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

Sanddeep 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.

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 \( \text{Boolean} \) 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.

\( D \times T \)
Computation and 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 \( e_1, e_2, \ldots, e_n, \ldots \) 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 \( s_1, s_2, \ldots, s_n \in S \) and output signals \( s_1', s_2', \ldots, s_m' \in S' \) then a relation \( R \subseteq S^n \times S^m \) expresses a computation on \( S \to S' \). 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 \( \{P_1, \ldots, P_n\} \) in the architecture model.

**Mapping**: A mapping of the behavior model to the architecture model is defined as a relation \( m : \mathbb{N} \to (2^F \times 2^E) \to (P \times B) \) 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.](image)

### 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 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:

\[
\text{node F2 (current\_speed, d1, d2 : Real) returns (c, u : Real)}; \\
\text{var set\_speed, K : Real; let} \\
\text{if d1} > 200 \text{ when d2} \\
\text{then} \\
\text{c} = K \ast (\text{set\_speed} - (0 \rightarrow \text{pre} (\text{current\_speed}))); \\
\text{u} = \text{maximum}(\text{minimum}(c,1), -0.5); \\
\text{tel.}
\]

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 \( ck : E \rightarrow \text{Boolean} \) 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 \( ck(c) = (d_1 \wedge d_2) \wedge \text{current\_speed} \). 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 \( ck(c) = \text{current\_speed} \) 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 \( P \) 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 $m : \mathbb{N} \rightarrow (2^F \times 2^B) \rightarrow (P \times B)$ 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 $X^u_{ij}$ where $F_i \in F$ and $P_j \in P$ such that:

$X^u_{ij} = 1$ if $m(F_i) \in P_j$ and $X^u_{ij} = 0$ otherwise. Note that since every function $F_i \in F$ needs to be mapped to a processor $P_j \in P$ , therefore we can state that $\sum_{j=1}^{k} X^u_{ij} = 1$.

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 $\sum_{i} X^u_{ij}(P_j)(F_i) \leq 100\%$ . 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 $P_2$ 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 $F_3$ which has at least one output going to the actuators. We can state the constraint of special processors mathematically as $\forall i \in F_i, X^v_{ij} = 1$ where $F_i \in S_F$ and $S_F \in 2^F$ is a set of special functions for sensors and $\forall j \in F_j, X^v_{ij} = 1$, where $F_j \in S_F$ and $S_F \in 2^F$ 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 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 $m \times n$ 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 $e$ 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 $u = v$. 

1. \begin{align*}
   \text{Figure 6: A task schedule graph of the data-flow graph in Fig5.}
\end{align*}
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 \( \sum_{e_{ij} \in E} Y_{ij} X_i^t X_j^t d(e_{ij}) \)
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 processors 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 \( F_1, F_4, F_6 \) to processors \( P_1, P_4, P_6 \) 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 \( P_2, P_3 \). \( F_8 \) is allocated to processor \( P_4, P_5 \) and \( F_4 \) is allocated to processor \( P_3 \). The edge data-flow content is:
\[
F_d(e_{12}) + F_d(e_{23}) + F_d(e_{34}) + F_d(e_{45}) + F_d(e_{59}) + F_d(e_{68}) + F_d(e_{92})
\]

2. \( F_2, F_7 \) are in \( P_2, P_3 \). \( F_8 \) is allocated to processor \( P_5 \) and \( F_4, F_5 \) are allocated to \( P_3 \). The edge data-flow content is:
\[
F_d(e_{12}) + F_d(e_{23}) + F_d(e_{34}) + F_d(e_{45}) + F_d(e_{68}) + F_d(e_{92})
\]

3. \( F_2, F_3, F_4 \) and \( F_5 \) are in processor \( P_2 \). \( F_6, F_7 \) and \( F_8 \) are in processor \( P_3 \). The edge data-flow content is:
\[
F_d(e_{12}) + F_d(e_{68}) + F_d(e_{67}) + F_d(e_{92})
\]

Table 1: Data weight of the edges in the DFG

<table>
<thead>
<tr>
<th>Functions</th>
<th>Data Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>e12</td>
<td>4d</td>
</tr>
<tr>
<td>e23</td>
<td>2d</td>
</tr>
<tr>
<td>e34</td>
<td>1c</td>
</tr>
<tr>
<td>e35</td>
<td>1c</td>
</tr>
<tr>
<td>e49</td>
<td>1d</td>
</tr>
<tr>
<td>e59</td>
<td>1d</td>
</tr>
<tr>
<td>e67</td>
<td>2d</td>
</tr>
<tr>
<td>e78</td>
<td>1d</td>
</tr>
<tr>
<td>e89</td>
<td>1d</td>
</tr>
<tr>
<td>e92</td>
<td>2d</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Functions</th>
<th>Period T</th>
<th>Computation Time C</th>
<th>Priority P</th>
</tr>
</thead>
<tbody>
<tr>
<td>F2</td>
<td>80</td>
<td>40</td>
<td>3</td>
</tr>
<tr>
<td>F3</td>
<td>40</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>F4</td>
<td>20</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>
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


