Mapping Functional Behavior onto Architectural Model in a Model Driven Embedded System Design

Prachi Joshi
Virginia Polytechnic Institute and State University,VA
prachi@vt.edu

Sandeep K. Shukla
Virginia Polytechnic Institute and State University,VA
shukla@vt.edu

Jean Pierre Talpin INRIA
Rennes, France
jean-pierre.talpin@inria.fr

Huafeng Yu
Toyota-ITC, California
hyu@us.toyota-itc.com

ABSTRACT

The ever-increasing complexity in embedded systems especially in the automobile industry in the recent years has necessitated model-driven engineering. In this paper, we consider the problem of mapping functional behavior onto an architectural model captured in AADL in order to optimize end-to-end delay in executing a distributed function on the specified platform architecture. Our work presupposes that an architectural platform model is fixed due to existing hardware platform that the designers have to work with, whereas a specific functional feature is being designed in software, and implemented on the given platform. We therefore, consider the problem as a behavior modeling followed by mapping of behavioral components including computation, and communication. This mapping requires both spatial mapping as well as temporal mapping. Spatial mapping means binding of computational nodes in the behavioral model to processors/controllers etc., communications to the platform bus; and the temporal binding is done by scheduling the computation on the platform. We explain our method by way of a case study of an adaptive cruise control(ACC) system whose behavior model is represented with a data-flow graph(DFG) captured in LUSTRE. A static schedule is derived from the DFG and then mapped to the platform architecture model by formulating a non-linear optimization problem. The resulting problem being at least NP-hard, we propose use of simulated annealing or other heuristic algorithms to solve the optimization problem. The limitations of our method are discussed as well.

1. INTRODUCTION

Due to the growing complexity of embedded systems, model driven engineering (MDE) is becoming the preferred choice for their development. There are various methodologies used for MDE, one of which is to capture the architectural models and the functional/behavioral models separately and then provide a mapping for them [1]. We use this concept and present an approach for automated mapping of the behavior and architecture models in the system design phase. The problem stated in this work is specific to our on-going project on system integration in the automotive electronics and hence the solution suggested is particularly for this domain. However the approach can be extended and utilized for similar problems as discussed in future work under Section 8.

The main contribution of the paper is a methodology for mapping synchronous data-flow based behavioral model on to a target platform whose architecture is captured formally in an AADL model with the goal to optimise some system performance criteria. In our case-study the controller is modeled as a state-machine and the functional-nodes are specified in LUSTRE. A single cycle of execution starts from the instant the sensor acquires and sends a signal to the controller, which processes and sends an actuation command to the actuators. Our contribution in the paper is to map this model of data-flow to an architecture model. The sequential execution cycle assumption and the pseudo-static scheduling makes the problem formulation easier since the exploration space for the optimization is reduced. Fig.1 describes the exploration space and two possible mapping solutions in red and green.

The outline of the paper is as follows: In Section 2 we define the important terms used to describe and formulate the problem. An automated approach for the mapping problem is proposed in Section 3. An example of an adaptive cruise control system to illustrate the approach is presented in Section 4. Further, we talk about the assumptions that are taken into account in the problem formulation. In Section 5 we formulate the problem and Section 6 proposes a heuristic based solution to this problem. Section 7 discusses the related work and Section 8 gives the results and conclusion.

, April 13 - 17 2015, Salamanca, Spain
http://dx.doi.org/10.1145/2695664.2695934

2. PRELIMINARY DEFINITIONS

Since we use formal terminology in the rest of the paper, we are defining the important terms here:

Event: An event, $e \in D \times T$ where $D \in \mathbb{Z}, \mathbb{R}$ or *Boolean* and T is a tag domain that in general may indicate the sampling of a quantity, an interrupt or assignment of a new value to a variable. T could be partially or totally ordered.

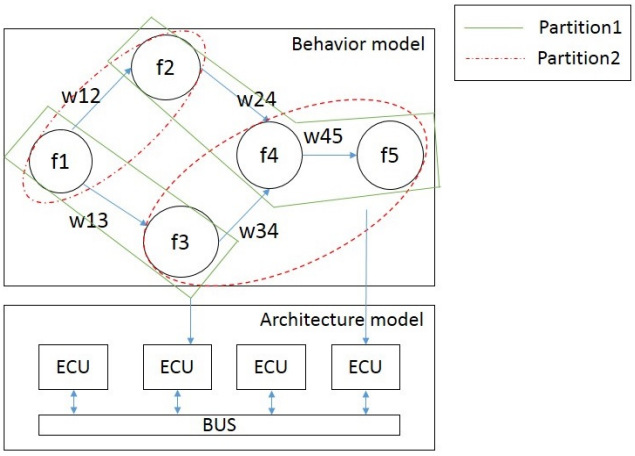

Figure 1: An example of mapping the behavior to the architecture model.

Signal: A signal comprises of a sequence (possibly infinite) of a set of events $e \in D \times T$ where T is totally ordered. For example $e_1, e_2, ..., e_n, ...$ are the events of an infinite signal that denotes the sampling of the relative velocity values for $T = (\mathbb{Z}^+, \leq)$ by the Radar.

Data Flow: A data flow model is represented by 'nodes' and 'the data connections' [2]. In this paper the data-flow graph (DFG) is a directed acyclic graph which is represented as $G = (F, E)$ where G is a graph with F as the set of vertices and $E \subseteq F \times F$ as the set of edges.

Computation: Let S and $S^{'}$ be the set of input and output signals of a system respectively. Then if there are input signals $s_1, s_2, ..., s_n \in S$ and output signals $s_1^{'}, s_2^{'}, ..., s_m^{'} \in S^{'}$ then a relation $R \subseteq S^n \times S^{'m}$ expresses a computation on $S \rightarrow S^{'}$. In case of a deterministic computation R is a function.

Behavior Model: The behavior of a computation depends on the model of computation, for example finite state machine (FSM), data-flow graph, etc., on which the computation is to be implemented. In our work we assume a synchronous data-flow model of computation as in LUSTRE. For our purpose we define a behavior model for any functionality as a data-flow model that takes as input one or multiple signals and produces as output one or multiple signals and also captures the intermediate computations on the signals. For the rest of the paper all behavioral models are LUSTRE data-flow graph models which consist of block diagrams also called *long grain data-flow* (LGDF) graphs [3]. The nodes of these data-flow graphs may be composed of atomic operators like adders, multipliers, dividers, etc. and data-flow operators like 'pre', 'followed by', 'sample' and 'current' in LUSTRE and the edges represent the data-flow connections.

Architecture Model: An architecture model (based on the execution platform of AADL) is given by a tuple $A = (P, B, M, C)$ where P is the set of Processors, B is the set of Buses, M is the set of Memory devices and C is the set of Devices in the model. For our work the set of processors P consists of $\{P_1, ..., P_6\}$ in the architecture model.

Mapping: A mapping of the behavior model to the architecture model is defined as a relation $m : \mathbb{N} \rightarrow (2^F \times 2^E) \rightarrow (P \times B)$ where $\mathbb{N}$ is the set of Natural numbers, $(F, E) = G$, and $P = A(1)$.

Static Schedule: Given a data-flow model G with a set of blocks or atomic operators, F and having signals connecting them, we can derive a static schedule by first studying the corresponding 'clock-calculus' [4] obtained using the data dependence ordering between the nodes. For the sake of simplifying the problem we assume a static schedule for the case study in this paper, so that no scheduling decision is dependent on dynamically computed data at the run-time.

3. PROPOSED APPROACH

We propose a methodology for the automation of mapping of a behavior model to the architecture model as shown Fig.2 which gives a flow-chart to present the sequence of steps that are taken. The process takes as input the data-flow graph of the system, determines its static schedule based on the data-dependency of the DFG, allocates a mapping using an optimization technique like Simulated Annealing and ends when the mapping satisfies the given constraints. This methodology is inspired from the concept of layered representation of a platform-based design as shown in Fig.3 as described by Sangiovanni-Vincentelli [6]. For this paper we follow the Fig.2 steps manually, however we aim to automate the procedure. We present a simple case study in this work to illustrate the methodology.

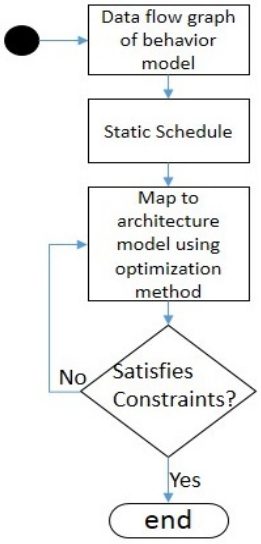

Figure 2: Automation of the mapping of behavior model to architecture model.

4. AN ADAPTIVE CRUISE CONTROL CASE STUDY

We developed a case study of a simple adaptive cruise control (ACC) system in order to illustrate the mapping of behavior model on the architecture for the system design. We use a fault tolerant data-flow (FTDF) graph [5], a variant of data-flow graph to formally present the tasks' interaction and simplify the analysis. Fig.4 inspired from [7] gives a task and data interaction of a simple ACC in

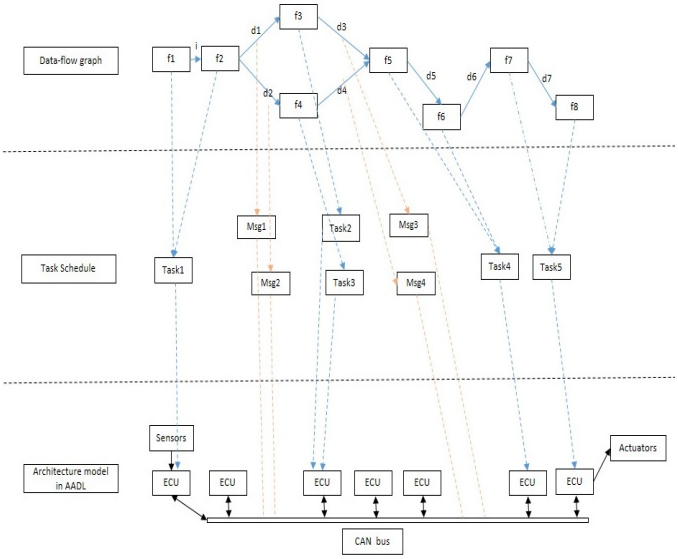

Figure 3: A layered platform based design example. The top layer shows the data-flow specification, the second layer describes its scheduling and the third layer is the architecture to which the behavior is mapped [6].

the form of an FTDF specification graph with T1 and T2 tasks designated for the controller of the system. In order

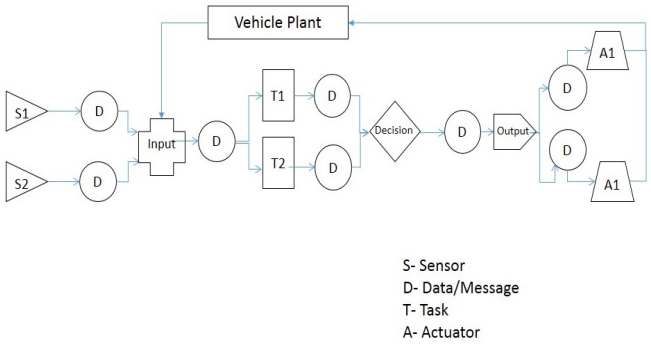

Figure 4: A FTDF graph of an adaptive cruise control system.

to model the functional and temporal aspects of the system we transform the FTDF graph as a 'data-flow' formalism. The data-flow is useful for analysis here since it is both a functional model(to represent functional/behavioral aspect) and a parallel model(to take care of sequencing and synchronization constraints arising due to data-dependencies) [2]. As described in [4] the language LUSTRE uses a program specification called 'node' which has a stream of finite or infinite values of the same type. A node has one or more output parameters similar to functions. In Fig.5 we present a data-flow graph, $G = (F, E)$ for our case study of an ACC which is based on LUSTRE code. The blocks $\{F1, F2, ...F9\} \in F$ denote the main functions/nodes in the program. The edges, $E = \{d_1, d_2, c, u, t, b, l, k, m\} \cup \{0, 1\}$ in

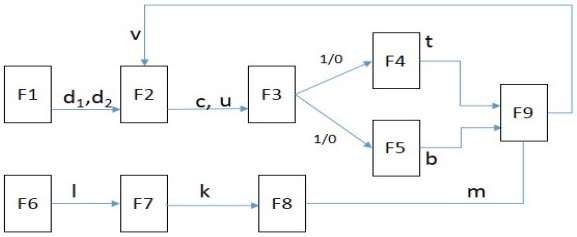

Figure 5: A data-flow graph with functions as nodes and weights as edges.

the data-flow represent the input and output data that flows from one node to another. An example of one node, F2 which has multiple conditional cases, one of which is written as follows:

```
node F2 (current_speed, d1,d2 : Real)
returns (c,u : Real);
var set_speed, K: Real;
let
if d1 > 200 when d2
   then
    c =K*(set_speed-(0 -> pre(current_speed)));
    u =maximum(minimum(c,1), -0.5);
tel.
```

F2 takes the current_speed (v in Fig.5), d1 and d2 as input and gives c and u as outputs. Here, *maximum* and *minimum* are pre-existing nodes in LUSTRE which compute the maximum and minimum of real quantities. The next goal as per our approach is to deduce the scheduling scheme of the data-flow in Fig.5 using the clock equations. In order to write clock equations for the above code we define a function $ck : E \rightarrow Boolean$ which takes as input an edge representing a data variable and returns a Boolean expression to denote the instants when the variable is present with a value. The clock of c from the above code can be written as $ck(c) = (d_1 \wedge d_2) \wedge current\_speed$. The relationship among the signals $d1, d2$ and *current_speed* determines the value of the expression given by $ck(c)$. For example assuming $d1$ and $d2$ have equivalent clocks (since they are the same sensor outputs) and clock of $d1$ to be a subset of the clock of *current_speed* , the clock expression evaluates to $ck(c) = current\_speed$ which implies that c comes no sooner than *current_speed*. After analysing the clock equations for all the data variables we determine the scheduling of the data-flow as shown in Fig.6

4.1 Assumptions

As mentioned above one of the important assumptions taken in this approach is to consider the scheduling to be a static scheduling, which may not be the case in every graph and the computation may result in pseudo static scheduling. In addition to this, there are some more assumptions which we need to consider before proceeding with the problem formulation in the next Section. The period for each function is assumed to be known by a function $f_p : P \rightarrow F \rightarrow \mathbb{Z}^+$ that defines the period in milliseconds for each function in F associated to each processor in P. The processor utilization of a function on any processor can be estimated using a utilization function on a processor and function set

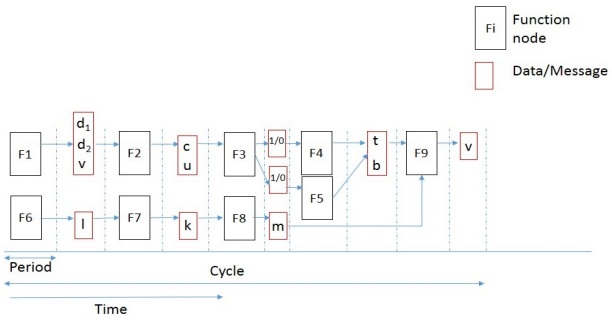

Figure 6: A task schedule graph of the data-flow graph in Fig5.

to their corresponding utilization in percentage given by $f_u : P \rightarrow F \rightarrow [0, 100]$. The worst case execution time (WCET) for each function on each processor can be estimated and is defined by a WCET assignment function $f_{WCET} : P \rightarrow F \rightarrow \mathbb{R}^+$. Also the quantity of data on an edge of the graph is analysed and given by the function $d : E \rightarrow \mathbb{R}^+$.

5. PROBLEM FORMULATION

The problem of mapping the behavior model to the architecture model is represented here as an edge partitioning problem such that the end-to-end communication of the system is optimized. This can also be considered as a resource allocation problem, since each of the partitions of the process groups could be allocated to a processor. The aspect of performance used for optimization in this case mainly focuses on the latency of the data communication. There are other constraints imposed on this optimization problem such as the prevention of over-utilization of the processor and inclusion of distantly related functions in the same partition. These constraints will be specified in detail in the following description of the problem formulation.The description of this problem formulation assumes the knowledge of the graph $G = (F, E)$ and the functions related to it mentioned in the previous section, $< G, P, f_p, f_{WCET}, f_u, d >$. The numbering of the nodes in G can be done in a topological order if the graph is a directed acyclic graph (DAG). Now the challenge here is to find a map $m : \mathbb{N} \rightarrow (2^F \times 2^E) \rightarrow (P \times B)$ such that the data communication over the shared bus is minimized subject to the some constraints which will be described in the following paragraphs.

5.1 Constraints

The list of constraints can be edited as per requirement in order to make the problem more realistic and close to the actual restrictions. In order to mathematically represent the problem we define a binary valued function X_i^j where $F_i \in F$ and $P_j \in P$ such that:
$X_i^j = 1$ if $m(F_i) \in P_j$ and $X_i^j = 0$ otherwise. Note that since every function $F_i \in F$ needs to be mapped to a processor $P_i \in P$, therefore we can state that $\sum_{j=1}^{k} X_i^j = 1$.

Processor Utilization: After obtaining a possible solution for the partition problem we need to verify that the sum of CPU utilization for all the functions allocated to one partition are not beyond the limit of the processor. In other words, if the set of functions $S_F \in 2^F$ is mapped to $P_j \in P$ then $\sum_i X_i^j f_u(P_j)(F_i) \leq 100\%$. This inequality gives t equations if there are t processors to be mapped to functions.

Special Processors: There are certain processors which need to be assigned to specific functions and are not flexible with respect to allocation of any other functions to them. In our case the set of processors P has two special processors P_1 which is connected to the sensors and P_6 which is associated with the actuators in the architecture model. Corresponding to these processors we have functions $F_1, F_2 \in F$ which have at least one input coming from the sensors and function F_9 which has at least one output going to the actuators.We can state the constraint of special processors mathematically as $\forall i \in F_i, X_i^1 = 1$ where $F_i \in S_{Fs}$ and $S_{Fs} \in 2^F$ is a set of special functions for sensors and $\forall j \in F_j, X_j^6 = 1$, where $F_j \in S_{Fa}$ and $S_{Fa} \in 2^F$ is a set of special functions for actuators.

Closely related functions: In a partition we want to avoid those function nodes which are connected to each other by summing a series of greater than l adjacent edges. Here l is a number which is deduced by analysing the number of nodes and edges and the data weights on each edge. The deduction of l is another optimization problem in itself. In order to implement this we consider the edge adjacency matrix of the given data-flow graph. The adjacency matrix, A' for a graph with n edges is an $n \times n$ matrix which has an entry $(i, j) = 1$ if the i^{th} and j^{th} vertices are connected and 0 if the vertices are not connected. The number of 2-step sequences between any two nodes in a graph can be obtained from the $(i, j)^{th}$ entry in the matrix A'^2. Similarly the number of k-step sequences between two nodes in a graph can be obtained from the $(i, j)^{th}$ entry in the matrix A'^k. Hence in order to limit the partition to a maximum l -step sequence from each node, we need to ensure that the adjacency matrix of the graphs formed by each partitions A' is such that $A'^{k+1} = 0$.

5.2 Weights on edges

The weights on the edges are a measure of the amount of data flowing between the two nodes. Hence they depend on the frequency of the update of data and the quantity of this data. In order to assign weights to an edge between any two nodes we need to measure the amount of data that flows between them since their corresponding weights would be directly proportional to this data flow. As mentioned above we use the function $d : E \rightarrow \mathbb{R}^+$ to obtain the data-flow weights for the edges. Table.1 gives the weights for each edge in the data-flow graph of Fig.5. Here c refers to the control data type while d is the data type which has an alphanumeric quantity.

5.3 Optimization of the communication latency

We aim to create partitions on the data-flow graph to minimize the inter-functional data-flow latency occurring between different partitions over the bus. Thus, we need to group the function nodes in such a way that communication for functions takes minimum time. In order to optimize this data-flow latency we minimize the sum of the weights of the edges of the graph that belong to different partitions.
Let us consider another binary valued function Y_{ij} such that if F_i and F_j, $(i \neq j)$ are allocated to processors P_u and P_v respectively, $Y_{ij} = 1$ if $u \neq v$ and $Y_{ij} = 0$ if $u = v$.

Table 1: Data weight of the edges in the DFG

Functions	Data Weight
e12	4d
e23	2d
e34	1c
e35	1c
e49	1d
e59	1d
e67	2d
e78	1d
e89	1d
e92	2d

In order to obtain an optimal data flow over the bus we need to minimize the weights on edges occurring between different processors. Therefore minimize $\sum_{e_{ij} \in E} Y_{ij} X_i^u X_j^v d(e_{ij})$

subject to $\sum_{t=1}^{k} X_i^t = 1$ where t denotes the number of processors in the architecture model.

6. SOLUTION

The solution to the above stated problem would give tasks/-functions which can be allocated to each processor such that the communication between different processors is optimized. However, each processor can be allocated only a certain number of tasks based on the utilization of each task on each processor. Therefore we need to check for the processor utilization constraint as discussed in the Section: 5.1. Another important criterion for the allocation of a task to a processor is the schedulability of the processor for the allocated tasks. Even if a set of tasks does satisfy the processor utilization constraint, it does not guarantee the tasks' completion within the deadline of task. Additional analysis requires the knowledge of the period, deadline and priority of the task in addition to the WCET and the scheduling scheme used by the processor. Hence we have to do an analysis on the scheduling of the processes like utilization based analysis or response time analysis(RTA) [18].
Consider the data-flow graph of the ACC as shown in Fig.5 This system has 9 functions, 9 edges and there are 6 processors present in the architecture specification. Since the problem formulation described above is NP-complete, we provide a heuristic based solution for the mapping problem. First, let us assign the functions F_1, F_6 and F_9 to processors P_1 and P_6 respectively as per the first constraint. In our example we manually allocate the functions to the processors in all possible ways and verify the required constraints. If any of the constraints do not satisfy then we re-allocate the tasks and so on. Manual analysis of this problem results in the following data communication combinations (which have the least latencies) for the given example:

1. F_2, F_3 are allocated to P_2 , F_7, F_8 are allocated to processor P_3 and F_4, F_5 are allocated to processor P_4. The edge data-flow content is:
   $F_d(e_{12})+F_d(e_{34})+F_d(e_{67})+F_d(e_{59})+F_d(e_{89})+F_d(e_{92})$

2. F_2, F_7 are in P_2 , F_3, F_8 are allocated to processor P_3 and F_4, F_5 are allocated to P_4. The edge data-flow content is:

Table 2: Period, Computation and Priority of F2,F3 and F4

Functions	Period T	Computation Time C	Priority P
F2	80	40	1
F3	40	10	2
F4	20	5	3

   $F_d(e_{12})+F_d(e_{67})+F_d(e_{23})+F_d(e_{78})+F_d(e_{34})+F_d(e_{89})+F_d(e_{59})$

3. F_2, F_3, F_4 and F_5 are in processor P_2 and F_6, F_7 and F_8 are in processor P_3. The edge data-flow content is: $F_d(e_{12}) + F_d(e_{59}) + F_d(e_{67}) + F_d(e_{89}) + F_d(e_{92})$

After putting in the respective function values of data weights in the above analysis we can see that the minimum data communication is achieved in the third option when F_2, F_3, F_4, F_5 are assigned to the processor P_2 and F_7, F_8 are assigned to another processor P_3 which is also intuitive since in this case there is least inter functional bus communication required. However, we also need to verify the scheduling of the processor and ensure that it satisfies the deadlines of each of the functions. We use the RTA method which computes the worst-case response time of a task and then checks with its corresponding deadline [18]. For the Response Time Equation, $R_i = C_i + \sum_{j \in hp(i)} \lceil \frac{R_i}{T_j} \rceil \cdot C_i$ where,
R_i is the response time of function i,
C_i is the computation time of function i,
T_i is the time period of function i and
$hp(i)$ is the set of functions that have higher priority than function i.
By calculating the fixed point of R_i i.e. $R_i = f(R_i)$ where $f(R_i)$ is the given equation, we can obtain a solution. Let us consider Table.2 (whose data is taken from [18]) for the three functions F_2, F_3 and F_4 (since the functions F_4 and F_5 occur exclusively).

Calculation of Response time:
$R_{F4}^0 = 5$. $R_{F4} = 5$.
$R_{F3}^0 = 10 + \lceil \frac{5}{20} \rceil \cdot 5 = 15$
$R_{F3}^1 = 10 + \lceil \frac{15}{20} \rceil \cdot 5 = 15$. Thus $R_{F3} = 15$
$R_{F2}^0 = 40$
$R_{F2}^1 + \lceil \frac{40}{20} \rceil \cdot 5 + \lceil \frac{40}{40} \rceil \cdot 10 = 60$
$R_{F2}^2 + \lceil \frac{60}{20} \rceil \cdot 5 + \lceil \frac{60}{40} \rceil \cdot 10 = 75$
$R_{F2}^3 + \lceil \frac{75}{20} \rceil \cdot 5 + \lceil \frac{75}{40} \rceil \cdot 10 = 80$
$R_{F2}^4 + \lceil \frac{80}{20} \rceil \cdot 5 + \lceil \frac{80}{40} \rceil \cdot 10 = 80$. Thus $R_{F2} = 80$
On computing the response time for each function we get the following values, $R_1 = 80$, $R_2 = 15$ and $R_3 = 5$ which sums to 100 % utilization.
Check for meeting deadline:
In order for the functions to meet their respective deadlines RTA performs a check $R_i \leq C_i + I_i$, where I_i is the influence of the higher priority tasks over function F_I . For R_{F4} , since there is no higher priority function we can see that $R_{F4} = C_{F4}$. Also, we can observe that for F_3 , $R_{F3} \leq C_{F3} + \lceil \frac{R_{F3}}{T_{F4}} \rceil \cdot C_{F4}$ and for F_2 , $R_{F2} \leq C_{F2} + \lceil \frac{R_{F2}}{T_{F4}} \rceil \cdot C_{F4} + \lceil \frac{R_{F2}}{T_{F3}} \rceil \cdot C_{F3}$. Thus, all the functions meet their given deadlines. Therefore we choose the third option as the best optimal allocation of functions in F to the processors in P . Fig.7 shows a sample partitioning solution and processor allocation for the function-dataflow graph in Fig.5.

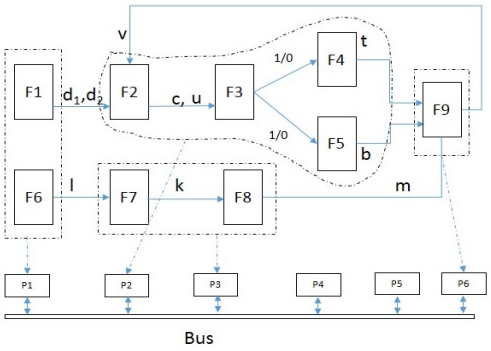

Figure 7: A solution for the allocation of the functions in the DFG to the processors in the architecture model.

7. RELATED WORK

The problem formulated in this work is specific in itself due to its constraints and requirements and as such it was solved in a particular fashion as presented in this work. However some relevant work in literature are mentioned in this section. An insightful perspective of the challenges present in the design of embedded systems today, especially focusing on automotive systems is given by [6]. In [8] the authors analyze and obtain the optimal synchronization model and communication by considering the trade-offs presented by the purely periodic and the priority based data-driven scheduling methods on the basis of the end-to-end latencies and the jitter. They use Integer Linear Programming (ILP) for the optimization problem formulation. A similar formulation is in [9] where the period optimization problem is represented using Geometric Programming (GP) and Mixed Integer Geometric Programming (MIGP). They conclude from the results that without modifying the set allocations or priority assignments, it is possible to reduce end-to-end latencies in a system by optimizing period assignment. In all the work mentioned above the problem of mapping of functions to the desired execution platform is not directly addressed. In addition to this another set of relevant work was presented in [11] and [12]. The first one talks about integration of a high-level language for formal behaviour modelling in the component based design to analyse system properties in the early development phase. The latter describes real-time modeling of heterogeneous components using BIP(Behavior Interaction Priorities) framework. A number of algorithmic approaches for mapping have been proposed [13], [14]. The important approaches are non-iterative and iterative heuristics. Intuitively iterative approaches yield better results than non-iterative ones as they cover many different possible solutions. Some of the important heuristics based iterative algorithms are Simulated Annealing (SA) [15], Genetic Algorithms (GA) [16], Tabu Search (TS) [17], local search heuristics, load balancing techniques and Monte Carlo. We intend to use some of these algorithms for scaling and automating our approach as future work.

8. RESULTS AND CONCLUSION

Our motive in this work is to be able to find mapping solutions to assign/allocate the functions in the behavior model to ECUs in the architecture model in such a way so as to optimize the data-flow over the communication bus. In order to achieve this we have formulated the problem mathematically, in the form of a non-linear optimization problem and we provide a heuristic based solution to the problem in this work. A case study from the automotive industry has been taken as an example to show the approach and analyze the results. This mapping is extremely important for an efficient and safe system integration as it provides an analysis of the system properties at an earlier stage of development.
The solution presented here is based on certain assumptions like limiting the scheduling of the computation to be a static one and having knowledge about the period, processor utilization and WCET for each function with respect to each processor and knowledge of the weights on the edges of the data-flow graph.
We wish to extend the work presented here by automating the approach proposed in this paper, which is better suited for a scaled version of the problem. The approach can be broadened and used for optimizing other parameters of the system design like power utilization, cost, performance, etc. In addition to that the solution for any mapping problem can be made more realistic by imposing constraints that are as close to industrial projects as possible. Hence we intend to include more pragmatic constraints to improve the modeling of the problem. Some other optimization techniques mentioned in Section 7 may also be explored to provide solutions for this problem.

9. REFERENCES

[1] Anderson, Matthew, and Sandeep K. Shukla. “APECS: An AADL and polychrony based embedded computing system design environment with an elevator control case study.” MEMOCODE, 2013 Eleventh IEEE/ACM International Conference on IEEE, 2013.

[2] Dennis, Jack Bonnell. “Data flow supercomputers.” Computer 13.11 (1980): 48-56.

[3] Babb, Robert G. “Parallel processing with large-grain data flow techniques.” Computer 17.7 (1984): 55-61.

[4] Pilaud, Daniel, N. Halbwachs, and J. A. Plaice. “LUSTRE: A declarative language for programming synchronous systems.” Proceedings of the 14th Annual ACM Symposium on Principles of Programming Languages (14th POPL 1987). ACM, New York, NY. Vol. 178. 1987.

[5] Pinello, Claudio, Luca P. Carloni, and Alberto L. Sangiovanni-Vincentelli. “Fault-tolerant deployment of embedded software for cost-sensitive real-time feedback-control applications.” Proceedings of the conference on Design, automation and test in Europe-Volume 2. IEEE Computer Society, 2004.

[6] Sangiovanni-Vincentelli, Alberto, and Marco Di Natale. “Embedded system design for automotive applications.” IEEE Computer 40.10 (2007): 42-51.

[7] Sangiovanni-Vincentelli, Alberto, and Grant Martin. “Platform-based design and software design methodology for embedded systems.” IEEE Design & Test of Computers 18.6 (2001): 23-33.

[8] Zheng, Wei, et al. “Synthesis of task and message activation models in real-time distributed automotive systems.” Proceedings of the conference on Design,

automation and test in Europe. EDA Consortium, 2007.

[9] Davare, Abhijit, et al. "Period optimization for hard real-time distributed automotive systems." Proceedings of the 44th annual Design Automation Conference. ACM, 2007.

[10] Racu, Razvan, Marek Jersak, and Rolf Ernst. "Applying sensitivity analysis in real-time distributed systems." Real Time and Embedded Technology and Applications Symposium, 2005. RTAS 2005. 11th IEEE. IEEE, 2005.

[11] Vulgarakis, Aneta, et al. "Integrating behavioral descriptions into a component model for embedded systems." Software Engineering and Advanced Applications (SEAA), 2010 36th EUROMICRO Conference on. IEEE, 2010.

[12] Basu, Ananda, Marius Bozga, and Joseph Sifakis. "Modeling heterogeneous real-time components in BIP." Software Engineering and Formal Methods, 2006. SEFM 2006. Fourth IEEE International Conference on. Ieee, 2006.

[13] Orsila, Heikki. "Optimizing algorithms for task graph mapping on multiprocessor system on chip." Tampereen teknillinen yliopisto. Julkaisu-Tampere University of Technology. Publication; 972 (2011).

[14] Braun, Tracy D., et al. "A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems." Heterogeneous Computing Workshop, 1999.(HCW'99) Proceedings. Eighth. IEEE, 1999.

[15] Brooks, S. P., and B. J. T. Morgan. "Optimization using simulated annealing." The Statistician (1995): 241-257.

[16] Konak, Abdullah, David W. Coit, and Alice E. Smith. "Multi-objective optimization using genetic algorithms: A tutorial." Reliability Engineering & System Safety 91.9 (2006): 992-1007.

[17] Beaty, Steven J. "Genetic algorithms versus tabu search for instruction scheduling." Artificial Neural Nets and Genetic Algorithms. Springer Vienna, 1993.

[18] The University of York, n.d. Web. 6 Oct. 2014. <http://www.cs.york.ac.uk/rts/books/CRTJbook/Lecture19.ppt> <http://www.cs.york.ac.uk/rts/books/CRTJbook/Lecture13.ppt>