Model-Driven Approach to the Optimal Configuration of Time-Triggered Flows in a TTEthernet Network

Sofiene Beji, Abdelouahed Gherbi, John Mullins, Pierre-Emmanuel Hladik

Model-driven Approach to the Optimal Configuration of Time-triggered Flows in a TTEthernet Network

Sofiene Beji[1], Abdelouahed Gherbi[2], John Mullins[1], and Pierre-Emmanuel Hladik[3]

[1] Department of Computer and Software Engineering
École Polytechnique de Montréal
firstname.lastname@polymtl.ca
[2] Department of Software and IT Engineering, École de Technologie Supérieure, Montréal, QC, Canada
abdelouahed.gherbi@etsmtl.ca
[3] LAAS-CNRS, Université de Toulouse, CNRS, INSA, Toulouse, France
pehladik@laas.fr

Abstract. The SAE standard Time-triggered Ethernet defines a strong networking infrastructure, which supports the engineering of avionic systems. Avionic functions are often designed independently and integrated to form the avionic system. The iterative integration approach helps in controlling the design complexity of evolving avionic systems and aims at minimizing the cost associate with the reconfiguration of scheduling parameters of already integrated parts. On the other hand, the iterative approach requires to specify and manage a huge set of constraints, which are then solved to compute the optimal scheduling parameters. In this paper, we focus on this issue of manual specification of these constraints by the system engineer. We propose a model-driven approach, which provides the required abstractions and automation to support the system engineer in using effectively the iterative integration approach. The abstractions consist in a metamodel, which describes the system at a given integration step and a metamodel for the constraints. The automation consists in a model transformation which enables generating automatically the relevant constraints at integration step.

Keywords: Time-Triggered Ethernet, IMA, Model-driven approach, Metamodel, Model transformation

1 Introduction

Avionic embedded systems are now engineered following the principles of Integrated Modular Avionics (IMA) architecture [1]. The IMA architecture is characterized essentially by the sharing of distributed computing resources called modules. Since avionic systems are inherently safety-critical systems, sharing these resources requires to guarantee some safety properties such as the collision-free.

Moreover, IMA-based avionic systems are distributed systems which depend on a robust and deterministic networking infrastructure. The Avionic Full Duplex Switched Ethernet (AFDX) [2] has long been adopted as a networking standard for the avionic systems. Therefore, the IMA and AFDX became the main components of a typical architecture model for the recent civil aircrafts such as $B787$ and $A380$. More recently, the SAE standard Time-Triggered Ethernet (TTEthernet) is emerging as a new standard of the avionic network [3]. With respect to AFDX, the TTEthernet standard enables to achieve a best usage of the network resources and is more deterministic. In particular, the TTEthernet network schedule is established off-line.

The avionic functions designed independently need to be integrated within an existing system deployed on a TTEthernet-networked IMA architecture. The integration of avionic system is a complex engineering task. We have presented in [4] an iterative integration approach, which enables the integration of multiple IMA partitions as well as TTEhernet frames. This approach addresses the issue of finding an appropriate scheduling for the partitions and frames. The synthesized schedule may require the reconfiguration of the already integrated frames or partitions which lead to a supplementary re-certification cost that we aim to minimize. We consider this issue as Constraints Optimization Problem (COP) [13] where we satisfy not only a set of constraints but also we optimize the reconfiguration cost function.

In order for avionic engineers to use the iterative integration approach, they have to specify both the existing system and the new avionic functionality using a set of formal constraints. These constraints can then solved effectively using Constraints Programming (CP) techniques. The number and complexity of these constraints grow up very sharply even for small system examples. Therefore, using effectively the iterative integration approach faces a challenging constraints management complexity. In order to overcome this issue, we propose in this paper a model-driven engineering approach, which provides the required abstractions (i.e. metamodels) and defines the transformation process to enable the automatic generation of these constraints.

The paper is organized as follows. Section 2 presents the background knowledge and the model for the iterative integration problem. In Section 3, we introduce our model-driven engineering approach. We dedicate Section 4 and Section 5 to present the meta-models of our approach. We define in Section 6 the transformation process that generates the constraint program for a given integration problem. Our case study is presented in Section 7. We present the related works in Section 8 and we conclude the paper in Section 9.

2 Iterative Integration on TTEthernet Networks: Backgrounds and Model

The TTEthernet is a layer 2 protocol standardized under *SAE AS6802* [3]. It defines a strategy of clock synchronization in a distributed system. The TTEthernet supports two classes of traffic: time-triggered traffic and event-triggered

traffic. The time-triggered traffic is relevant mainly for the critical applications. In this work, we are only interested in time-triggered (TT) traffic. In contrast to the event-triggered traffic, the time-triggered one is static and fixed time windows are reserved for the transmission of each frame on a given dataflow link.

A TTEthernet network can be represented by a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ represents the nodes of the network and $\mathcal{E}$ the set of physical links. The nodes are of two types: the set of End-Systems (ES) and the set of Network-Systems (NS) Each physical link connecting two nodes defines a bidirectional communications, each of which is a dataflow link.

A TTEthernet frame f_i is communicated from its source to its destinations throughout fixed paths called virtual links. A virtual link vl_i is therefore associated with each frame f_i and defines a tree structure where its nodes are a set of dataflow links. The root element of this tree is the first link on which a frame f_i is transmitted, denoted $first(f_i)$. The leaves are the last dataflow links on which the frame f_i is transmitted, designated $last(f_i)$. We denote by $next(f_i, l)$ the next dataflow links on which the frame f_i is transmitted taking as reference the dataflow link l. We denote by f_i^l the transmission of the frame f_i on the dataflow link l, $f_{i,k}^l$ the k^{th} instance of the frame f_i on the dataflow link l and by $\mathcal{L}_i$ the set of dataflow links on which f_i is defined.

The transmission of each frame f_i is characterized by the parameters: the period of f_i, $f_i.Period$, and its transmission delay on a dataflow link $f_i.Length$. The periodic pattern describing the communication of all the frames is called hyper-period (HP) and defined as the least common multiple of all frame periods. We denote by $Instances(f_i)$ the number of instances considered for the frame f_i and formally defined as $\frac{HP}{f_i.Period}$. The schedule of the k^{th} instance of the frame f_i on the dataflow link l is determined by the variable $f_i^l.Offset$ which designate the offset time with the respect to the beginning of HP. The offsets are the only variables of the integration problem.

In the iterative integration problem as defined in [9] and [4], we have some configured applications which communicate through a TTEthernet network and we want to integrate new ones. To ensure the real-time requirements, we may reconfigure the scheduling of the previous ones. We focus in the scope of this paper on the reconfiguration of the network. This reconfiguration induces an additional cost of the re-certification of the system. We designate by $Cost(f_i)$ the cost of reconfiguring a frame f_i on a given dataflow link. We denote by $\mathcal{F}$ the set of considered frames, $\mathcal{F}_{old}$ the set of configured frames and $\mathcal{F}_{new}$ the set of frames to configure.

For a configured frame f_i, we denote by $f_{i/b}^l.Offset(k)$ the offset of the k^{th} instance of the frame f_i on the dataflow link l before the integration and by $f_{i/a}^l.Offset(k)$ this offset after the integration. When it is clear from the context, we simply designate the offset after the integration by $f_i^l.Offset(k)$. We define by $R_i^l(k)$ the reconfiguration function that returns 1 if the k^{th} instance of frame f_i on the dataflow link l is reconfigured and 0 otherwise. The goal of the iterative integration problem is then to minimize the total reconfiguration cost of the

network. The reconfiguration cost of a frame instance $f^l_{i,k}$ is equal to $Cost(f_i) \times R^l_i(k)$

3 Overall Approach

We present in this section our model-driven engineering approach to automatically generate the constraints program that solves the problem of a given integration step. As shown by Figure 1, our approach relies on the definition of two meta-models. The first one, called the Integration Specification Meta-Model, characterizes an iterative integration problem on TTEthernet networks. The second one called CP Integration Meta-Model defines the CP formalization to solve this problem. An instance of the first meta-model describes a real case of the iterative integration problem. We specify in this instance the configured frames and the frames to be configured and how they are deployed on the network architecture. An instance of the second meta-model models the CP program that solves a specific iterative integration problem. A transformation tool, which relies on the meta-models, enables transforming a given instance of the Integration Specification meta-model to the corresponding CP model. The latter is then transformed to a CP code structured following the targeted CP language specification and solved by a CP solver to find a new optimal configuration. By defining an intermediate CP model before generating the CP integration code, our approach can target different CP solvers. To test our approach, we have used *MiniZinc* [6] as target CP language. We specify in the following section the Integration Specification meta-model.

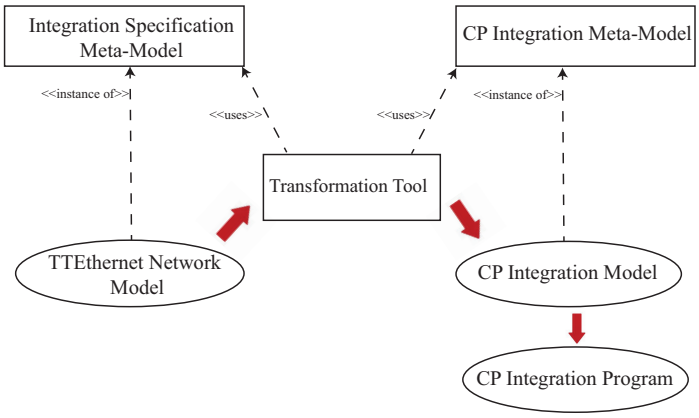

Fig. 1: Approach Overview

4 Integration Specification Meta-Model

This meta-model is depicted in Figure 2. An integration problem is based on the definition of integration steps, which are represented by the metaclass *IntegrationStep*. This metaclass has an attribute *step* which indicates the step order of the integration. It has also the attributes *switchTreatmentDelay* and *HP* which designate respectively the required delay for a switch to handle a frame and the Hyper-Period for all the frame periods. An *IntegrationStep* is composed of a set of frames to configure and the set of already configured frames if any. A *ConfiguredFrame* differs from a *FrameToConfigure* by the indication of the *actualOffsets* attribute which indicates the schedule before the integration of each instance of the considered frame on each dataflow link. A *Frame* is characterized by several attributes including *period*, *length*, *reconfigurationCost*, *nbInstances*, etc. A

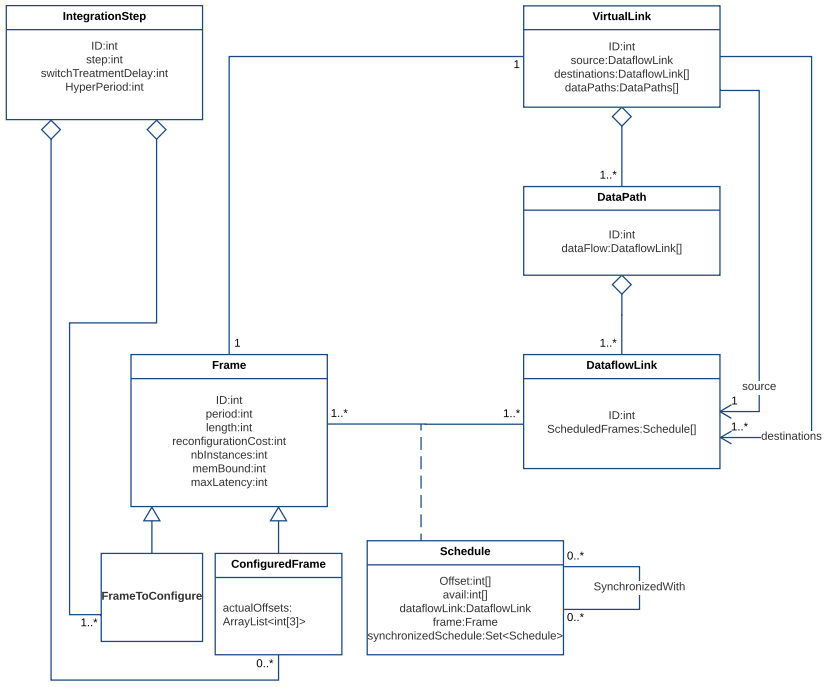

Fig. 2: The Integration Specification Meta-Model

VirtualLink is associated with each frame. Each *VirtualLink* is characterized by a source and destinations *DataflowLink*. A *VirtualLink* can be hence composed of a number of DataPaths. Each *DataPath* defines a path from the source to one destination. It is constituted by the adjacent sequence of a dataflow links. A *Schedule* characterizes the allocation of time windows of each frame on each

dataflow link. Obviously, many frames can be scheduled on a dataflow link and a frame is transmitted on the different dataflow links that defines its associated virtual link. The *Schedule* of a frame on a dataflow link may be synchronized with another of the same one on another dataflow link. This case occurs when a frame must be relayed simultaneously on different dataflow links

5 CP Integration Meta-Model

Our CP Integration Meta-Model is depicted by Figure 3. It is composed of a set of *VariableDeclaration* metaclasses, a set of *Constraint* metaclasses and a *SolveItem* metaclass. For readability purpose, we do not include in Figure 3 the comprehensive set of the relationships between these metaclasses. A *SolveItem* models a directive to the solver, which consists in the definition of two attributes, *type*, which is the type of the problem either a satisfaction or minimization problem; and *objective*, which is the optimized objective in the case of an optimization problem. In our case, the optimized objective is the reconfiguration cost.

5.1 Variables Declaration

As shown by Figure 3, three types of variables are considered to solve the integration problem. The metaclass *FrameInstanceOffsets* specifies the offsets of the different frame instances of a frame on a dataflow link. *FrameInstanceOffsets* has the two attributes *name* and *type*.The attribute *name* is of type *FrameInstanceName* and defines the name of the frame instances on one dataflow link. The *name* is therefore uniquely identified by the *IDFrame* and the *IDLink*. The attribute *type* is of type *int[]*. The metaclass *FrameInstanceReconfig* captures the information whether the offsets of the already configured frame instances are changed after the integration. It has two attributes *name* which is also the name of the considered frame instances on a dataflow link and the attribute *type* of type *bool[]*. The metaclass *ReconfigurationCost* specifies the cost of the schedule after the integration.

5.2 Constraints

To solve the integration problem, we consider nine types of constraints illustrated in Figure 3. We introduce for the definition of these constraints a new type *Quantifier* which quantifies the instance order of a frame and has four attributes: (1) *ident* which gives an identification name for the quantifier, (2) *type* which specifies the type of the quantifier (e.g. *exists* or *forall*), (3) *min* the minimum value of the quantifier and (4) *max* the max value of the quantifier.

Contention-Free Constraints: A Contention-Free Constraint expresses the mutual exclusion of a transmission on a dataflow link. Given two frames f_i and f_j transmitted on a dataflow link l, the end of transmission of an instance of f_i

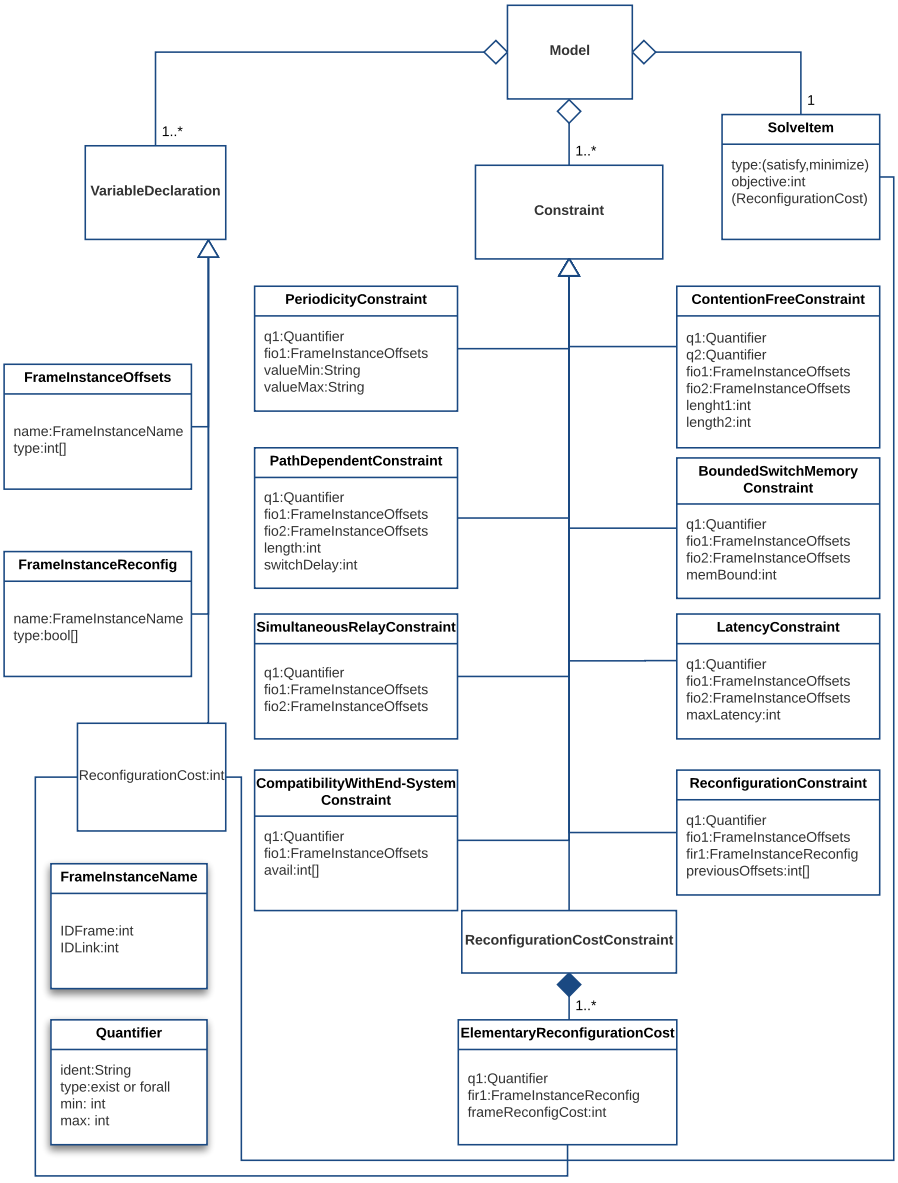

Fig. 3: The CP Integration Meta-Model

on l occurs before the beginning of transmission of a given frame instance of f_j

or vice versa. The contention free constraints can be formalized as follows:

$$\forall f_i, f_j \in \mathcal{F}, \forall l \in \mathcal{L}_i \cap \mathcal{L}_j, \forall k \in [1..Instances(f_i)], \forall k^{'} \in [1..Instances(f_j)],$$
$$\big(f_i^l.Offset(k) + f_i.Length \leq f_j^l.Offset(k^{'})\big) \vee$$
$$\big(f_j^l.Offset(k^{'}) + f_j.Length \leq f_i^l.Offset(k)\big)$$

In the CP Integration Meta-Model, a Contention-Free Constraint is modeled by the class *ContentionFreeConstraint* which has the attributes (1) *q1* and *q2* as two *Quantifier*s on respectively the instance order k and $k^{'}$, (2) *fio1* and *fio2* the offsets of respectively f_i^l and f_j^l and (3) *length1* and *length2* to designate the transmission delays of f_i and f_j on a dataflow link.

Path-Dependent Constraints: We introduce this constraint to express the sequential transmission of a frame f_i along a data path of the virtual link vl_i. Formally this constraint is defined as follows:

$$\forall f_i \in \mathcal{F}, \forall l^{'} \in next(f_i, l), \forall k \in [1..Instances(f_i)],$$
$$\big(f_i^l.Offset(k) + f_i.Length + switch\_delay\big) \leq (f_i^{l^{'}}.Offset(k))$$

where, *switch_delay* denotes the processing delay of a frame by a switch. A Path-Dependent constraint is specified in our meta-model by the metaclass *PathDependentConstraint*, which has the following attributes: (1) *q1* as a *Quantifier* on the instance order k of f_i, (2) *fio1* and *fio2* the offsets of respectively f_i^l and $f_i^{l^{'}}$, (3) *length* the parameter $f_i.Length$ and (4) *switchDelay* the processing delay of a frame by a switch.

Latency Constraints In order to ensure that frames meet their deadline requirements, we define latency constraints. These constraints bound the transmission delay of a frame along their datapaths and are formalized as follows

$$\forall f_i \in \mathcal{F}, \forall l \in last(f_i), \forall k \in [1..Instances(f_i)],$$
$$f_i^l.Offset(k) - f_i^{first(f_i)}.Offset(k) \leq max\_latency_i$$

We represent a latency constraint in our CP meta-model by the metaclass *LatencyConstraint* which has the attributes (1) *q1* to represent the instance order k, (2) *fio1* to designate the offsets of $f_i^{first(f_i)}$, (3) *fio2* to designate the offsets of f_i^l and (4) *maxLatency* the maximal tolerated bound of latency.

Reconfiguration Constraint The reconfiguration constraints detect if the already configured frames are reconfigured after the integration of new frames. The reconfiguration constraints are formalized as follows

$$\forall f_i \in \mathcal{F}_{old}, \forall l \in \mathcal{L}_i, \forall k \in [1..Instances(f_i)],$$
$$(f_{i/b}^l.Offset(k) = f_{i/a}^l.Offset(k)) \Leftrightarrow (R_i^l(k) = 0)$$

In the CP Meta-Model, a reconfiguration constraint is modeled by the meta-class *ReconfigurationConstraint* which has the attributes (1)*q1* to represent the instance order k, (2) *fio1* to designate the offsets after the integration (3) *fir1* to designate the reconfiguration variables $R_i^l(k)$ and (4) *previousOffsets* which contains the previous offsets of f_i on the dataflow link l.

6 Model Transformation Process

We detail in this section the transformation rules implemented in our *Transformation Tool* to generate automatically the CP Integration Model. We note by $ModelIn$ the integration specification model and by $ModelOut$ the CP integration model. In the remainder, we detail the transformation rule corresponding to each component of a CP Integration model.

6.1 Variables

For the variables of the CP Model, we define the transformation rule given by Algorithm 1 that generates the *FrameInstanceOffset* instances. In *Line* 1, we select from the input model an instance of the metaclass *Schedule*, denoted by A. We create in *Line* 2 an instance of the *FrameInstanceOffsets* metaclass corresponding to A that we denote by B. In *Line* $3-5$, we assign the relevant attributes.

Algorithm 1 Generation of *FrameInstanceOffset* Instances

1: **for each** $A = InstanceOf(ModelIn.Schedule)$ **do**
2: ***create*** B=***new*** $InstanceOf(ModelOut.FrameInstanceOffsets)$
3: $B.type \leftarrow int[A.Frame.nbInstances]$
4: $B.name.IDFrame \leftarrow A.frame.ID$
5: $B.name.IDLink \leftarrow A.dataFlowLink.ID$
6: **end for**

6.2 Constraints

For the constraints, we present only the transformation rule that define the *ContentionFreeConstraint* instances. This rule is specified by Algorithm 2. In *Line* $1-2$, we select from the input model two instances A and B of the metaclass *Schedule*. In order to define correctly a Contention-Free Constraint, we must check in *Line* $3-4$ that A and B are two instances that define a schedule of two different frames in the same dataflow link. To ensure constraint unicity, we impose that the *ID* of the frame associated with A is inferior that of B. We create then in *Line* 4 a new instance C of the *ContentionFreeConstraint* metaclass. In *Line* $5-6$, we define the two quantifiers of the created instance C. *q1* is reserved

for the schedule of the frame associated with the instance A and *q2* for that of B. In *Line* $7-11$, we define the attributes *fio1* and *length1* that correspond respectively the schedule and the transmission delay of the frame associated with A. Similarly, in *Line* $12-16$, we define the schedule and the transmission delay of the frame associated with B.

Algorithm 2 Generation of *ContentionFreeConstraint* Instances

```
1: for each A = InstanceOf(ModelIn.Schedule) do
2:     for each B = InstanceOf(ModelIn.Schedule) do
3:         if (A.frame.ID < B.frame.ID)and(A.dataflowLink = B.dataflowLink)
   then
4:             create C=new InstanceOf(ModelOut.ContentionFreeConstraint)
5:             C.q1 ← new Quantifier("i", forall, 1, A.Frame.nbInstances)
6:             C.q2 ← new Quantifier("j", forall, 1, B.Frame.nbInstances)
7:             for each D = InstanceOf(ModelOut.FrameInstanceOffsets) do
8:                 if (D.name.IDFrame = A.frame.ID)and(D.name.IDLink =
   A.dataflowLink.ID) then
9:                     C.fio1 ← D
10:                    C.length1 ← A.frame.length
11:                end if
12:            end for
13:            for each D = InstanceOf(ModelOut.FrameInstanceOffsets) do
14:                if (D.name.IDFrame = B.frame.ID)and(D.name.IDLink =
   B.dataflowLink.ID) then
15:                    C.fio2 ← D
16:                    C.length2 ← B.frame.length
17:                end if
18:            end for
19:        end if
20:    end for
21: end for
```

6.3 SolveItem

For the *SolveItem*, we have one instance by a CP Model (i.e. a singleton). We generate this instance by following the Algorithm 3. In *Line* 1, we check the existence of any already configured frame. This allows the definition of the nature of the problem. If no frame is already configured which is the case in *Line* $2-3$, we assign the value *satisfy* to the *type* of the problem. In the contrary case, shown in *Line* $5-6$, the problem is rather of *type minimize* and the *objective* to minimize is the *ReconfigurationCost*.

Algorithm 3 Generation of *SolveItem*

1: **if** $(nbInstancesOf(ModelIn.IntegrationStep.ConfiguredFrame) = 0)$ **then**
2: $ModelOut.SolveItem.type = satisfy$
3: $ModelOut.SolveItem.objective = null$
4: **else**
5: $ModelOut.SolveItem.type = minimize$
6: $ModelOut.SolveItem.objective = ReconfigurationCost$
7: **end if**

7 Case Study

In this section, we illustrate our model-driven engineering approach through the integration of the communication part of the distributed system whose physical architecture is illustrated by Figure 4. We identify in this figure the different dataflow links by numbers and we illustrate the direction of each flow by a dashed arrow.

The temporal characterization of the frames is given by Table 1. We propose as shown by the first column to integrate the set of frames in three integration steps. The *ID*s of the frames are indicated in the second column. The frame periods are given by the third column. We reserve the fourth column to the indication of the information availability dates at the ES level. The fifth column indicates the transmission delays of the frames on a dataflow link. We indicate in the last column the structure of the virtual link. We adopt in this field the notation $l_s - \{l_{d_1}, ..., l_{d_n}\}$ to indicate that the associated frame is transmitted first in l_s and the simultaneously in l_{d_1} to l_{d_n}. The reconfiguration cost of each frame in this example is equal to 1.

In the following, we use this case study to illustrate through two examples: (1) the integration specification in input as instance of the *Integration Specification Meta-Model*, (2) an instance of the CP Integration Meta-Model corresponding to the spec input, and (3) the associated *MiniZinc* code relative the to the CP model.

Although this example illustrates the integration of only 26 virtual links, we note that the resolution of each integration step requires about one thousand of code lines. We only illustrate some relevant aspects of our approach using two small examples. We set in the first example as goal to show the generation of some CP variables and frame constraints through the example of a Contention-Free Constraint. In the second example, we explain a constraint that exploits the structure of the virtual link.

7.1 Example 1

Focusing in the first integration step and more precisely the integration of frames f_3 and f_4, we notice that f_3 is scheduled on the dataflow links with the *ID*s 3 and 2. f_4 is scheduled on the dataflow links 3 and 8. The integration specification model corresponding to this part is given by Figure 5. We illustrate only

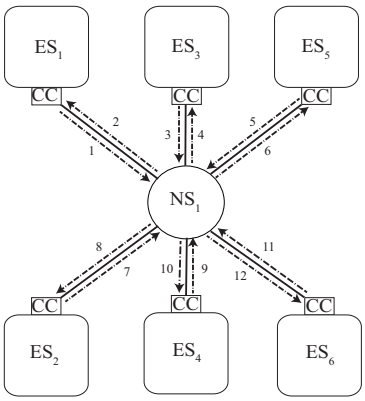

Fig. 4: Physical Architecture of the system

Integration step	Frame Id	Frame period	Availabilities	Frame length	Virtual Link
1	3	30	[7,35,65,95]	1	3-2
	4	30	[5,33,63,93]	1	3-8
	40	60	[21,21]	2	3-2
	41	60	[15,15]	4	3-8
	42	60	[17,17]	1	1-4
	43	60	[17,17]	3	7-4
	1030	60	[9,9]	2	1-4
	1031	60	[9,9]	1	7-4
2	2	30	[9,41,67,97]	2	3-8
	10	60	[51,51]	3	3-2
	11	60	[31,31]	7	3-8
	12	60	[27,27]	3	3-8
	13	30	[25,61,75,107]	4	7-2
	14	60	[71,71]	1	1-8
	15	60	[89,89]	2	7-4
	1000	30	[17,53,89,105]	1	1-4
	1001	30	[19,55,69,101]	2	7-4
	1002	60	[63,63]	1	1-4
	1003	60	[7,7]	1	7-4
3	0	60	[35,95]	1	9-4
	1	30	[0,30,60,90]	1	3-10
	109	60	[22,82]	11	11-{2,4,6,8,10}
	110	60	[14,74]	5	11-{2,4,6,8,10}
	111	60	[7,67]	2	11-{2,4,6,8,10}
	140	60	[61,81]	4	3-6
	141	60	[44,104]	1	5-4

Table 1: Temporal characteristics of integrated frames

the schedules $s1$ and $s2$ of frames $f3$ and $f4$ on the dataflow link $l3$. As shown

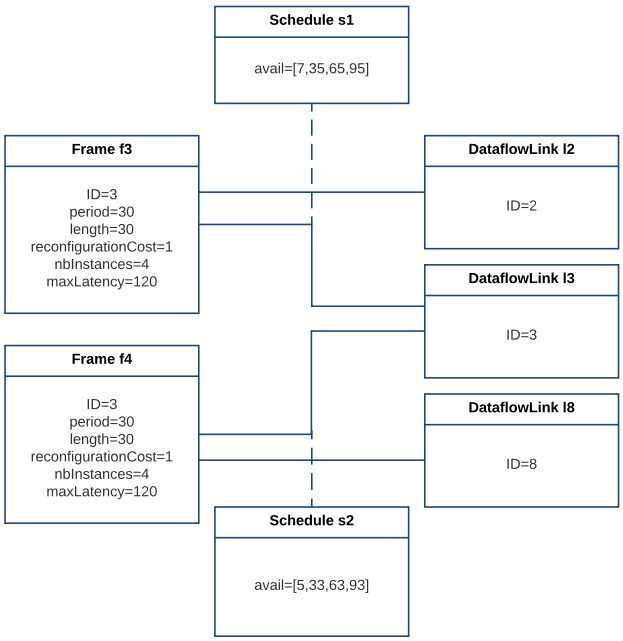

Fig. 5: Integration Specification Model of Example 1

by Figure 6, two instances $fio1$ and $fio2$ of the class *FrameInstanceOffsets* are defined in our output CP model to represent the schedules $s1$ and $s2$. As the number of instances of f_3 considered in the integration specification model is equal to 4, the *type* attribute of the variable *fio1* has a value of *int[4]*. The corresponding *MiniZinc* Code is given by Figure 7. We note that we limit our offsets to the interval [1..120] to have an enough large finite domain that represents the different possible values. Now that we have illustrated the different variables of

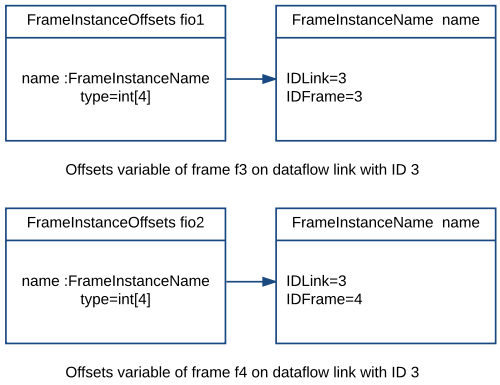

Fig. 6: CP Variables considered in Example 1

```
array[1..4]  of var  0..120:Link3Offset3;
array[1..4]  of var  0..120:Link3Offset4;
```

Fig. 7: Declaration of the variables of Example 1 in *MiniZinc*

the CP Integration Model, we present the contention-free constraint of Example 1. This constraint is defined by the instance $c1$ of the metaclass *Contention FreeConstraint* as illustrated by Figure 8. The instance $c1$ contains the information required to generate the corresponding *MiniZinc* code as illustrated by Figure 9

7.2 Example 2

The frame f_3 is transmitted on the virtual link vl_3 which is composed by the datapath $3 - 8$. The transmission of f_3 on its associated datapath induces the definition of a *Path-Dependent Constraint* that has the structure presented by Figure 10. In addition to the attributes *switchDelay* and *length*, to define this

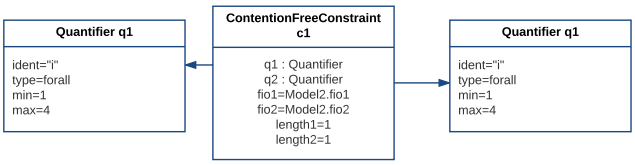

Fig. 8: Contention-Free Constraint considered in Example 1

```
constraint forall(i in 1..4,j in 1..4)
((Link3Offset3[i]+1<=Link3Offset4[j])\/
(Link3Offset4[j]+1<=Link3Offset3[i]));
```

Fig. 9: Example 1:Contention-Free Constraint Code in *MiniZinc*

constraint, we consider from Example 1 the instance $fio1$ to characterize the offsets of f_3 on the dataflow link 2. We consider also the instance $fio3$ that defines the offsets associated to the transmission of f_3 on the dataflow link 2. The corresponding *MiniZinc* code corresponding to this constraint is given by Figure 11

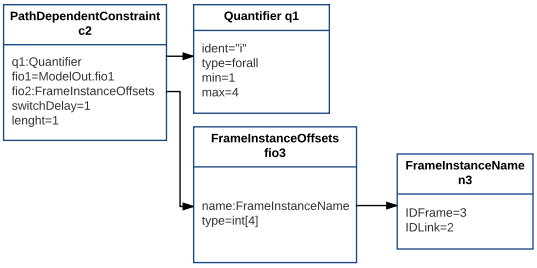

Fig. 10: CP Integration Model of Example 2

```
constraint forall (i in 1..4)
(Link3Offset3[i] + 1 + 1 <= Link2Offset3[i]);
```

Fig. 11: Example 2: Path-Dependent Constraint Code in *MiniZinc*

8 Related Works

The constraint programming approach to the scheduling problems in avionic systems is now a very active and popular research field. Several formal definitions and frameworks have been proposed for reasoning about the problem of scheduling in IMA architecture (e.g. [11, 10]), the problem of scheduling in Time-Triggered Networks (e.g [14, 15, 12, 7]) and the cost optimization problems for evolving avionic systems (e.g. [9]). The most related work to ours is the recent paper of Lauer et al. [9] and the one of Steiner [14]. In [9], the authors address the problem of an iterative integration in an *IMA* Architecture. Its objective is to find an optimal scheduling configuration that minimizes the cost of the integration. However, it does only consider the integration of IMA partitions and the proposed iterative approach handles only the scheduling of system model that evolves by adding a single partition at each iteration. The work in [4] extends the work in [9] to consider a SMT-based approach that handles not only the integration of IMA partitions but also TTEthernet flows.

The combination of model-driven software engineering approach and constraints programming approach has been the focus of some other research works including [8] and [5]. In [8], the authors propose a formalization of constraint programming solving tasks in a model-driven process chain. In [5], the authors discuss the need for a visual high level modeling language and the quality of metamodeling techniques to implement the transformations. In particular, they present a platform called s-COMMA, which efficiently implements the chain from modeling to solving constraint problems.

9 Conclusion

In this paper, we have proposed a model-driven engineering approach to support the automatic synthesis of programs that resolve the integration of TT flows of TTEthernet. The proposed approach relies on the definition of two meta-models. The first one specifies an integration problem on TTEthernet networks. The second one describes the structure of the corresponding CP program. Further to the two meta-models, this approach is based also on the definition of transformation processes that automatizes the generation of the CP model to a given integration problem. The resulted CP model is transformed to a CP code which is resolved by a CP solver to find the new optimal configuration of the network. As Future Work, we plan to extend our approach by considering the schedule of IMA partitions. We expect no difficulties to extend the two meta-models to consider the integration of IMA partitions and their associated constraints. We will define also the necessary transformation process to automatize the synthesis of IMA constraints.

Acknowledgment This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] Integrated modular avionics (ima). *AERONAUTICAL RADIO, INC.*, ARINC 653, 2006.

[2] ARINC 664, part 7: Avionics full duplex switched ethernet (afdx). *AERONAUTICAL RADIO, INC.*, 2009.

[3] AS6802: Time-triggered ethernet (ttethernet). *SAE Aerospace*, 2011.

[4] Sofiene Beji, Sardaouna Hamadou, Abdelouahed Gherbi, and John Mullins. Smt-based cost optimization approach for the integration of avionic functions in ima and ttethernet architectures. In *Proceedings of the 2014 IEEE/ACM 18th International Symposium on Distributed Simulation and Real Time Applications*, pages 165–174. IEEE Computer Society, 2014.

[5] Raphaël Chenouard, Laurent Granvilliers, and Ricardo Soto. Model-driven constraint programming. In *Proceedings of the 10th international ACM SIGPLAN conference on Principles and practice of declarative programming*, pages 236–246. ACM, 2008.

[6] Optimisation Research Group. Minizinc 2.0. http://www.minizinc.org/ide/index.html, 02 2016.

[7] Jia Huang, Jan Olaf Blech, Andreas Raabe, Christian Buckl, and Alois Knoll. Static scheduling of a time-triggered network-on-chip based on smt solving. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 509–514. EDA Consortium, 2012.

[8] Mathias Kleiner, Marcos Didonet Del Fabro, and Patrick Albert. Model search: Formalizing and automating constraint solving in mde platforms. In *European Conference on Modelling Foundations and Applications*, pages 173–188. Springer, 2010.

[9] Michaël Lauer, John Mullins, and Moez Yeddes. Cost optimization strategy for iterative integration of multi-critical functions in ima and ttethernet architecture. In *Computer Software and Applications Conference Workshops (COMPSACW), 2013 IEEE 37th Annual*, pages 139–144. IEEE, 2013.

[10] Y-H Lee, Daeyoung Kim, Mohamed Younis, Jeff Zhou, and James McElroy. Resource scheduling in dependable integrated modular avionics. In *Dependable Systems and Networks, 2000. DSN 2000. Proceedings International Conference on*, pages 14–23. IEEE, 2000.

[11] Y-H Lee, Daeyoung Kim, Mohammed Younis, and Jeff Zhou. Scheduling tool and algorithm for integrated modular avionics systems. In *Digital Avionics Systems Conference, 2000. Proceedings. DASC. The 19th*, volume 1, pages 1C2–1. IEEE, 2000.

[12] Rupak Majumdar, Indranil Saha, and Majid Zamani. Performance-aware scheduler synthesis for control systems. In *Proceedings of the ninth ACM international conference on Embedded software*, pages 299–308. ACM, 2011.

[13] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming*. Elsevier, 2006.

[14] Wilfried Steiner. An evaluation of smt-based schedule synthesis for time-triggered multi-hop networks. In *Real-Time Systems Symposium (RTSS), 2010 IEEE 31st*, pages 375–384. IEEE, 2010.

[15] Domitian Tamas-Selicean, Paul Pop, and Wilfried Steiner. Synthesis of communication schedules for ttethernet-based mixed-criticality systems. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 473–482. ACM, 2012.