**Abstract**—We propose a formalism including a set of constraints for the modeling of a correct and efficient HW/SW systems. The hardware architecture is described as the interconnection of several heterogeneous MPSoCs (called tiles). The presented formalism encompasses all known tiles. The software model is described as a Kahn process network. Using a dedicated framework, we define a domain language targeting the proposed formalism along with its constraints. The goal is to specify a (1) correct hardware architecture, (2) correct software model, and (3) correct mapping which binds a software model on a hardware architecture. Finally, we show the possibility to automatically construct a set of complete models from a partial one, where the constructed models satisfy the set of predefined constraints.

I. INTRODUCTION

General purpose and embedded application programmers have a great challenge ahead, as they will experience a drastic increase in the number of processors included in integrated devices in a near future. In 2007, the International Technology Road-map for Semiconductors [1] was expecting over 250 embedded processors per device by 2015. The 2010 update of the road-map predicted more than 380 of them at this date. At chip level, scalable programming methods are required, as these many-core architectures call for more and more demanding applications, specifically in consumer appliances where multiple, evolving and computation intensive radio and multimedia standards have to be implemented quickly.

Heterogeneity has always been in use as it is the only way to achieve the performance per Watt ratio that allows to either run longer on a battery or avoid thermal issues that would lead to the use of heat-sink or fans. Widely available System-on-Chips (SoC) combine the heterogeneity of their processors, usually RISC and DSPs/VLIWs, with efficient and specific hardware accelerators, I/O infrastructure and on-chip memory organization and access. This is required to keep silicon area and power consumption cost low. In this paper, an heterogeneous SoC is called a tile. Typical examples of tiles forming the basis of the hierarchical architectures we consider are the OMAP [2], Nexperia [3] or Diopsis 940 [4] platforms. Scaling such an architecture to 100 or more computational units is a challenge for application programmers, and some tools need to be introduced to hide the complexity of the whole computation and communication infrastructures.

On the other hand, performance of embedded applications strongly depends on features of the underlying hardware platform. For this, different frameworks have been proposed for modeling HW/SW systems (e.g., SPADE [5], Sesame [6], ESMoL [7], etc.). Combining both hardware and software increases the challenge of implementing the best design. This is mainly due to the fact that, a huge number of possible designs become possible. For instance, to find the best trade-offs in terms of performance, a system designer has to take into account the network topology, the architecture (e.g., processor type, speed, etc.) the application software (e.g., the code executed by the software processes), the mapping of the software processes and software channels, the communication protocols, the operating system(s), and many other parameters. Thus, it is necessary to formalize these parameters as well as the corresponding constraints characterizing a correct and efficient design.

In this paper, we propose a formalism for the system model as a mixed HW/SW. The hardware architecture is described as the interconnection of several heterogeneous MPSoCs (called tiles). Each tile consists of a set of processors, buses, memories, interfaces, etc. The presented formalism encompasses all known tiles. The software model is described as a Kahn Process Network [8]. It consists of a set of software processes communicating through software channels. We define also the mapping which binds software and hardware. In particular, we map (1) software processes on hardware processing units, (2) channels on memories or interfaces, (3) links between processes and software channels to hardware communication paths. Writing (resp. reading) data from (resp. to) memory or interface by a processing unit requires to go through a set of devices to access the memory or the interface. This entails to define a set of communication paths which are necessary for the mapping process. Moreover, as the hardware architecture designer is the only person able to know the hardware communication paths, it is necessary to give a well-defined formalization of the hardware communication paths which will help the system designer, who is not the hardware architecture designer, to define a correct and efficient mapping.

Using the FORMULA framework [9], [10] (Formal Modeling Using Logic Programming and Analysis), we implement a domain specific language (DSL) targeting the proposed formalization. Finally, using the model finding techniques defined in FORMULA, we show that it is possible to automatically construct a set of complete models from a partial one, where the constructed models satisfy the set of predefined constraints.
constraints.

To the best of our knowledge, this is the first instance of such a formalism which gives a detailed description of (1) the hardware architecture (especially the hardware communication paths) and, (2) the application software and, (3) the mapping. Moreover, using the advantages of FORMULA, it is possible to (1) define the domain language targeting the predefined formalism, (2) find and complete the best design starting from a partial system using efficient techniques. Similar research efforts such as [11], [12], [13] have defined frameworks for modeling HW/SW. Nonetheless, these methods do not provide neither a detailed description of the hardware architecture nor a well-defined constraints models specifying a correct and efficient system model.

The paper is structured as follows. Section II presents a detailed formalism of a system model. Section III presents a domain specific language for the proposed formalism using FORMULA. Section IV is dedicated to experimental results. Section V concludes and discusses future work directions.

II. SYSTEM MODEL FORMALIZATION

In this section we introduce a formalization of the system model. The system model represents the application mapped on the hardware architecture. Thus it consists of three sub-model: (i) the application software, (ii) the hardware architecture and, (iii) the mapping.

A. Application Software Model

We consider application software defined using the Kahn process network model [8]. It consists of a set of processes communicating through FIFO channels by executing read/write operations. Notice that, we only show the process network structure, and we do not specify the functionality of each process. The only requirement for this process implementation is the exclusive use of a communication library for inter-process communication (read/write, send/recv, . . .). Formally, the application software model is defined as follows:

Definition 1 (Application Software model) An application software model A is defined as a tuple \( A = (Process_A, Channel_A, WConn_A, RConn_A) \), where:

- \( Process_A \) is a finite set of processes;
- \( Channel_A \) is a finite set of channels;
- \( WConn_A \subseteq Process_A \times Channel_A \) is a finite set of write connections;
- \( RConn_A \subseteq Channel_A \times Process_A \) is a finite set of read connections.

We consider only point-to-point and unidirectional software channels. Thus, the application software model A must satisfy the constraint \( C_A \), defined as follows:

\[
C_A = \forall (p_0, p_1 \in Process_A), (c_0, c_1 \in Channel_A) : (p_0, c), (c_0, p_1) \in WConn_A \Rightarrow p_0 = p_1 \land (c_0, c_1, p_1) \in RConn_A \Rightarrow p_0 = p_1 \tag{1}
\]

We denote by \( A \models C_A \) the fact that the application software model A satisfies constraint \( C_A \).

Example 1 Fig. 1 shows an example of an application software A. It has two software processes \( Process_A = \{P_1, P_2\} \), connected through two software channels \( Channel_A = \{C_1, C_2\} \) w.r.t. the set of write and the read connections \( WConn_A = \{(P_1, C_1), (P_2, C_2)\} \), \( RConn_A = \{(C_1, P_1), (C_2, P_2)\} \). Clearly, for this example the application software A satisfies constraint \( C_A (A \models C_A) \).

\[
\begin{align*}
&\text{x} = 1; \\
&w \text{write}(C_1, x); \\
&\text{read}(C_2, y); \\
&\text{...}
\end{align*}
\]

Fig. 1: Example of an application software.

B. Hardware architecture model

We consider hardware architectures described as the interconnections of tiles (multi-tile). Tiles are the basic elements of the hardware architecture.

The hardware architecture we are focusing on are heterogeneous for two reasons: the base tile is heterogeneous in terms of processors, memory hierarchy and I/O organization, and the tiles can be of different kinds. Moreover, the topology of the whole interconnection infrastructure is not imposed.

1) Tile

A tile consists of the interconnections of computational, interface and communication devices such as processors, buses, bridges, memories, etc. To simplify the study, we assume the presence of a special peripheral (called interface) attached to each tile of the system which makes the bridge with the outside communication fabric. Formally, a tile is defined as follows:

Definition 2 (Tile) A tile t is defined as a tuple \( t = (\mathcal{P}U_t, \mathcal{M}_t, \mathcal{B}US_t, \mathcal{B}R_t, \mathcal{I}_t, \mathcal{L}_t, \mathcal{B}_t) \), where,

- \( \mathcal{P}U_t \) is a finite set of processing units;
- \( \mathcal{M}_t \) is a finite set of memories;
- \( \mathcal{B}US_t \) is a finite set of buses;
- \( \mathcal{B}R_t \) is a finite set of bridges;
- \( \mathcal{I}_t \) is a finite set of interfaces;
- \( \mathcal{L}_t \subseteq (\mathcal{D}_t \times \mathcal{B}US_t) \cup (\mathcal{B}US_t \times \mathcal{D}_t) \) is a finite set of links, where, \( \mathcal{D}_t = \mathcal{P}U_t \cup \mathcal{M}_t \cup \mathcal{B}R_t \cup \mathcal{I}_t \). A link \((a, b) \in \mathcal{L}_t\) is denoted by \( a \rightarrow b \);
- \( \mathcal{B}_t : \mathcal{P}U_t \mapsto \mathcal{M}_t \cup \emptyset \) is a function defining the binding memory for each processing unit.

In general, processing units trigger the communication. However, they can be relieved of the actual transfer by hardware accelerators (Direct Access Memories). Consequently, we consider that some of the processing units behave as Direct Access Memories (DMAs). In this case, we bind their memories to the empty set (\( \emptyset \)). Moreover, we assume that, if we bind a memory \( m \), where \( m \neq \emptyset \), to the processing unit \( \mu \), then there must exit a bus connected to \( m \) and \( \mu \) with bidirectional links. Thus, the tile \( t \) must satisfy the constraint \( C_t \), defined as follows:

\[
C_t = B_\mu (\mu) = M \models \mathcal{M}_t \Rightarrow \exists B_\mu \mathcal{B}_\mu (\mu, B_\mu), (B_\mu, \mu), (M, B_\mu), (B, M) \in \mathcal{L}_t \tag{2}
\]
We denote by \( t \models C_t \) the fact that the tile \( t \) satisfies constraint \( C_t \).

**Example 2** Fig. 2(a) shows an example of a tile \( t \) consisting of three processing units \( \mathcal{PU}_t = \{\mu_1, \mu_2, \mu_3\} \), three memories \( \mathcal{M}_t = \{M_1, M_2, M_3\} \), three bridges \( \mathcal{BR}_t = \{B_1, B_2, B_3\} \), one interface \( I_t = \{I_1\} \), and four buses \( \mathcal{BUS}_t = \{B_1, B_2, B_3, B_4\} \). We use oriented arrows to represent the set of links. Also, we consider that we bind processing units \( \mu_1, \mu_2 \) and \( \mu_3 \) to memories \( M_1, M_2 \), and \( \emptyset \), respectively. That is, \( \mu_3 \) behaves as a DMA.

![Example of a tile.](image)

**Fig. 2:** Example of hardware architecture

2) Multi-tile

A multi-tile system is composed of a set of tiles \( \{t_i\}_{i=1}^n \) connected through a set of virtual links. Formally, a multi-tile is defined as follows:

**Definition 3 (Multi-tile)** A multi-tile system is defined by \( T = (\{t_i\}_{i=1}^n, L_T) \), where,

- \( \{t_i\}_{i=1}^n \) is a finite set of tiles, where, \( t_i = (\mathcal{PU}_t, \mathcal{M}_t, \mathcal{BUS}_t, \mathcal{BR}_t, I_t, L_t, B_t) \);
- \( L_T \subseteq \mathcal{I} \times \mathcal{I} \) is a finite set of links, where, \( \mathcal{I} = \cup_{i=1}^n \mathcal{I}_t \).

We assume that there is no link between two interfaces belonging to the same tile. Thus, the multi-tile \( T \) must satisfy the constraint \( C_T \), defined as follows:

\[
C_T = \forall \{I, I'\} \subseteq L_T : (I \in \mathcal{L}_t_i) \wedge (I' \in \mathcal{L}_t_{i'}) \Rightarrow i \neq j
\]

(3)

Given a multi-tile \( T = (\{t_i\}_{i=1}^n, L_T) \). We denote by \( \mathcal{PU}_T = \cup_{i=1}^n \mathcal{PU}_t_i \) the set of processing units of \( T \), and by \( \mathcal{M}_T = \cup_{i=1}^n \mathcal{M}_t_i \) the set of memories of \( T \).

**Example 3** Fig. 2(b) shows an example of a multi-tile \( T \) consisting of four tiles connected by a set of links between interfaces.

3) Communication paths

Writing/reading data from a memory of a given processing unit to/from another memory, requires to pass through a list of hardware components (e.g., bus, bridge). We call this a hardware communication path. In order to facilitate the mapping of an application software on the hardware architecture, it is necessary to define the set of valid communication paths. In general, we distinguish two types of communication paths: (1) write communication paths, and (2) read communication paths. A write communication path represents the communication infrastructure of the tile needed to connect the local memory of a given processing unit \( \mu \) to an interface or another memory. A read communication path represents the communication infrastructure of the tile needed to connect a memory or an interface to the local memory of a given processing unit \( \mu \). In general, the processing unit triggers the communication. However, they can also be relieved of the actual transfer by hardware accelerators (DMA). In the following, we define the write and read communication path as the concatenation of smaller paths (sub-paths).

**Preliminaries and notations.** We introduce some preliminary concepts and notation. Consider a finite set of elements \( E \). We denote by \( E^* \) the set of finite sequences over \( E \). The length (number of elements) of a finite sequence \( \sigma \) is denoted \( |\sigma| \) and the \( (i+1) \)-th element of \( \sigma \) is denoted by \( \sigma_i \). For two sequences \( \sigma, \sigma' \in E^* \), we denote by \( \sigma . \sigma' \) the concatenation of \( \sigma \) and \( \sigma' \). For \( \sigma \in E^* \) and \( n \in \mathbb{N} \), when \( |\sigma| > n \), the sub-sequence \( \sigma_n \)... is the sequence containing all elements of \( \sigma \) but the \( n \) first ones. Finally, for \( \sigma \in E^* \) we denote by \( \sigma^{-1} \) the inverse of sigma, that is, \( \sigma^{-1} = \sigma_{|\sigma|-1}.\sigma_{|\sigma|-2} \ldots \sigma_0 \).

Consider a tile \( t = (\mathcal{PU}_t, \mathcal{M}_t, \mathcal{BUS}_t, \mathcal{BR}_t, I_t, L_t, B_t) \). Let \( \mu, \mu' \in \mathcal{PU}_t \) and \( X \in \mathcal{M}_t \cup I_t \), we distinguish the following sub-paths:

- **Local Write** (\( LW_t \)): given a processing unit \( \mu \), \( LW_t \) specifies paths to write on the binding memory \( M = B_t(\mu) \) (local memory) of \( \mu \). In other words, local write specifies the paths for store operations. Equation 4 specifies local write paths. They consists of the processing unit \( \mu \), the bus \( B \), and the binding memory \( M \), respectively. Where \( B \) is the bus connecting \( \mu \) and \( M \).

- **Local Read** (\( LR_t \)): given a processing unit \( \mu \), \( LR_t \) specifies paths to read from the binding memory \( M = B_t(\mu) \) of \( \mu \). In other words, local read specifies paths for load operations (see Equation 5).

- **Sub Write** (\( SW_t \)): given a processing unit \( \mu \), and an interface (or memory) \( X \), \( SW_t \) specifies all paths to access the interface (or memory) starting from \( \mu \). Equation 6 specifies sub write paths. They are equal to sequences starting from \( \mu \) arriving to \( X \) by crossing buses and bridges consequently. Such that, there must exist links between the crossing elements. Moreover, a path should not contain a cycle.

- **Sub Read** (\( SR_t \)): given a processing unit \( \mu \), and an interface (or memory), \( SR_t \) specifies all paths to access this interface (or memory) starting from \( \mu \) (see Equation 7).

- **Request Transfer** (\( RT_t \)): given two processing units \( \mu \) and \( \mu' \), where \( \mu' \) behaves as DMA. That is, \( B(\mu') = \emptyset \). \( RT_t \) specifies all paths from \( \mu \) to \( \mu' \) (see Equation 8).

Formally, we define these sub-paths as follows:
an example of the write and read communication paths. For instance, a write communication path from the processing unit \( \mu_1 \) to the memory \( M_3 \) is defined by \( (lr, sw) \), where \( lr = \mu_1.B_1.M_1.\mu_1 \) is a local read path, and \( sw = \mu_1.B_3.B_4.M_3 \) is a sub write path. A read communication path from memory \( M_3 \) to processing unit \( \mu_2 \) is defined by \( (sr, lw) \), where \( sr = \mu_2.B_1.B_2.\mu_3.B_4 \) is a sub read path, and \( lw = \mu_2.B_2.M_2 \) is a local write path. Using the processing unit \( \mu_3 \) as DMA (since \( B_1(\mu_3) = \emptyset \), by definition), we define a write communication path from processing unit \( \mu_1 \) to memory \( M_3 \) by \( (rt, sr, sw) \), where \( rt = \mu_1.B_3.\mu_3, sr = \mu_3.B_3.Br_1.M_1.B_1.Br_1.B_3.\mu_3 \), and \( sw = \mu_3.B_3.B_4.B_3.M_3 \).

C. Mapping

Given an application software and a hardware architecture, a mapping associates software processes to hardware computing units, software channels to memories or interfaces, write software connections to write hardware paths, and read software connections to read hardware paths. Formally, a mapping is defined as follows:

**Definition 4 (Mapping)** Given an application software\( A = (Process_A, Channel_A, WConn_A, RConn_A) \) and a hardware architecture (multi-tile) \( T = (\{t_i\}_{i=1}^n, \mathcal{L}_T) \), a mapping \( Map_{A \to T} \) is defined as a tuple \( Map_{A \to T} = (ProcessMap, ChannelMap, WConnMap, RConnMap) \), where,

- \( ProcessMap : Process_A \to \mathcal{P}U_T \), maps each process of the application model to a computing unit of the system;
- \( ChannelMap : Channel_A \to \mathcal{M}_T \cup (\mathcal{I}_T \times \mathcal{I}_T) \), maps each channel to either a memory or a pair of interfaces;
- \( WConnMap : WConn_A \to WRP_{T} \), maps write software connections to write hardware paths;
- \( RConnMap : RConn_A \to RDP_{T} \), maps read software connections to read hardware paths.

A correct mapping must satisfy a set of constraints. For a channel \( c \), let \((p, c) \in WConn_A\), \((c, p') \in RConn_A\), if we map the process \( p \) to the processing unit \( \mu \in PU_t \), and the process \( p' \) to the processing unit \( \mu' \in PU_t \), then the following constraints must be satisfied:

- if \( \mu \) and \( \mu' \) belong to two different tiles \( t \) and \( t' \), then the channel \( c \) must be mapped to a pair of interfaces \((I, I')\), where \( I \in \mathcal{I}_t \) and \( I' \in \mathcal{I}_{t'} \);
- if \( \mu \) and \( \mu' \) belong to the same tile \( t \), then the channel \( c \) must be mapped to a memory \( m \), where \( m \in \mathcal{M}_t \);
- if we map the channel \( c \) to a pair of interfaces, there must exist a link between these two interfaces;
- if we map the write software channel \((p, c)\) to a write communication path \( \sigma \) and the channel \( c \) to a pair of interfaces \((I, I')\), then \( \sigma \) must be in \( WRP_{t}(\mu, I) \);
- if we map the write software channel \((p, c)\) to a write communication path \( \sigma \) and the channel \( c \) to a memory \( m \), then \( \sigma \) must be in \( WRP_{t}(\mu, m) \);
if the read software channel \((c, p')\) to a read communication path \(\sigma\) and the channel \(c\) to a memory \(m\), then \(\sigma\) must be in \(\mathcal{RP}_v(\mu', \mu)\);

- if we map the software process \(p\) to a processing unit \(\mu\), then we must have \(B_\mu(\mu) \neq \emptyset\), that is \(\mu\) does not behave as a DMA.

A mapping must be consistent. That is, it satisfies the constraint \(C_M = \bigwedge_{i=1}^k C_i\), where each \(C_i\) represents one of the constraints defined above. Formally, \(C_1, \ldots, C_8\) are defined as follows. For every \(c \in \text{Channel}_A, p, p' \in \text{Process}_A, (p, c) \in \text{WConn}_A, (c, p') \in \text{RConn}_A, \text{ProcessMap}(p) = \mu \in \text{PU}_i, \text{ProcessMap}(p') = \mu' \in \text{PU}_i\), we have:

\begin{align*}
C_1 &= \left\{ t \neq t' \right\} \Rightarrow \left[ \text{ChannelMap}(c) = (I, I') \wedge I \in \text{I}_i \wedge I' \in \text{I}_i \right] \\
C_2 &= \left\{ t = t' \right\} \Rightarrow \left[ \text{ChannelMap}(c) \in \text{M}_i \right] \\
C_3 &= \left[ \text{ChannelMap}(c) = (I, I') \right] \Rightarrow \left[ (I, I') \in \text{C}_i \right] \\
C_4 &= \left[ \text{WConnMap}(p, c) = \sigma \land \text{ChannelMap}(c) = M \right] \\
& \Rightarrow \left[ \sigma \in \text{WP}_v(\mu, I) \right] \\
C_5 &= \left[ \text{WConnMap}(p, c) = \sigma \land \text{ChannelMap}(c) = M \right] \\
& \Rightarrow \left[ \sigma \in \text{WP}_v(\mu, M) \right] \\
C_6 &= \left[ \text{RConnMap}(c, p') = \sigma \land \text{ChannelMap}(c) = M \right] \\
& \Rightarrow \left[ \sigma \in \text{RP}_v(\mu', M) \right] \\
C_7 &= \left[ \text{RConnMap}(c, p') = \sigma \land \text{ChannelMap}(c) = M \right] \\
& \Rightarrow \left[ \sigma \in \text{RP}_v(\mu', M) \right] \\
C_8 &= \left[ \text{ProcessMap}(p) = \mu \right] \Rightarrow B_\mu(\mu) \neq \emptyset \quad (13)
\end{align*}

### III. Domain-Specific Language and Automated Mapping

We have implemented in FORMULA modeling tool [9], [10] a domain specific language (DSL) for the formalization presented in Section II. FORMULA ( Formal Modeling Using Logic Programming and Analysis) is a formal specification language based on statically-typed constraint logic programming (CLP). FORMULA, can be used to define domain specific languages (DSLs).

Defining a new domain in FORMULA requires to specify three modules: (1) algebraic data types; (2) CLP rules; (3) CLP queries. Algebraic data types are used to specify the terms of the domain. CLP rules are used to generate intermediate terms used by CLP queries. CLP queries are used to specify constraints over the terms of the domain.

#### A. DSL for System model

As we have seen, defining a new domain requires to specify three modules. In the following we show how to use each of these modules to define our formalization. First, algebraic data types used to specify the structure of the application software model, the hardware architecture model, and the mapping. Second, using CLP rules it is possible to write rules to construct read and write hardware communication paths according to Equations 4, 5, 6, 7, 8, 9, and 10. Finally, using CLP queries we describe the set of constraints where the domain must respect. In our case, we must satisfy the constraint \(C_S = C_A \land \bigwedge_{i=1}^k C_i \land C_T \land C_M\), where constraints \(C_A, C_t, \text{ and } C_M\) are defined in Equations 1, 2, 3, and 13, respectively. Fig. 3 shows an abstract presentation of the domain representing the system model in FORMULA.

### B. Automated Mapping

As a core technique, FORMULA can be used to construct system models satisfying complex domain constraints. The user inputs a partially specified model and FORMULA searches the space of completed models until it finds a globally satisfactory design (CLP satisfiability). This is done by using symbolic execution which automatically translate DSL specifications into the state-of-the-art satisfiability modulo theories (SMT) solver Z3 [14].

After defining the domain of the system, it is possible to define either a complete or a partial model of the system which conforms to the domain. Fig. 4 shows a partial model with the description of the application software model presented in Fig. 1 and abstract presentation of the hardware architecture. A major advantage of FORMULA is its model finding and design space exploration (DSE) facilities. That is, it is possible to define a partial model describing only the application software and the hardware architecture. Then, using FORMULA we automatically complete our partial model by finding a set of correct mappings satisfying the constraints defined in the domain. Moreover, the model will be verified w.r.t. the defined constraints. Finally, a designer can pick one of the constructed complete models either randomly or according to some other information.

### IV. Experimental results

We have conducted some experiments to study (1) the efficiency to automatically generate mappings and, (2) to
check the correctness and the flexibility of the defined domain implementing the proposed formalism. For each experiment, we measure the size of the model and the execution time in order to verify the model according the defined constraints and to automatically construct correct mappings. All experiments are conducted on Intel 2.6GHz machines with 4GB RAM.

We have implemented a set of partial models conforming to the domain which implements the proposed formalism. We consider partial models consisting only of a description of the hardware architecture and the application software. That is, mappings are kept unknown. For each partial model, we increase the number of tiles and the number of processes. More precisely, we use the same tile architecture as Fig. 2(a) by removing $Br_3$. The number of processes in the application software is two times the number of tiles.

Table I shows execution times, as a function of the number of tiles, to automatically generate from a partial model a set of complete models. That is, we automatically complete the corresponding partial model by finding all the possible correct mappings satisfying the constraints defined in the domain. Moreover, Table I shows the number of terms (e.g., number of processing units, memory, links, processes, \ldots) of the constructed models. Notice that, the execution time to compute all possible mappings for a system consisting of 50 tiles requires only 8.1 seconds.

![Fig. 4: An example of a partial model in FORMULA](image)

### Table I: Execution time to automatically generate mappings

<table>
<thead>
<tr>
<th>Nb of tiles</th>
<th>Nb of terms</th>
<th>Execution time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>27</td>
<td>0.7</td>
</tr>
<tr>
<td>5</td>
<td>70</td>
<td>0.9</td>
</tr>
<tr>
<td>10</td>
<td>124</td>
<td>1.3</td>
</tr>
<tr>
<td>15</td>
<td>178</td>
<td>1.6</td>
</tr>
<tr>
<td>20</td>
<td>231</td>
<td>2.3</td>
</tr>
<tr>
<td>25</td>
<td>284</td>
<td>2.8</td>
</tr>
<tr>
<td>30</td>
<td>338</td>
<td>3.6</td>
</tr>
<tr>
<td>35</td>
<td>391</td>
<td>4.5</td>
</tr>
<tr>
<td>40</td>
<td>445</td>
<td>5.4</td>
</tr>
<tr>
<td>45</td>
<td>498</td>
<td>6.5</td>
</tr>
<tr>
<td>50</td>
<td>552</td>
<td>8.1</td>
</tr>
</tbody>
</table>

Notice that, it is possible to restrict the behavior of the system model by adding other constraints. For instance, a designer may reserve some of the processing units to behave as supervisors. This can be done by forbidding to map any of the software processes on these processing units. Moreover, it is possible to check if a given mapping is correct with respect to the defined constraints.

### V. CONCLUSION

The paper provides a formalism for the hardware architecture, the software model, and the mapping. The proposed formalism includes a set of constraints characterizing a correct design. We have implemented a domain language targeting the proposed formalism in the FORMULA framework. Moreover, we have presented a method to automatically derive a set correct designs (i.e., satisfying the defined constraints) from a given partial model. This work contrasts with existing approaches for HW/SW design which lack clear and detailed description of the system and in the use of the design state space exploration techniques. We have shown the efficiency and the scalability of finding correct models as illustrated by non-trivial systems.

Further developments will focus on integrating the notion of operating system(s), drivers, as well as other parameters. Another interesting problem is to develop other constraints for expressing other requirements (size of code, efficiency, \ldots). Another important line of research is to develop a set of partial models as libraries which define a set of standard architectures.

### ACKNOWLEDGMENT

This work has been mainly funded by the European Community through the Seventh Framework Program (FP7 EU-RETiles project under the Grant Agreement number 247846).

### REFERENCES