 min** with a dedicated machine), and even with *deplCasebANoRC* (02h 59 min with the computing machine).
One may note the symmetry of the case of application, concerning the material resources, the constitution of the boards, and their arrangement (2 boards with FPGA, and 2 boards without). Consequently, some of the solutions proposed by the solver are equivalent. This is a problem identified in the literature, and the filtering of these solutions by applying a specific algorithm is generally more efficient to eliminate "duplicate" solutions.

5 DISCUSSIONS

First, a sum-up of the observations is made:

- *Obs#1* and *Obs#2* point out the limits of the Integers to deal with properties from different dimensions and scales;
- As revealed by *Obs#3*, with the experimented version of the tool, the classical feature modeling approach is not (whereas it could intuitively be) the best option to use when modeling an allocation problem;
- *Obs#4* and *Obs#5* set the limits of the current *Chocosolver* implementation, and highlights areas for improvement regarding translation in the resolution space;
- *Obs#6*, *Obs#7*, *Obs#8* focus on constraints and queries and underlines the benefits of having to deal with ("high level") architectural concepts to express them;
- The essential element of the study is summed-up by the last 4 observations (*Obs#9*, *Obs#10*, *Obs#11*, and *Obs#12*): the

importance of the resolution. In the study, we explore some features provided, and observed that it is clearly difficult to go generic on this subject. However, focusing on the *resolution is essential to address large problems*. Besides, the results on the experiments show that the applied strategy needs to evolve when developing the model.

With this work, we outline some characteristics for architecture exploration modeling and for an associated resolution tooling.

5.1 Modeling and synthesis architectures

First, knowing how to manage different types and scales of data is a key element in addressing engineering problems.

Then, one difficulty is to maintain a genericity and modularity in the modeling, which can be done at the expense of the efficiency of the resolution. A compromise must be found between: modelling at the architectural level, which is and must be generic in nature for the architect; modelling for resolution, which to be most effective must be specific, depending on the type of problem to be solved.

Several approaches are possible to help the efficiency of the resolution: i) An implementation of specific declarative resolution strategies (which would imply adapted means of expression of these strategies) or, ii) An analyze of the model to suggest/apply adapted strategies, and iii) Guidelines to assist in the modeling and resolution.

5.2 On the use of the Clafer/Choco approach for the resource allocation problem

First of all, it is important to mention that it was possible to model an allocation problem application case and solve it with Clafer and its back-end Choco. However, efforts have to be provided towards resolution to reach such a goal.

The Clafer language provides all the constructs to address our application case study. Note that we agree with the observation from [22], that from a modeling perspective, said that an inverse relationships (opposite) can be marked. Nevertheless, as shown in our experiments, it has to be distinguished in the resolution phase (to have access for the branching).

As said before, the resolution is important. An interesting part of the Clafer approach is that the reasoning over Clafer models can naturally benefit from all performance improvements achieved in the underlying Choco library, e.g. to gain in performances from distributed solving.

As pointed out in [22], native deficiencies of Clafer like scalability issues and debugging are critical and may become the most impeding factors in modeling using Clafer. The last aspect is not trivial, and all the more necessary as the constructions of the language are not standard. As it is hard to observe and understand the solver resolution, and identify the constraints closing the resolution, we recommend that the user construct incrementally of the model, with a step-by-step verification on the result of the resolution. That is to say, applying the problem solving community approach, starting small on the domain modeling (metamodel), and also on the instances. Indeed, a model that compiles perfectly, and therefore, that is syntactically correct can rapidly be over-constrained and provide no results.

Therefore, validation could be enhanced at Clafer compile time to: i) validate the definition/instance conformity, and ii) highlight the free *resolution* variables. As it is costly to let free *resolution* variables by inadvertence (furthermore integer variables), *resolution variables* could be detected at validation, or could by tagged and set visible at the architecture language level in the Clafer language.

5.3 Threats to validity

Internal. The main threat to internal validity is the selection of the used case study. Although it was built with system and platform experts to be representative in terms of shape and complexity of the modeling, it is a single case. One other threat is our level of knowledge and expertise with Clafer, its *Chocosolver* module. We made our best to understand the capabilities of the tool consulting published works, and observing the implementation, as can be made in industrial practices. However, in future work the threat could be further mitigated by having experts from Clafer and Choco.
External. Several threats to external validity exist. We expect the proposed discussion to be applicable in other domains and applying to other engineering problems, but do not have any evidence yet of this generalization possibility. Future work will consider it.

6 CONCLUSION

In this paper, we explore the advantages of designing and configuring the variability problem to solve one of the problems of exploring (synthesizing) candidate architectures in systems engineering: the resource allocation problem.

With this work, we outlined some characteristics for architecture exploration modeling and for an associated resolution tooling. Based on the experiments, and we have successfully encoded the industrial application in Clafer, 12 observations are carried out on i) the use of the language, ii) the importance of resolution strategy.

Experiments shows that the approach of variability design problems, and more particularly Clafer, offers interesting features to address Systems Engineering problems resolution. However, works have to be done towards filling the gap between architectural formalisms and constraint programming solving paradigm, in order to obtain a gain in the resolution efficiency and a viable approach. Resolution is essential, and we suggest a practical technique to scale the existing approach to larger systems by using the branching options in the generation tool.

Facing the translation into integer problem, future work will consider extensions to Numerical Constraint Satisfaction Problems.

ACKNOWLEDGMENTS

This research work has been carried out as part of the ISC project under the leader-ship of the Institute for Technological Research SystemX, and therefore granted with public funds within the scope of the French Program Investissements d'avenir.

REFERENCES

[1] *Process Integration and Design Optimization for Model-Based Systems Engineering With SysML*, volume Volume 2: 31st Computers and Information in Engineering Conference, Parts A and B of *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, 08 2011.

[2] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya. Software architecture optimization methods: A systematic literature review. *IEEE Transactions on Software Engineering*, 39(5):658–683, 2013.

[3] Michał Antkiewicz, Kacper Bąk, Alexandr Murashkin, Rafael Olaechea, Jia Hui (Jimmy) Liang, and Krzysztof Czarnecki. Clafer tools for product line engineering. In *17th SPLC Co-Located Workshops*, SPLC '13 Workshops, page 130–135, New York, NY, USA, 2013. Association for Computing Machinery.

[4] Eugene Asarin, Thao Dang, Oded Maler, and Romain Testylier. Using redundant constraints for refinement. In Ahmed Bouajjani and Wei-Ngan Chin, editors, *Automated Technology for Verification and Analysis*, pages 37–51, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[5] Kacper Bąk, Zinovy Diskin, Michał Antkiewicz, Krzysztof Czarnecki, and Andrzej Wąsowski. Clafer: Unifying class and feature modeling. *SoSyM*, 2014.

[6] Jaclyn M. Branscomb, Christiaan J.J. Paredis, Judy Che, and Mark J. Jennings. Supporting multidisciplinary vehicle analysis using a vehicle reference architecture model in sysml. *Procedia Computer Science*, 16:79 – 88, 2013. 2013 Conference on Systems Engineering Research.

[7] G. Campeanu, J. Carlson, and S. Sentilles. Component allocation optimization for heterogeneous cpu-gpu embedded systems. In *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 229–236, 2014.

[8] Xavier Lorca Charles Prud'homme, Jean-Guillaume Fages. *Choco3 Documentation*. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2014.

[9] R. H. Dinger. Engineering design optimization with genetic algorithms. In *Northcon/98. Conference Proceedings (Cat. No.98CH36264)*, pages 114–119, 1998.

[10] J. Feljan and J. Carlson. Task allocation optimization for multicore embedded systems. In *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 237–244, 2014.

[11] Sanford Friedenthal, Alan Moore, and Rick Steiner, editors. *A Practical Guide to SysML*. The MK/OMG Press. Morgan Kaufmann, Boston, third edition, 2015.

[12] Lars Grunske, Peter Lindsay, Egor Bondarev, Yiannis Papadopoulos, and David Parker. *An Outline of an Architecture-Based Method for Optimizing Dependability Attributes of Software-Intensive Systems*, pages 188–209. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[13] GSD Lab, University of Waterloo. The Language clafer. BNF-converter (https://github.com/gsdlab/clafer/raw/master/doc/clafer.pdf), 2015.

[14] Fernando Herrera, Héctor Posadas, Pablo Peñil, Eugenio Villar, Francisco Ferrero, Raúl Valencia, and Gianluca Palermo. The complex methodology for uml/marte modeling and design space exploration of embedded systems. *Journal of Systems Architecture*, 60(1):55 – 78, 2014.

[15] Technical Committee ISO/IECJTC1/SC7. Software, systems and enterprise — architecture processes. *ISO/IEC/IEEE 42020:2019*, page 110, 07 2019.

[16] Eunsuk Kang, Ethan Jackson, and Wolfram Schulte. An approach for effective design space exploration. In Radu Calinescu and Ethan Jackson, editors, *Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems*, pages 33–54, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[17] Kyo Kang, Sholom Cohen, James Hess, William Novak, and A. Peterson. Feature-oriented domain analysis (foda) feasibility study. 01 1990.

[18] Eldar Khalilov, Jordan Ross, Michał Antkiewicz, Markus Völter, and Krzysztof Czarnecki. Modeling and optimizing automotive electric/electronic (e/e) architectures: Towards making clafer accessible to practitioners. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications*, pages 447–464, Cham, 2016. Springer International Publishing.

[19] S. Lazreg, M. Cordy, P. Collet, P. Heymans, and S. Mosser. Multifaceted automated analyses for variability-intensive embedded systems. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 854–865, 2019.

[20] Sami Lazreg, Philippe Collet, and Sébastien Mosser. Assessing the functional feasibility of variability-intensive data flow-oriented systems. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, SAC '18, page 2066–2075, New York, NY, USA, 2018. Association for Computing Machinery.

[21] Patrick Leserf, Pierre de Saqui-Sannes, and Jérôme Hugues. Trade-off analysis for sysml models using decision points and csps. *Software and Systems Modeling*, 18(6):3265–3281, Dec 2019.

[22] Murashkin, Alexandr. Automotive electronic/electric architecture modeling, design exploration and optimization using clafer, 2014.

[23] Rafael Olaechea, Steven Stewart, Krzysztof Czarnecki, and Derek Rayside. Modelling and multi-objective optimization of quality attributes in variability-rich software. In *Proceedings of the Fourth International Workshop on Nonfunctional System Properties in Domain Specific Modeling Languages*, NFPinDSML '12, New York, NY, USA, 2012. Association for Computing Machinery.

[24] A. Pretschner, M. Broy, I. H. Kruger, and T. Stauner. Software engineering for automotive systems: A roadmap. In *FOSE '07*, pages 55–71, 2007.

[25] Jordan A. Ross, Michał Antkiewicz, and Krzysztof Czarnecki. Case studies on e/e architectures for power window and central door locks systems, 2016.

[26] Jordan A. Ross, Alexandr Murashkin, Jia Hui Liang, Michał Antkiewicz, and Krzysztof Czarnecki. Synthesis and exploration of multi-level, multi-perspective architectures of automotive embedded systems. *Software & Systems Modeling*, 18(1):739–767, Feb 2019.

[27] Norbert Siegmund, Marko Rosenmüller, Martin Kuhlemann, Christian Kästner, Sven Apel, and Gunter Saake. Spl conqueror: Toward optimization of non-functional properties in software product lines. *Software Quality Journal*, 20(3):487–517, Sep 2012.

[28] Pierre-Alain Yvars and Laurent Zimmer. [deps: A language for the specification of system design problems] deps : Un langage pour le spécicification de problèmes de conception de systèmes. In *10ème Conférence Francophone de Modélisation, Optimisation et Simulation- MOSIM'14*, 11 2014.

[29] I. Švogor, I. Crnković, and N. Vrček. Multi-criteria software component allocation on a heterogeneous platform. In *Proceedings of the ITI 2013 35th International Conference on Information Technology Interfaces*, pages 341–346, 2013.

Listing 4: P8 Penalty

```
1  LocalfpgagpxConnectors -> FPGAAFConnector *     //intermediate var
2  [all afc : FPGAAFConnector | (afc in LocalfpgagpxConnectors) <=>
       ((afc.src.deployedFTo.ref = this.fpga) && ((afc.dest.
       deployedFTo.ref = this.gpp)||(afc.dest.deployedFTo.ref =
       this.gpu)))]                 //query
3  p8GPP -> integer                  //intermediate var
4  [p8GPP = (if LocalfpgagpxConnectors then #LocalfpgagpxConnectors
       * 1000 else 0)]
5  p8Pci -> integer
6  [p8Pci = (if LocalfpgagpxConnectors then sum
       LocalfpgagpxConnectors.tokenSize else 0)]
```

Listing 5: Constraint on concrete collection

```
1  abstract ProcFunction
2   deployedFTo -> Processor
3    [parent in this.allocatedTo]
4  abstract Processor
5   allocatedTo -> ProcFunction *
6    [this.deployedFTo = parent]
7  //FPGA specialisation example
8  abstract FPGAProcessor : Processor
9   [allocatedTo in mydataFlowArchitecture.localFPGAFunctions] //C1
10 //Functional data flow architecture
11 abstract FunctionalArchitecture
12  localFPGAFunctions -> ProcFunction *
13 Architecture
14  mydataFlowArchitecture : FunctionalArchitecture
```

Listing 6: Constraint on specialised Clafer

```
1  abstract ProcFunction
2   deployedFTo -> Processor
3    [parent in this.allocatedTo]
4  abstract FPGAFunction : ProcFunction
5   [deployedFTo in FPGAProcessor] // C1
6  abstract Processor
7   allocatedTo -> ProcFunction *
8    [parent in this.deployedTo]
9  abstract FPGAProcessor : Processor
```

Listing 7: Feature modeling of the specialization

```
1  abstract ProcFunction
2   deployedFTo -> Processor
3    [parent in this.allocatedTo]
4  abstract FPGAFunction : ProcFunction
5   [deployedFTo.processorKind.FPGA] //C1
6  abstract Processor
7   allocatedTo -> ProcFunction *
8    [parent in this.deployedTo]
9   xor processorKind
10   FPGA
11   GPP
12   GPU
```