System Level Design Paradigms: Platform-Based Design and Communication Synthesis

ALESSANDRO PINTO, ALVISE BONIVENTO, and ALBERTO L. SANGIOVANNI-VINCENTELLI
University of California, Berkeley
ROBERTO PASSERONE
University of Trento
and
MARCO SGROI
DoCoMo Euro-Labs

Embedded system level design must be based on paradigms that make formal foundations and unification a cornerstone of their construction. Platform-Based designs and communication synthesis are important components of the paradigm shift we advocate.

Communication synthesis is a fundamental productivity tool in a design methodology where reuse is enforced. Communication design in a reuse methodology starts with a set of functional requirements and constraints on the interaction among components and then proceeds to build protocols, topology, and physical implementations that satisfy requirements and constraints while optimizing appropriate measures of efficiency of the implementation. Maximum efficiency can be reached when the communication specifications are entered at high levels of abstraction and the design process optimizes the implementation from this specification. Unfortunately, this process is very difficult if it is not cast in a rigorous framework. Platform-Based design helps define a successive refinement process where each step can be carried out automatically and optimized appropriately. We present two cases, an on-chip and a wireless sensor network design, where the resulting methodology gave encouraging results.

Categories and Subject Descriptors: J.6 [**Computer Applications**]: Computer-Aided Engineering

General Terms: Design, Theory

Additional Key Words and Phrases: Embedded systems, platform-based design, communication synthesis

Authors' addresses: A. Pinto, A. Bonivento, and A. L. Sangiovanni-Vincentelli, Department of Electrical Engineering and Computer Science, University of California at Berkeley, Berkeley, CA 94720; email: apinto@eecs.berkeley.edu; R. Passerone, Department of Information and Communication Technology, University of Trento, Universitá degli Studi di Trento via Belenzani, 12 38100 Trento, Italy; M. Sgroi, DoCoMo Euro-Labs, Landsbergerstr. 312, 80687 Munich, Germany.

1. INTRODUCTION

System design complexity is reaching levels that were unthinkable just a few years ago. Combined with the increased demands on time-to-market that are typical of consumer applications, it is creating the perfect storm in the electronic industry. Design teams are growing constantly, and errors that go into the market undetected increase in number and cost millions of dollars (not to mention the verification effort that has become even larger than the design effort). We believe that these problems can only be solved by developing paradigms based on formal foundations that can be applied at different levels of abstraction so as to provide a much needed unification in the design space.

In recent years, reuse has been touted as a possible cure to this malaise, whereby a design is the result of combining appropriately predesigned and preverified components. In this case, verification amounts to checking the correctness of the interconnection among components. The verification process is further aided if the composition can be formally proven correct. There has been a flurry of activity in the formal verification of interfaces that has clarified the main issues that need to be resolved [Rowson and Sangiovanni-Vincentelli 1997a; Passerone et al. 1998; de Alfaro and Henzinger 2001; Shimizu and Dill 2002; Chaki et al. 2002; Passerone et al. 2002; Chakrabarti et al. 2002, 2003]. One of the key findings is that if the components are designed with "clean" interfaces, then composing the components, that is, interconnecting them and establishing protocols that guarantee "correct" communication, can be done automatically. We call the automation of composing building blocks *communication synthesis*. Communication synthesis has been studied for years and among the first pioneering works, we list Yen and Wolf [1995], Ortega and Borriello [1998], and an interesting approach to the synthesis of communication topologies by Gasteier et al. [1998]. Recently, interest has shifted to on-chip networks, a very rich research area that we will analyze using our framework in Section 3.4.

As in any design problem, to perform communication design we need to first specify the functionality and constraints that must be satisfied. Then, the synthesis process consists of using a set of primitives available to the designer to implement the specification so that the constraints are satisfied and the functionality guaranteed. The higher the level of abstraction, the easier it is to express the functionality and constraints, as well as to catch design errors early. However, quickly reaching a high-quality implementation is more difficult due to the semantic gap between specification and implementation. Thus, researchers have either chosen to remain at high levels of abstraction and to optimize high-level structures, or to begin with a low level of abstraction that could reflect the characteristics of the implementation space. This is the case for early work in system level design, as in Prakash and Parker [1992], Gupta and Michelli [1993], and Ernst et al. [1993], where the specification and captured using a formal model is partitioned in hardware and software. The approach in Gajski et al. [1998] already proposed the idea of having two abstraction layers, but it was not cast into a formal framework. Another important work related to mapping applications to architecture while taking into account aspects such

as communication [Rhodes and Wolf 1999] and memory hierarchy [Li and Wolf 1998].

In this article, we present a successful design paradigm, platform-based design (PBD) [Ferrari and Sangiovanni-Vincentelli 1999; Chang et al. 1999; Vincentelli 2002], that can be formalized using a rigorous algebraic framework where we conjugate the ease of expressing and verifying the design of high levels of abstraction with the quality of low-level implementations.

As an example, assume we want to interconnect a set of nodes (e.g., computers) so that every node in the set can access every other node. Initial specifications may include the quality of service that each connection must be able to support, such as the required bandwidth and the maximum latency of the communication. We can solve this problem by constructing a network made of several different components such as routers, hubs, modems, protocol stacks, and links of different natures. The resources must be sized to satisfy the required constraints. However, the gap between our original high level specification and the implementation is obviously too large to be bridged in a single synthesis step: clearly, enumerating all possible topologies and interconnections is not practical, even for networks of modest complexity. A better way of approaching this problem is to divide this gap into several layers, where each layer focuses on a particular design choice. The question is then whether this division is optimal and, more importantly, how much of the entire design space can be explored. Answering these questions gives us an idea of the quality of the solutions that we obtain. Our approach consists of quantifying the design exploration process by relating the levels of abstraction corresponding to different layers. If two layers are too far apart, then the performance estimation will likely be poor and will not provide the necessary support for the synthesis algorithms.

In this context, a *platform* consists of a set of library elements, or resources, that can be assembled and interconnected according to predetermined rules to form a platform *instance*. One step in a platform-based design flow involves mapping a function or a specification onto different platform instances, and evaluating its performance. By employing existing components and interconnection resources, reuse in a platform-based design flow shifts the functional verification problem from the verification of the *individual elements* to the verification of their *interaction* [Rowson and Sangiovanni-Vincentelli 1997b; Sgroi et al. 2001]. In addition, by exporting an abstracted view of the parameters of the model, the user of a platform is able to estimate the relevant performance metrics and verify that they satisfy the design constraints. The mapping and estimation step is then repeated at increasingly lower levels of abstraction in order to come to a complete implementation. These principles are embodied in the Metropolis project, a software infrastructure and a design methodology for heterogeneous embedded systems that supports platformbased design by exploiting refinement through different levels of abstraction [Balarin et al. 2002]. To make the methodology effective, the particular abstraction layers must be tuned to each application area.

The ability to express different levels of abstraction within the same infrastructure is important in creating a central repository where all models can be

stored and compared. This is especially convenient in a platform-based design methodology where each step consists of the fusion of two different views, one emphasizing the function to be performed, and the other the architecture that supports the computation. These views are brought together in the context of a third abstraction level, one that emphasizes the relations between the other two. Crossing the boundaries between abstraction levels, that is, the process of abstracting or refining a specification, is often nontrivial. The most common pitfalls include mishandling corner cases and inadvertently misinterpreting changes in the communication semantics.

These problems arise because of poor understanding and the lack of precise definitions of the abstraction and refinement maps used in the flow. In addition, abstraction and refinement should be designed to preserve, whenever possible, the properties of the design that have already been established. This is essential to increase the value of early high-level models and to guarantee a speedier path to implementation.

The article is organized as follows: First, we approach the problem of abstraction and refinement of PBD from a formal standpoint, and then we apply the principles of architecture exploration in the context of our formalism to the areas of interest—the synthesis and optimization of wired and wireless communication networks.

2. FORMALIZING PLATFORM-BASED DESIGN

Our formalization of the platform-based design methodology is based on the framework of agent algebra [Passerone 2004]. Informally, an agent algebra $\mathcal{Q}$ is composed of a domain D that contains the agents under study for the algebra, and of certain operators that formalize the most common operations of the models of computation used in embedded system design. Different models of computation are constructed by providing different definitions for the domain of agents and the operators. The algebra also includes a master alphabet $\mathcal{A}$ that is used as the universe of "signals" that agents use to communicate with other agents.

Definition 2.1. An agent algebra $\mathcal{Q}$ has a domain $\mathcal{Q}.D$ of agents, a master alphabet $\mathcal{Q}.\mathcal{A}$, and three operators: renaming, projection, and parallel composition, denoted by $\text{rename}(r)$, $\text{proj}(B)$, and $\|$. Each agent $p \in \mathcal{Q}.D$ is associated with an alphabet $A \subseteq \mathcal{A}$.

The operators of the algebra are partial functions on the domain D and have an intuitive correspondence with those of most models of concurrent systems. The operation of renaming, which takes as argument a renaming function r on the alphabet, corresponds to the instantiation of an agent in a system. Projection corresponds to hiding a set of signals, and takes the set B of signals to be retained as a parameter. Hence, it corresponds to an operation of scoping. Finally, parallel composition corresponds to the concurrent "execution" of two agents. It is possible to define other operators. We prefer to work with a limited set and add operators only when they cannot be derived from existing ones. In particular, in this work we will be mainly concerned with the operator of

parallel composition. The operators must satisfy certain axioms that formalize their intuitive behavior and provide some general properties that we want to be true, regardless of the model of computation. For example, parallel composition must be associative and commutative. The definition of the operators is otherwise unspecified, and depends on the particular agent model being considered.

The notion of refinement in each model of computation is represented by adding a preorder (or a partial order) on the agents, denoted by the symbol $\preceq$. The result is called an *ordered agent algebra*. We require that the operators in an ordered agent algebra be monotonic relative to the ordering. This is essential to apply compositional techniques. However, since these are partial functions, this requires generalizing monotonicity to partial functions. This generalization is, however, beyond the scope of this article. The interested reader is referred to Passerone [2004] for more details.

It is easy to construct an agent algebra $\mathcal{Q}$ to represent the interface that components expose to their environment. In this case, the set D consists of agents of the form $p = (I, O)$, where $I \subseteq \mathcal{Q}.\mathcal{A}$ is the set of input ports of the components and $O \subseteq \mathcal{Q}.\mathcal{A}$ is the set of output ports. The alphabet of an agent p is simply $A = I \cup O$, and we require that the set of inputs and outputs be disjoint, that is, $I \cap O = \emptyset$. The parallel composition $p = p_1 \parallel p_2$ is defined only if the sets O_1 and O_2 are disjoint, to ensure that only one agent drives each port. When defined, a port is an output of the parallel composition if it is an output of either agent. Conversely, it is an input if it is an input of either p_1 or p_2 and it is not concurrently an output of the other agent. Thus, $O = O_1 \cup O_2$ and $I = (I_1 \cup I_2) - (O_1 \cup O_2)$. Given the definitions, it is clear that in this example connections are established by name.

The model can be enriched with information about the nature of the signals used by the agents. For instance, in the case of agents that describe communication topologies, signals can be distinguished between those that belong to a link, denoted by the symbol l, and those that belong to a component, denoted by the symbol n (nonlink). We call this a *typed IO agent algebra*. The sets I and O of an agent p thus become sets of pairs of signals together with their type, that is, $I \subseteq \{(a, t) : a \in \mathcal{Q}.\mathcal{A} \wedge t \in \{l, n\}\}$, and similarly for the output ports. Parallel composition can also be modified so that the operation is defined only if the ports of the agents being connected are not of the same type, that is, a link must be used to connect two regular ports. Hence, $p_1 \parallel p_2$ is defined if and only if for all $i \in I_1$ and for all $o \in O_2$, if $i.a = o'.a$, then $i.t \neq o'.t$, and vice versa for p_2 and p_1.

With these definitions, it is in general not possible to derive the components from the composite. Later, we will see how this can be accomplished for a different model that we use to define architectures. There, we will also introduce nontrivial orderings of the agents.

We relate different agent algebras by means of conservative approximations. A conservative approximation from $\mathcal{Q}$ to $\mathcal{Q}'$ is a pair $\Psi = (\Psi_l, \Psi_u)$, where Ψ_l and Ψ_u are functions from $\mathcal{Q}.D$ to $\mathcal{Q}'.D$. The first mapping is an upper bound of the agent relative to the order of the algebra: For instance, the abstract agent represents all of the possible behaviors of the agent in the more detailed domain, plus possibly some more. The second mapping is a lower bound: The abstract

agent represents only possible behaviors of the more detailed one, but possibly not all. Formally, a conservative approximation is an abstraction that maintains a precise relationship between the orders in the two agent algebras.

Definition 2.2. Let $\mathcal{Q}$ and $\mathcal{Q}'$ be ordered agent algebras, and let Ψ_l and Ψ_u be functions from $\mathcal{Q}.D$ to $\mathcal{Q}'.D$. We say that $\Psi = (\Psi_l, \Psi_u)$ is a conservative approximation from $\mathcal{Q}$ to $\mathcal{Q}'$ if and only if for all agents p and q in $\mathcal{Q}.D$,

$$\Psi_u(p) \preceq \Psi_l(q) \Rightarrow p \preceq q.$$

Thus, when used in combination, the two mappings allow us to relate refinement verification results in the abstract domain to results in the more detailed domain. Hence, the verification can be done in $\mathcal{Q}'$, where it is presumably more efficient than in $\mathcal{Q}$. The conservative approximation guarantees that this will not lead to a false positive result, although false negatives are possible depending on how the approximation is chosen.

To define the inverse Ψ_{inv} of an approximation, we investigate whether there are agents in $\mathcal{Q}.D$ that are represented exactly by Ψ_u and Ψ_l, rather than just being bounded. We do so by only considering those agents p for which $\Psi_l(p)$ and $\Psi_u(p)$ have the same value p'. Intuitively, p' represents p exactly in this case, and we therefore define $\Psi_{inv}(p') = p$. If $\Psi_l(p) \neq \Psi_u(p)$, then p is not represented exactly in $\mathcal{Q}'$. In this case, p is not in the image of Ψ_{inv}.

Definition 2.3. Let $\Psi = (\Psi_l, \Psi_u)$ be a conservative approximation from $\mathcal{Q}$ to $\mathcal{Q}'$. For $p' \in \mathcal{Q}'.D$, the inverse $\Psi_{inv}(p')$ is defined and is equal to p if and only if $\Psi_l(p) = \Psi_u(p) = p'$.

If the algebra $\mathcal{Q}$ is partially ordered (as opposed to preordered), the inverse of the conservative approximation is uniquely determined. Otherwise, a choice may be possible among order-equivalent agents. In all cases, however, because of the defining properties of a conservative approximation, Ψ_{inv} is one-to-one, monotonic, and inverse of both Ψ_l and Ψ_u.

Assume now that for an agent p, $\Psi_{inv}(\Psi_l(p))$ and $\Psi_{inv}(\Psi_u(p))$ are both defined. It is easy to show that $\Psi_{inv}(\Psi_l(p)) \preceq p \preceq \Psi_{inv}(\Psi_u(p))$. This fact makes precise the intuition that $\Psi_l(p)$ and $\Psi_u(p)$ represent a lower and an upper bound of p, respectively.

We can use agent algebras to describe formally the process of successive refinement in a platform-based design methodology. There, refinement is interpreted as the concretization of a *function* in terms of the elements of a platform. The process of design consists of evaluating the performance of different kinds of instances in the platform by mapping the functionality onto its different elements. The implementation is then chosen on the basis of a cost function. We use three distinct domains of agents to characterize the process of mapping and performance evaluation. The first two are used to represent the platform and the function, while the third, called the *common semantic domain*, is an intermediate domain that is used to map the function onto a platform instance.

A platform, depicted on the right in Figure 1, corresponds to the implementation search space.

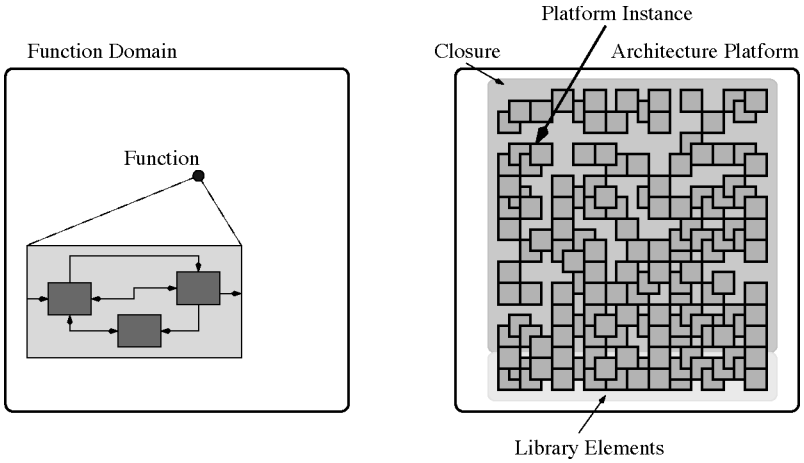

Fig. 1. Architecture and function platforms.

Definition 2.4. A platform consists of a set of elements, called the *library elements*, and of *composition rules* that define their admissible topologies of interconnection.

To obtain an appropriate domain of agents to model a platform, we start from the set of library elements D_0. The domain of agents D is then constructed as the closure of D_0 under the operation of parallel composition. In other words, we construct all the topologies that are admissible by the composition rules, and add them to the set of agents in the algebra. Each element of the architecture platform is called a platform instance.

Performance evaluation usually requires that the elements of a platform include information regarding their internal structure. Thus, an algebra, such as the typed IO agent algebra described, is not suitable for this purpose since the composition does not retain the structure of the agent. The IO agents can, however, be used as library elements D_0. A new domain of agents D can then be constructed as follows. If $p_0 \in D_0$ is a library element, we include the *symbol* $\mathbf{p_0}$ in the set of agents $\mathcal{Q}.D$. We then close the set D under the operation of parallel composition. However, we represent a composition $p = p_1 \parallel p_2$ in $\mathcal{Q}$ as the *sequence* of symbols $\mathbf{p_1} \parallel \mathbf{p_2}$. By doing so, we retain the *structure* of the composite, since all the previous composition steps are recorded in the representation. We call this process a platform *closure*.

Definition 2.5. Given a set of library elements D_0 and a composition operator $\parallel$, the *platform closure* is the algebra with domain

$$D = \{\mathbf{p} : p \in D_0\} \cup \{\mathbf{p_1} \parallel \mathbf{p_2} : \mathbf{p_1} \in D \wedge \mathbf{p_2} \in D\} \tag{1}$$

where $p_1 \parallel p_2$ is defined if and only if it can be obtained as a legal composition of the agents in D_0.

The construction we have just outlined is general and can be applied to building several different platforms, as will be shown later. The result is similar

to a term algebra with the "constants" in D_0 and the operation of composition. Unlike a term algebra, however, our composition is subject to the constraints of composition rules. For example an "architecture" platform may provide only one instance of a particular processor. In that case, topologies that use two or more instances are ruled out. In addition, the final algebra must be taken up to the equivalence induced by the required properties of the operators. For example, since parallel composition must be commutative, $\mathbf{p_1} \parallel \mathbf{p_2}$ should not be distinguished from $\mathbf{p_2} \parallel \mathbf{p_1}$. This can be accomplished by taking the appropriate quotient relative to the equivalence relation. The details are outside the scope of this article.

On the other hand, the function, depicted in Figure 1 on the left, is represented in an agent algebra called the *specification domain*. Here, the desired function may be represented denotationally as the collective behavior of a composition of agents, or it may retain its structure in terms of a particular topology of simpler functions. The denotational representation is typically used at the beginning of the platform-based design process, when no information on the structure of the implementation is available. Conversely, after the first mapping, the subsequent refinement steps are started from the mapped platform instance, which is taken as the specification. Thus, a common semantic domain, described in the following, is used as the specification domain. However, contrary to the mapping process that is used to select one particular instance among several, when viewed as a representation of a function, the mapped instance is a specification and is therefore fixed.

The function and the platform come together in an intermediate representation called the common semantic domain. This domain plays the role of a common refinement and is used to combine the properties of both the platform and the specification domain that are relevant to the mapping process. The domains are related through conservative approximations.

Definition 2.6. Given a platform $\mathcal{Q}^P$ and specification domain $\mathcal{Q}^S$, a *common semantic domain* is an agent algebra $\mathcal{Q}^C$ related to $\mathcal{Q}^P$ and $\mathcal{Q}^S$ through conservative approximations Ψ^P and Ψ^S, respectively.

In particular, we assume that the inverse of the conservative approximation is defined at the function that we wish to evaluate. The function is therefore mapped onto the common semantic domain as shown in Figure 2. This mapping also includes all the refinements of the function that are consistent with performance constraints, which can be interpreted in the semantic domain.

If the platform includes programmable elements, the correspondence between the platform and the common semantic domain is typically more complex. In this case, each platform instance may be used to implement a variety of functions or behaviors. Each of these functions is in turn represented as one agent in the common semantic domain. A platform instance is therefore projected onto the common semantic domain by considering the collection of agents that can be implemented by the particular instance. This projection, represented by the rays that originate from the platform in Figure 2, may or may not have a greatest element. If it does, the greatest element represents the nondeterministic choice of any of the functions that are implementable by the instance.

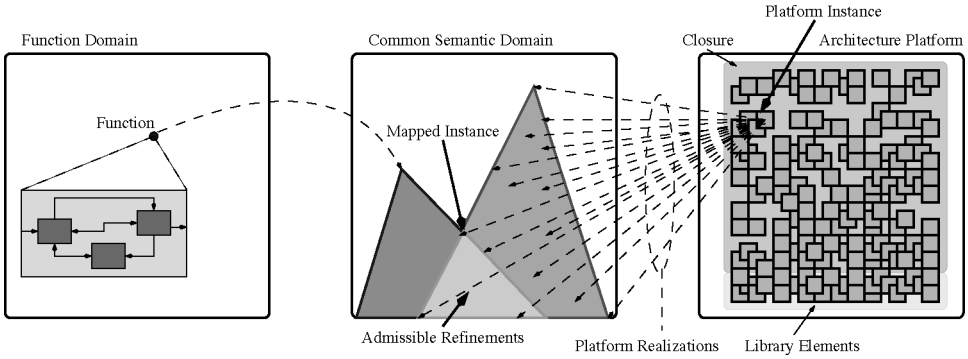

Fig. 2. Mapping of function and architecture.

The common semantic domain is partitioned into four different areas. We are interested in the intersection between the refinements of the function and the functions that are implementable by the platform instance. This area is marked "Admissible Refinements" in Figure 2. Each of the admissible refinements encodes a particular mapping of the components of the function onto the services offered by the selected platform instance. These can often be seen as the *covering* of the function through the elements of the platform library. Of all these agents, those that are closer to the greatest element are more likely offer the most flexibility in the implementation. Once a suitable implementation has been chosen (possibly by considering different platform instances), the same refinement process is iterated to descend to an even more concrete level of abstraction. The new function is thus the intersection between the behavior of the original function and the structure imposed by the platform. The process continues recursively at increasingly detailed levels of abstraction to reach the final implementation.

3. ON-CHIP COMMUNICATION SYNTHESIS

In this section, we address the problem of interconnecting predesigned cores on a chip, a problem that we will use as one of the two application areas where the power of our proposed approach is demonstrated.

The number of intellectual property (IP) components placed on a single chip (or in a single package) has increased to a point that the traditional bus-like interconnect structures are not offering a satisfactory "service" to communication. This, together with the increasing latency of global wires with respect to gate delays, suggests the use of on-chip interconnection networks [Dally and Towles 2001]. In these networks, the position of IPs routers as well as and the links that interconnect them is referred to as the *network topology*. Given a topology, the exchange of information is regulated by a *protocol* that arbitrates access to shared media and decides the routing of information flows.

In this application, the topology optimization problem becomes challenging because of the heterogeneity of the cores, the nonuniformity of interconnection requirements, and the fact that network agents can be freely placed on the chip with the only constraint being that the total chip area remains fixed. Given

the tight constraints in terms of throughput and energy consumption, topology optimization plays an important role in chip design.

The standard design flow for on-chip networks starts with the application, an MPEG decoder, for instance. The first step is partitioning the application into subblocks which are allocated to processing elements. The result of this step is the extraction of a set of communication constraints, such as bandwidth and latency, among processing elements. Given the constraints, an optimal communication topology is selected and then passed to the last step of the flow that generates a hardware implementation of the network (this step is known as a network compilation). Topology selection uses a library of components, such as wires and on-chip routers, that is characterized by abstracting the available silicon implementation.

3.1 A Brief Overview of On-Chip Network Design

The network compilation step has been addressed well by the Xpipes [Bertozzi and Benini 2004] library and compiler. For simulation, the work of Lahiri et al. [2004] has provided a fast environment for the simulation of on-chip communication architectures based on busses. Orion [Wang et al. 2002], a detailed power simulator for networks on-chip, allowed us to evaluate the power/performance tradeoff for routers and on-chip networks [Wang et al. 2005]. Topology selection has been much more researched, as it has more impact on the final quality of the network. Topology selection has been formulated as a synthesis problem of a point-to-point network in Hu et al. [2002], and as a collection of two-node subnetworks in Pinto et al. [2002]. The benefits and challenges of networks on-chip were fully described in Benini and Micheli [2002], and after having realized the lack of theoretical frameworks for dealing with arbitrary topologies, regular structures such as meshes and toruses were adopted as communication fabric where each computing element sits in the tiles of the mesh. In Hu and Marculescu [2003, 2004], the problems of assigning cores to the tiles of a mesh network and of generating a routing protocol that minimizes the communication cost were explored. In this approach, the library of topologies is limited to meshes. A similar approach was taken by researchers regarding SUNMAP [Murali and Micheli 2004], an algorithm for automatic mapping and topology selection. Here, the library consists of several regular topologies such as meshes, toruses, and butterflies. SUNMAP is part of the NetChip project [Bertozzi et al. 2005], which aims at providing the entire flow for the design and optimization of networks on-chip. Other relevant works in the area of topology selection are more synthesis-oriented.

The general topology synthesis problem was not entirely abandoned: In Pinto et al. [2003], the authors proposed a two-step algorithm that first clusters the communication constraints and then synthesizes each cluster as a two-node network. In Srinivasan et al. [2004], the clustering step is performed by solving a mixed-integer linear program that can lead to more complicated topology than the previously mentioned one. However, the cost of links was not considered. Finally, OIDIPUS [Ahonen et al. 2004] optimizes the relative positions of processing elements to minimize communication cost.

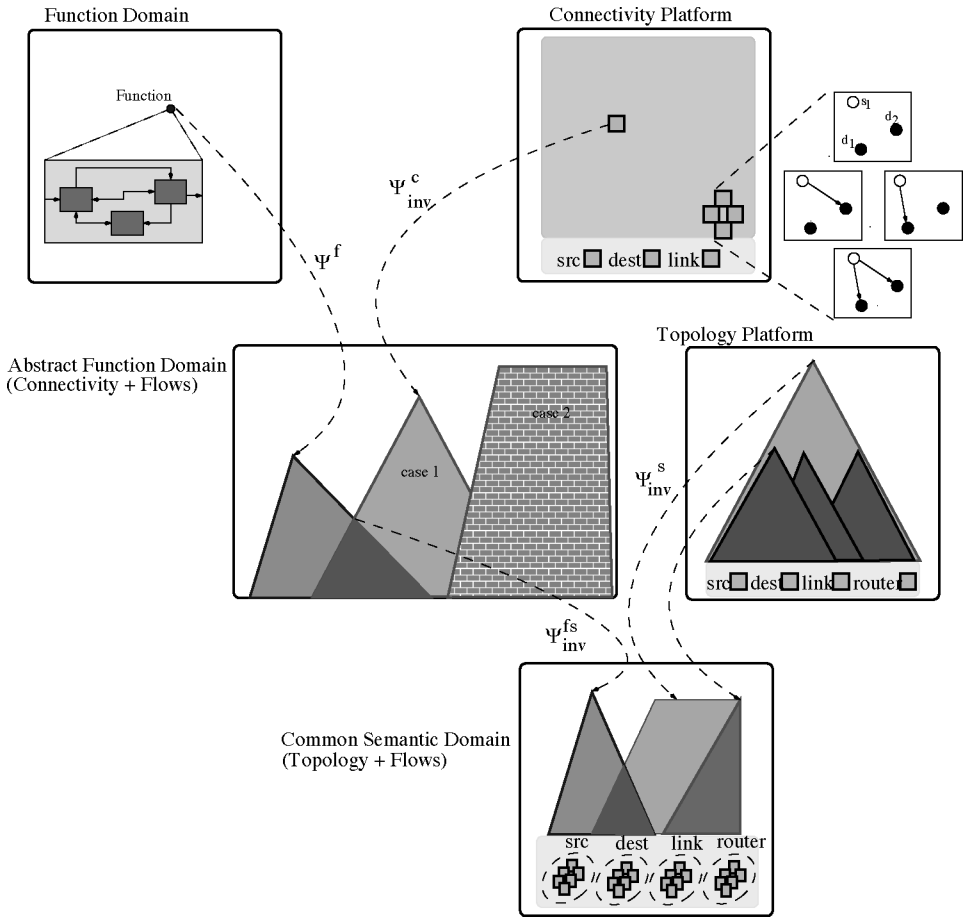

Fig. 3. Platform-based design flow for communication synthesis.

Many practical solutions have been proposed and adopted in industry. Most of them are implementations and standardizations of bus architectures like the advanced microprocessor bus architecture (AMBA) system[1] by ARM and the core-connect architecture[2] by IBM.

A few companies specialize only on the communication infrastructure, like Sonics[3] and recently, Arteris,[4] specifically for on-chip networks.

3.2 Constraint-Driven Communication Synthesis

Figure 3 shows the communication synthesis design flow. The domain for each platform is obtained following the construction of Definition 2.5. The set of

[1]http://www.arm.com/products/solutions/AMBA_Spec.html.
[2]http://www-03.ibm.com/chips/products/coreconnect.
[3]http://www.sonicsinc.com.
[4]http://www.arteris.net.

library elements consists in all cases of appropriate subsets of the typed IO agent algebra. At the highest level of abstraction, the *connectivity* platform $\mathcal{Q}^c$ is only concerned with point-to-point connections between sources and destinations. The library elements of $\mathcal{Q}^c$ are of three types: the set of *sources* $\mathcal{S} = \{(I, O) : I = \emptyset\}$, the set of *destinations* $\mathcal{D} = \{(I, O) : O = \emptyset\}$, and the set of *point-to-point links* $\mathcal{L} = \{(I, O) : |I| = |O| = 1\}$. Furthermore, the input and output ports of sources and destinations must all have type n, while ports that belong to links have type l. Hence, given the rules of composition, it is not possible to connect sources to destinations directly. Architecture templates in the connectivity platform are simply point-to-point connections amongst a set of source agents and a set of destination agents.

A preorder in this algebra can be defined by considering substitutability. In general, an architecture that offers more connections can be substituted for another that offers fewer connections. Hence, we define $p_1 \preceq p_2$ if and only if p_1 and p_2 have the same set of sources and destinations, and for each link between a source-destination pair in p_2, there is a corresponding link between the same source-destination pair in p_1. To illustrate the order, consider the simple case shown on the right in Figure 3. There, one source, s_1, is connected to two destinations, d_1 and d_2. The most refined architecture instance includes all links of s_1 to d_1 and d_2. Intermediate architectures include only one link to either d_1 or d_2. The greatest element is finally the architecture with no links. The connectivity platform could be used to evaluate the impact of connectivity on the performance of a system.

As described in Section 2, the function to be implemented on the architecture is represented in some suitable domain. This domain depends on the application [Balarin et al. 2002]. For instance, multimedia applications are usually described using nondeterministic Khan process networks (KPN). For the purpose of mapping, the function is abstracted using a conservative approximation Ψ^f from the function domain to a common semantic domain (which we call an *abstract function domain*) described by a typed IO agent algebra $\mathcal{Q}^{cf}$ that includes multicommodity flow information.

A commodity is a pair of elements (c_n, c_v), where c_n is the type of the commodity and $c_v \in \mathbb{R}^+$ is the commodity value (we only consider this simplified description of the communication requirements to keep the exposition simple, but other constraints could be take into account, such as latency and statistical properties). A port of an agent in the abstract function domain includes a commodity, in addition to its type. A parallel composition is defined only if the link connected to a port carries the same commodity with a higher value. An order for this model can be defined by considering the connectivity (for the connectivity platform) and the multicommodity flow containment (flows have the order induced by the reals). The approximation Ψ^f maps the input and output ports of a function to abstract source-destination pairs and the communication channels to links. The conservative approximation also assigns commodities to ports that are estimates of the bandwidth required by the communication. Since the abstract model has no information about behavior, none of the processes can be represented exactly and the inverse of the conservative approximation is not defined.

The inverse of the conservative approximation Ψ^c_{inv} maps a connectivity platform instance in the abstract function domain. If there exists an instance with the required connectivity (case 1 in Figure 3), then it is possible to find a set of admissible refinements in the common semantic domain. It might happen, though, that the intersection between the function instance concretization and the platform instance concretization is empty. This is the case, for example, when there is a constraint on the maximum number of links in the connectivity platform. Among all agents in the set of admissible refinements, the greatest element is the one having the minimum number of connections, each carrying the minimum commodity such that the function instance constraints are still satisfied. This agent, which we call a connectivity instance, is selected as the function instance for the next level of abstraction. This choice is made by considering that connections and commodity values are constraints that must be satisfied in some common semantic domain at lower levels of abstraction. This agent is the less constraining agent and is therefore a good candidate for cheap implementations.

Platforms used in communication synthesis, however, often include more complex topologies. To model this situation we build the *topology* platform $\mathcal{Q}^t$, which uses the same elements of the connectivity platform with the addition of a library component called a *router*. The set of routers is formally defined as $\mathcal{R} = \{(I, O) : (|I| \geq 1 \wedge |O| > 1) \vee (|O| \geq 1 \wedge |I| > 1)\}$. Notice that we are not yet considering simple one-input one-output FIFOs. The ports of a router are required to be of type n, so that links must be used to connect the routers to other elements of the platform. The routers allow us to construct all the well known topologies, like rings, crossbars, stars, and busses.

The ordering of agents is defined by the underlying connectivity and the number of hops on each source-destination path the number of routers among all paths from source to destination, in particular, $p_1 \preceq p_2$ if and only if p_1 connects more source-destination pairs than p_2, with fewer or as many hops. We can establish a relationship between the topology and the connectivity platform by a conservative approximation Ψ^t. The upper bound ignores the routers by constructing the underlying connectivity, while the lower bound is obtained by considering only the existing point-to-point links between sources and destinations. Thus, the inverse of the approximation Ψ^t_{inv} maps the connectivity instance to the corresponding fully connected topology.

A mapping between the connectivity instance and communication topology is realized in a common semantic domain $\mathcal{Q}^{sp}$ which contains both topology and multicommodity flow information. Point-to-point connections in $\mathcal{Q}^{cf}$ become source-destination paths in $\mathcal{Q}^{sp}$.

The algebra $\mathcal{Q}^{sp}$ is similar to $\mathcal{Q}^{cf}$, with the addition of routers as library components. The library of this platform consists of sets of elements, one for each of the element types: sources, destinations, links, and routers. For instance, there are several types of links depending on their commodities.

The common semantic domain $\mathcal{Q}^{sp}$ is related to the abstract function domain $\mathcal{Q}^{cf}$ and the topology platform $\mathcal{Q}^t$, through the appropriate conservative approximations Ψ^{fs} and Ψ^s, respectively. In either case, the approximation ignores information not contained in the respective abstract algebra. The lower

bound defines, as usual, the conditions under which the representation is exact. For example, a fully connected topology in $\mathcal{Q}^{sp}$ can be represented exactly in $\mathcal{Q}^{cf}$. Similarly, any topology with zero flows is represented exactly in $\mathcal{Q}^{t}$. The inverses Ψ^{fs}_{inv} and Ψ^{s}_{inv} of such approximations thus establish correspondences between the elements of $\mathcal{Q}^{cf}$ and $\mathcal{Q}^{sp}$, and between $\mathcal{Q}^{t}$ and $\mathcal{Q}^{sp}$. In particular, a topology in $\mathcal{Q}^{t}$ is refined in an ordered set of topologies by Ψ^{s}_{inv}, the greatest element being the topology with commodities all equal to zero. Similarly, a multicommodity flow connectivity graph in $\mathcal{Q}^{cf}$ is mapped by Ψ^{fs}_{inv} to all the possible topologies and their flows that satisfy the connectivity and multicommodity flow constraints imposed by the connectivity instance. These mappings are represented by the triangles in Figure 3 which denote all the refinements, in addition to the mapped element. The intersection of these sets represents all the possible networks that satisfy the connectivity and multicommodity flow requirement. It is possible that a specific topology maps to a set which does not intersect the concretization of the connectivity instance in the common semantic domain. In that case, the topology is ruled out from the search space.

3.3 Synthesis of Network Topology

The platform-based design of communication topology is implemented in COSI, the "communication synthesis infrastructure" under development at the University of California at Berkeley. COSI accepts as input a connectivity instance, described as a set of source-destination pairs, with an associated commodity that can be a general quantity defined by the user who has to specify both the commodity domain and a total order on it. The description format forces the connectivity instance to satisfy the composition rules of $\mathcal{Q}^{cf}$. For instance, it is not possible to specify a direct connection between a source and a destination, but a link will be instantiated to connect them. COSI also accepts the set of library components of the common semantic domain $\mathcal{Q}^{sp}.D_0$ as input. The data structure is able to represent agents in $\mathcal{Q}^{sp}.D$. A position can also be assigned to each agent instance and a cost can be assigned to each library element, leading to a more refined algebra. The first scenario is concerned with the automatic topology selection of a communication network satisfying connectivity and multicommodity flow constraints. Figure 4 shows one example of how the synthesis process moves into the search space. Since the topology synthesis problem is, in general, NP-hard [Garey and Johnson 1979], we build an optimal algorithm and use the general framework to select a good initial solution and a strategy to search in the solution space.

We build a *complete topology* using the following construction: For each source and for each destination, we add a router and a connection link connecting them; for all ordered pairs of routers we add a link. The resulting topology is shown in Figure 4: It is a two-level network where the outer network is represented by the set of sources and destination instances while the inner network is a complete topology on the routers. Agent **p** is fairly at the bottom in the preorder because it has full connectivity and a minimum of two hops for every source-destination pair. The values of the commodities supported by each component can either be the maximum available in the library, or the

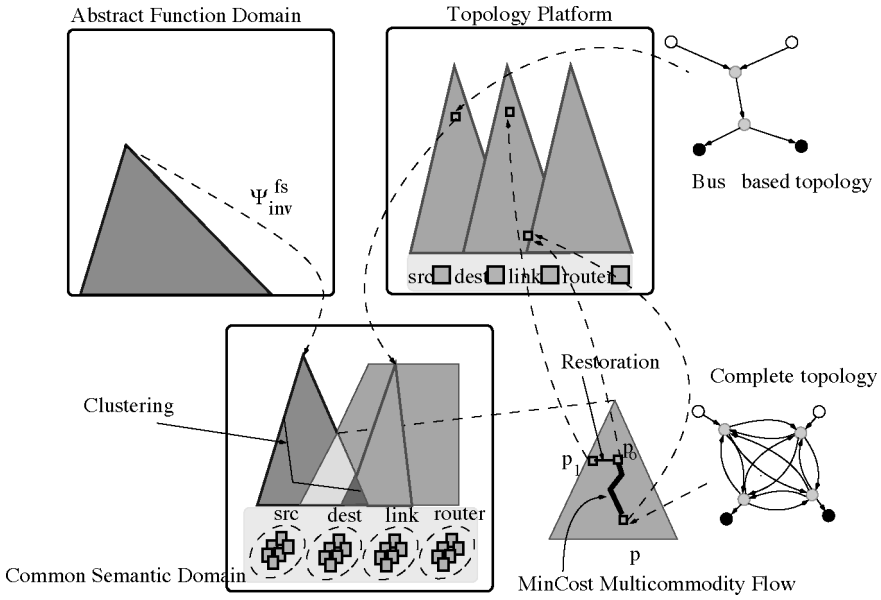

Fig. 4. Abstraction and refinement in communication topology selection.

minimum such that constraints imposed by the connectivity instance are still satisfied.

We can gauge whether an agent $p \in \mathcal{Q}^{sp}$ satisfies the constraints imposed by the connectivity instance $p_f \in \mathcal{Q}^{cf}$ by checking that $\Psi_l^{fs}(p) \preceq p_f$. The conservative approximation Ψ_l^{fs} is the all pair max multicommodity flow which given a network topology, computes the maximum multicommodity flow for each source-destination pair.

The complete topology $\mathbf{p}$ is a good starting point for the synthesis algorithm because of its generality. Abstracting $\mathbf{p}$ in the topology platform gives an agent which is also at the bottom of the order due to its high connectivity and low number of hops. This means that it can substitute many other known topologies and hence, is a very general communication architecture. The triangles in the topology platform of Figure 4 represent different topology templates, like rings, stars etc., that can all be substituted by $\Psi_l^s(\mathbf{p})$.

While complete topology is a good starting point for the synthesis algorithm, choosing an agent low in the order means propagating more restrictive constraints to the lower level of abstraction (high connectivity and low number of hops, in this case). The first step is to decrease the cost of the complete topology by decreasing the commodities value as much as possible. A mincost multicommodity flow algorithm [Wynants 2001] is used to decrease the commodities value while satisfying the constraints imposed by the connectivity instance. The mincost multicommodity flow will optimize flows in the original topology, obtaining another agent $\mathbf{p_o}$. Since the topology, platform has no information about commodities, we note that $\Psi_l^s(\mathbf{p}) = \Psi_l^s(\mathbf{p_o})$.

The second step is the topology selection. Given $\mathbf{p_o}$, it is possible to generate different topologies by removing edges in the complete graph. When an edge

is removed, the commodity it carries must be redistributed in the network. This algorithm is known as the *restoration* algorithm [Wynants 2001], which can be used to generate known topologies like rings, stars, etc. If we abstract the topology generated by the restoration algorithm, it gives an agent which is higher in the order than the abstraction of the complete topology.

In recent implementations, COSI also solves the joint optimization of node position and edge flow to minimize total energy consumption. The optimization problem can be stated as follows:

$$\begin{aligned} &\min C(\mathbf{p}) \\ &s.t.\ \Psi_l^{fs}(\mathbf{p}) \preceq \Psi^f(\mathbf{f}), \end{aligned}$$

where $C(\mathbf{p})$ is the cost of the selected full topology and $\mathbf{f}$ is the application. Let $\mathbf{q} \in_{||} \mathbf{p}$ denote that agent $\mathbf{q}$ appears in the symbol $\mathbf{p}$. The cost of a network is computed as:

$$C(\mathbf{p}) = \sum_{\mathbf{l} \in_{||} \mathbf{p} \wedge \mathbf{l} \in \mathcal{L}} C(\mathbf{l}) + \lambda \sum_{\mathbf{r} \in_{||} \mathbf{p} \wedge \mathbf{r} \in \mathcal{R}} C(\mathbf{r}),$$

where the cost of a link depends on its length and flow and the cost of a router depends on the cumulative input flow. The parameter λ measures the relative cost of computation versus communication that corresponds to the relative cost of storing data in the router's queues versus communicating the data on a wire of unit length. The key point is to explore the largest possible design space. The complete topology has this goal since every other topology can be obtained starting from it.

COSI (`http://embedded.eecs.berkeley.edu/cosi`) is an open infrastructure that allows an easy method to plug in other algorithms for communication synthesis (not limited to topology exploration only). Recent developments have improved the previous approach. In particular, complicated platform constraints can be included in the library. The increasing importance of latency asks for a careful abstraction of the communication link with respect to the maximum distance that a link can span. The critical sequential length l_{st} is the length that can be spanned within a clock cycle, after which a stateful (or clocked) repeater must be inserted. Stateful repeaters are single-input single-output buffers and, together with the routers' internal queues, account for a large amount of the on-chip network power consumption.

Another constraint to take into account is the available area to install routers and clocked repeaters. We assume that the available space is another input of the problem that limits the topology design space. As we have already said, point-to-point connections in $\mathcal{Q}^{cf}$ become source-destination paths in $\mathcal{Q}^{sp}$, therefore, the problem is to build a graph that encodes all the topology instances and then to select a set of source-destination paths to route all constraints while minimizing the cost function. Let $G = (V, E)$ be such a graph and let $\mathcal{P}$ be the set of all admissible positions for routers/repeaters. A vertex v belongs to V if and only if its position is in $\mathcal{P}$. The set of sources and destinations also belongs to the set V. Given two vertexes $u, v \in V$, the edge (u, v) belongs to E if and only if its length is shorter that the critical sequential length.

A set of algorithms can now be executed on graph G to remove edges and routers that are not contained in the final optimal topology. A set of well studied problems can be used to find an optimal solution. Currently, two options are available to the designer. A minimum cost routing based on the buy-at-buy networks design algorithm described in Charikar and Karagiozova [2005] can be directly executed to obtain an optimal network design. The synthesis can be divided into a network interface allocation phase and an optimal routing between interfaces. Network interfaces are expensive resources because they have to perform tasks like protocol adaptation, and the designer may want to limit the number of interfaces in the design. We solve the network interface allocation problem by casting it into an instance of the k-median problem, where available routers are considered facilities and sources, and destinations are considered cities. We use the primal-dual algorithm in Vazirani [2003]. Once the interfaces are installed, the source-destination constraints are translated into interface-to-interface constraints and the optimal routing is solved using the buy-at-bulk network design algorithm.

3.4 Comparison with Other Approaches

In Hu et al. [2002], the authors consider point-to-point networks. This means considering a platform that only contains sources, destinations and links. In this framework, the exploration is done in the abstract function domain $\mathcal{Q}^{cf}$ (in Figure 3) and does not consider the successive platform $\mathcal{Q}^{sp}$ where more complicated topologies (that could possibly be more cost efficient than a point-to-point implementation) can be explored. The design space exploration is done in the more refined domain where sources and destinations have a position associated with them. Let $\mathcal{Q}^{cfp}$ denote such a domain. The algorithm minimizes energy consumption and wire length by considering positions as optimization variables. Given $\mathbf{p}, \mathbf{q} \in \mathcal{Q}^{cfp}.D$, $\mathbf{p} \preceq \mathbf{q}$ if they have the same connectivity and the sum of distances of all links in $\mathbf{p}$ is greater than or equal to the sum in $\mathbf{q}$. The algorithm searches for the agent that is higher in the order. Such an agent will be less constraining for the successive step of routing.

In Hu and Marculescu [2003, 2004] and Murali and Micheli [2004], the exploration is done in $\mathcal{Q}^{sp}$. The method by which the exploration is carried out is by fixing the network topology to a specific one, like a mesh or a torus, and optimizing communication on such a structure. The first step that the algorithm performs is to map the functional units on the cores of the platform. This step corresponds to the selection of an agent $\mathbf{f} \in \mathcal{Q}^{cf}$. The second step is to select a topology template $\mathbf{q} \in \mathcal{Q}^{t}$ like a mesh, torus, butterfly, etc. The optimization algorithm selects an agent $\mathbf{p} \in \mathcal{Q}^{sp}$ that minimizes a cost function (typically, power consumption) and such that $\Psi_l^{f^s}(p) \preceq \mathbf{f} \wedge \mathbf{q} \preceq \Psi_u^s(\mathbf{p})$. This procedure is repeated for all possible $\mathbf{f}$ (i.e., for all mappings of the application on cores) and for all possible communication templates $\mathbf{q}$. In order to limit complexity, a heuristic is used to evaluate only a subspace of the possible mappings and the exploration is usually limited to a set of known topologies.

Our approach is substantially different. We start form an initial topology that is further down in the preorder, lower than meshes, tori, etc. We then use

our algorithms to minimize its cost by removing or contracting edges. Even if the algorithm complexity is the same, we explore a larger design space and we can potentially find a topology $\mathbf{p}$ such that $\Psi_l^s(\mathbf{p})$ does not compare with any known topology but is a heterogeneous network.

4. WIRELESS SENSOR NETWORK SYNTHESIS

The second application of our approach is the design of wireless sensor networks (WSN), an area of great interest in the research and industrial community. A WSN is a collection of wirelessly connected sensors, controllers, and actuators that are deployed in an environment and cooperatively work to implement a given monitoring or control application. The development of an operational WSN involves the cooperation of three different communities: at the application level, the end users (i.e., control engineers, mechanical engineers, biologists); at the node level, the designers of the wireless nodes together with their programmable interfaces; and the designers of the communication protocols. The interactions between design teams of different communities and companies have traditionally been a problem in handling complex design chains such as the one for WSN. In addition, it must be highly optimized for the use of the scarce resources that are usually available, and, being deeply embedded in their environment, must quickly adapt to the external changes that occur during operation. Hence, it is an ideal test case for PBD and the synthesis approach we advocate.

As in the case of on-chip networks, we identify a proper set of abstraction layers and formalize the refinement steps as a sequence of mappings onto more refined semantic domains. Although the platform-based design structure remains the same, the actual platforms and mapping tools are obviously specific to the WSN domain.

4.1 Sensor Network Service Platforms

At the application level, we introduce the highest layer of abstraction in our methodology, the sensor network service platform (SNSP) [Sgroi et al. 2004]. Similar to the role played by the Socket in Internet applications, an SNSP offers an application interface that is able to support the possible services that can be used in a WSN, independent of the network implementation.

To perform its functionality, a controller (algorithm) has to be able to read and modify the state of the environment. In a WSN, controllers do so by relying on communication and coordination among a set of distinct elements that are distributed in the environment in order to complete three different types of functions: sensing, control, and actuation. The role of the SNSP is to provide a logical abstraction for these communication and coordination functions. The SNSP offers a *query service (QS)* used by controllers to get information from other components, a *command service (CS)* used by controllers to set the state of other components, a *timing/synchronization service (TSS)* used by components to agree on a common time, a *location service (LS)* used by components to learn their location and a *concept repository service (CRS)* which maintains a map of the capabilities of the deployed system and is used by all the components

to maintain a common, consistent definition of the concepts they agreed upon during the network operation.

An SNSP is defined as a set of services to be made available to an application programmer independently of its implementation on any present and future sensor network platform. Refining an SNSP requires identifying the communication protocols among the nodes and implementing the middleware, as well as protocol functions, into physical nodes.

4.2 WSN Synthesis

Similarly to the on-chip network case, WSN synthesis consists of a sequence of successive refinements that starts with a description of the application already mapped onto the SNSP and delivers at the other end a network of wireless sensor nodes running a communication protocol. Although our methodology can be applied to a broad range of applications, for the sake of the description we consider here the simple example of a monitoring application where three robots must be checked for vibration and eventually powered down if the vibration values exceed a given threshold. We restrict the services of the SNSP to only the query service and command service, and the implementation space to a centralized control using a base station.

With reference to Figure 5, the highest level of abstraction is a functional description of the control algorithm. As explained in Bonivento et al. [2005], the control algorithm is specified as a sequence of queries and commands. Intuitively, a query is a request for sensed data from a specified area (i.e., a robot). Similarly, a command is a request for an actuation in a particular area. Restrictions on how queries and commands can be composed in the functional description depend on the model of computation that the end user wants to employ. The pseudocode for our example is shown by Algorithm 1.

The application consists of a cyclic routine that needs vibration data from each of the robots every 30 seconds. It further specifies the relative positions of three robots with respect to the controller, that the data should be sampled at 100 samples/seconds for 10 seconds, and that it should come back within 5 seconds with an error rate of 5%. It then averages the data collected from each

Algorithm 1

$Rob_1 \leftarrow (x_1, y_1)$
$Rob_2 \leftarrow (x_2, y_2)$
$Rob_3 \leftarrow (x_3, y_3)$
loop
  $Q_1 \leftarrow (Rob_1, Vib, All, scope = 10s, Sr = 100sam/s, L = 5s, Er = 5\%)$
  $Q_2 \leftarrow (Rob_2, Vib, All, scope = 10s, Sr = 100sam/s, L = 5s, Er = 5\%)$
  $Q_3 \leftarrow (Rob_3, Vib, All, scope = 10s, Sr = 100sam/s, L = 5s, Er = 5\%)$
  if $\overline{R(Q_1)} > V_{\max} || \overline{R(Q_2)} > V_{\max} || \overline{R(Q_r 3)} > V_{\max}$ **then**
    $C_1 \leftarrow (Rob_1, off)$
    $C_2 \leftarrow (Rob_2, off)$
    $C_3 \leftarrow (Rob_3, off)$
  end if
every 30s

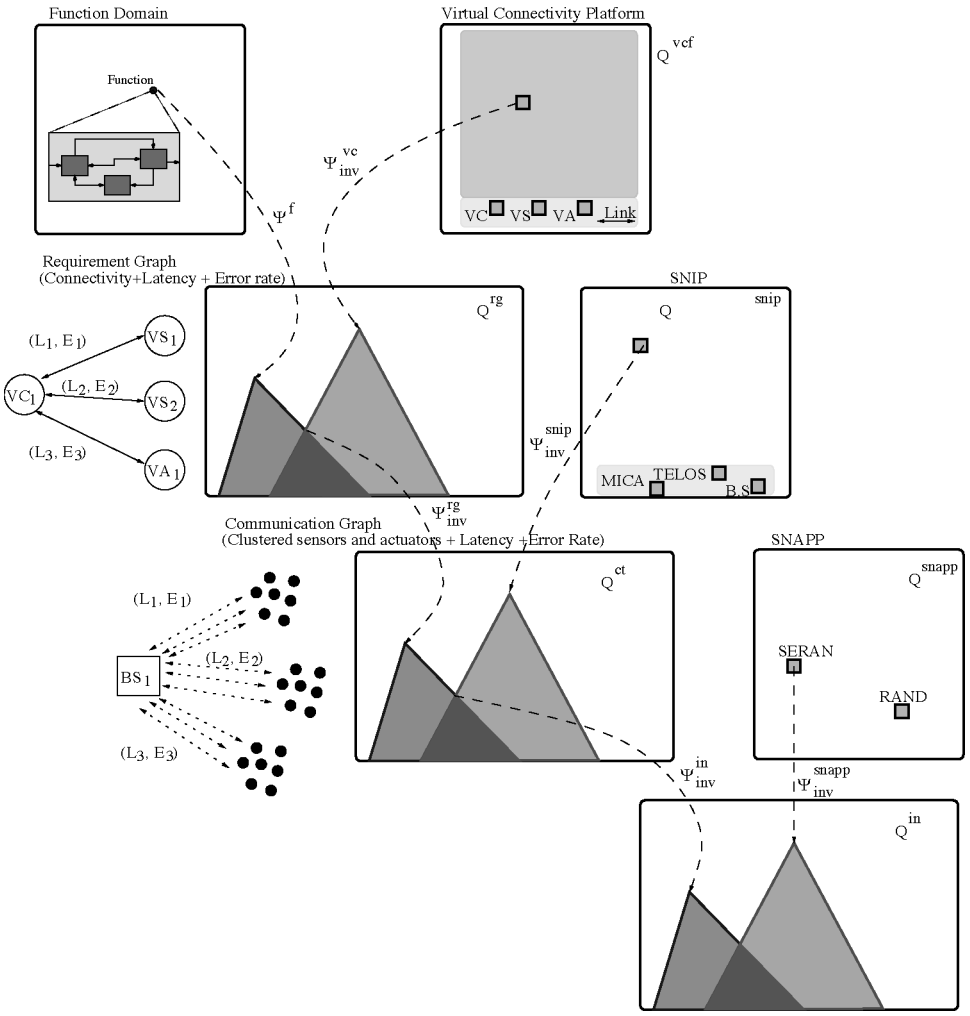

Fig. 5. Layers of abstraction and design flow.

robot and, if any of these three values is over a given threshold, shuts down all three robots.

The first platform is the virtual connectivity platform $\mathcal{Q}^{vcf}$. The library elements are of four types: the set of *virtual sensors* $\mathcal{S}_v$, virtual *controllers* $\mathcal{C}_v$, virtual *actuators* $\mathcal{A}_v$, and bidirectional links $\mathcal{L}$. A virtual sensor $\mathbf{s_v} \in \mathcal{S}_v$ is an abstraction of a sensing area; it is characterized by its position and will be refined later on in a cluster of sensor nodes. Similarly, a virtual actuator $\mathbf{a_v} \in \mathcal{A}_v$ is the abstraction of an actuation capability; it is characterized by its position and will eventually be refined in one or more actuators. A virtual controller $\mathbf{c_v} \in \mathcal{C}_v$ is an abstraction of a computation capability and, in this particular example, will be eventually refined in a base station. Virtual components and links can be composed to form other agents. It is not possible to directly connect virtual components, but links must be used. Furthermore, a virtual sensor can

be linked only to a virtual actuator, a virtual actuator can be linked only to a virtual controller, and virtual controllers cannot be directly linked. A partial order can be defined for this domain such that $v_1 \preceq v_2$ if and only if for each virtual component in v_2, there is a corresponding virtual component in v_1 and for each link in v_2, there is a corresponding link in v_1.

The common semantic domain for this step is called a *requirement graph* and is denoted by $\mathcal{Q}^{rg}$. This domain is similar to $\mathcal{Q}^{vcf}$, but links are annotated with a pair (L, E_r) that represents the end-to-end latency and error rate requirements, and virtual sensors are annotated with a pair (S_r, F) that represents the sensing rate requirement and the type of aggregate data (i.e., average value, max value, all values) for that area. While the rules of composition are the same as $\mathcal{Q}^{vcf}$, the order is such that $v_1 \preceq v_2$ if for each virtual component in v_2, there is a corresponding virtual component in v_1 with higher or equal S_r and the same F, and for each link in v_2 there is a corresponding link in v_1 with a smaller or equal latency and error rate. Intuitively, given two comparable instances of this domain, the highest is the one with looser communication and sensing requirements. We call ψ^f the conservative approximation that maps the functional description onto $\mathcal{Q}^{rg}$.

Formally, given an agent $\mathbf{r} \in \mathcal{Q}^{rg}$, we can define a conservative approximation as follows. The lower bound Ψ_l^{vc} abstracts the quantities S_r,F,L, and E_r. The upper bound Ψ_u^{vc} also abstracts all links. Agents $\mathbf{r} \in \mathcal{Q}^{rg}$ that are represented exactly in $\mathcal{Q}^{vcf}$ are agents with no links. We select an instance from the $\mathcal{Q}^{vcf}$ that has the minimum number of virtual components declared by the functionality. Then Ψ_{inv}^{vc} maps this instance onto $\mathcal{Q}^{rg}$. The cone in Figure 5 represents the set of agents that have the same number and types of virtual components as in the selected instance, but with links connecting them, and all possible combinations of latency, error rates, and sensing rates. For the sake of simplicity, in the sequel we do not specify the upper and lower bound for each conservative approximation whose construction should be intuitive.

The intersection of the two cones in $\mathcal{Q}^{rg}$ gives all the requirement graphs with connectivity, latency, error rates, and sensing rates sufficient to support the initial functionality. Among these possible refinements we choose the "highest," which is the one with the minimum number of virtual components and looser sensing, latency, and error rate requirements. We call this instance $\mathbf{r_g}$ and this is the starting point for the rest of the synthesis flow.

The tool that generates the instance $\mathbf{r_g}$ starting from the functional description is called Rialto [Bonivento et al. 2005]. Rialto starts from the sequential description of the control algorithm, then considers all possible branches in the decision tree of the algorithm and analyzes all the communication and sensing requirements for each of these branches. Starting from these requirements, it generates the minimum requirements that each link and virtual sensor have to satisfy so that the communication and sensing infrastructure is able to support the application in whatever decision branch it may end during the actual execution.

The next step is to refine $\mathbf{r_g}$, the requirement graph, into a clustered topology. The goal is to substitute virtual components with an adequate set of physical components.

To do this, we define the sensor network implementation platform $\mathcal{Q}^{snip}$. The elements of the library are a collection of *physical nodes* (e.g., Mica, Telos, and Intel motes), *base stations* (e.g., Stargate), and links. Nodes and base stations are characterized by their hardware abstraction (i.e., component size, memory, power consumption, clock speed), radio interface, sensing capabilities, location, and price. Physical components can be connected only using links. A link represents the capability of communication between two physical components. Restrictions on the possibility of directly linking two components reflect the reachability due to their radio interface. For example MicaZ and Telos motes can be linked since they both are ZigBee compatible, while Mica2 and MicaZ, that are not radio compatible, cannot be directly linked, but a path between the two can exist if there is also a third component (i.e., a base station or a node with a reconfigurable radio) that is able to support both radio interfaces. We can define a partial order similar to the one of $\mathcal{Q}^{rg}$, where the higher element is the one representing a minimum topology with looser requirements.

The common domain for this step is the *clustered topology* $\mathcal{Q}^{ct}$. An instance of this domain is a graph of interconnected physical components where the components are associated to a cluster. Furthermore, links are annotated with the usual requirement pair (L, E_r), components are annotated with a sensing requirement S_r, and clusters with the aggregate function type F. We can define a partial order in the same way as in $\mathcal{Q}^{rg}$, where the higher element is the one representing a minimum topology with looser requirements.

The instance $\mathbf{r_g}$ is mapped to a set of ordered agents in $\mathcal{Q}^{ct}$. Each agent in this set has at least a base station for each virtual actuator, a sufficient number of clustered sensor nodes for each virtual sensor so that the sensing requirement is satisfied, and a sufficient number of actuators for each virtual actuator. In addition, it must have a weighted link between the base station and each component with a latency and error rate less than or equal to the correspondent quantities between the virtual actuator and virtual sensor in $\mathbf{r_g}$.

Selecting an agent in $\mathcal{Q}^{snip}$ and mapping it to $\mathcal{Q}^{ct}$ means selecting the hardware platform and a lower bound on the density of nodes for each cluster. The mapping gives a set of ordered agents that may or may not intersect the set of agents obtained by mapping $\mathbf{r_g}$ to $\mathcal{Q}^{ct}$. This intersection represents all the clustered topologies with a sufficient number of interoperable nodes per cluster and connotations that are good enough to satisfy the link and sensing requirements.

We now have to eliminate a set of solutions that are unfeasible for any of the following reasons: size constraints (e.g., it is impossible to place 100 Telos motes in a square foot), external constraints (e.g., for regulatory issues, it is not possible to place the motes on some parts of the robots), and budget constraints (e.g., the overall solution has too many nodes and we cannot afford it). Among the remaining solutions, choosing the right one to propagate down in the synthesis process is not trivial. Given a number of nodes per cluster, we choose the solution with the loosest sensing and communication requirements. However, it is difficult to understand at this level what a good number of nodes per cluster is. On the one hand, more nodes involve a higher cost of the solution. On the other hand, the higher the density of the network, the more energy efficient the final solution will be because nodes can be duty cycled for energy preservation.

However, this energy consumption cannot be estimated until the communication protocol is decided and this happens at the next step of the design flow. Consequently, we suggest starting with a solution that is relatively high in the space of possible solutions (i.e., with a low number of nodes), and after the communication protocol is mapped, evaluating to see if the energy consumption per node is satisfactory for a good lifetime of the network. If this is not the case, the solution is to go back and select another possible solution with more nodes. As we will explain later, once the protocol is mapped on the nodes, the tradeoff curve between energy consumption and density is available. Consequently, a good number of nodes can be selected thus facilitating this iteration process.

As already mentioned, the last step is concerned with associating a communication protocol to the physical components such that the communication requirements are satisfied and the energy consumption minimized. To drive this step, we define the sensor network ad-hoc protocol platform (SNAPP), $\mathcal{Q}^{snapp}$. The library elements of the SNAPP are MAC and routing protocols. In the wireless sensor network domain space, layering between MAC and routing is usually not a good solution since it significantly reduces the energy optimization capabilities associated with crosslayer design. Consequently, the SNAPP is populated by noncomposable instances of integrated MAC and routing solutions. Different protocols have been developed for different application classes. For example, SERAN [Bonivento et al. 2005] was developed for periodic control applications with more than one cluster, while the randomized approach of Bonivento et al. [2006] (called RAND in Figure 5) is optimized for single cluster topologies. These protocols are "parametrized," meaning that their structure is specified, but their working point is determined by a set of parameters. For example, in SERAN the working point of the protocol is determined by a channel access probability p, a TDMA slot duration S, and a TDMA cycle duration Δ.

The common semantic domain in this step is represented by the instantiated network domain $\mathcal{Q}^{in}$. An instantiated network is an operational WSN, that is, a network of physical nodes with a communication protocol. Mapping the selected clustered topology onto this common domain, we obtain all the possible instantiated networks that satisfy the given E2E requirements on latency and error rate, while mapping a SNAPP instance we obtain all the possible instantiated networks that use the selected protocol with all the feasible combinations of the free parameters (i.e., p, S, and Δ for SERAN). The intersection between the two mappings gives all the possible instantiated networks that use the selected protocol and satisfy the given communication constraints. Among these solutions, we select the one that minimizes energy consumption. At this point we can evaluate if the synthesized solution can comply with the lifetime requirements of the network. If this is the case, we are done, otherwise we need to get back to the clustered topology domain and select an instance with more nodes.

This final refinement is obtained as the solution of a constrained optimization problem, where the constraints are the latency and error rate requirements while the cost function is the energy consumption that is estimated based on an abstraction of the physical properties of the candidate hardware platform.

An important and usually nontrivial step in solving this constrained optimization problem consists in translating the E2E requirements into hop-to-hop (H2H) requirements, and to project the energy cost of a single hop communication over a sequence of hops considering aggregate effects such as collisions and path reconvergences. The actual equations that perform these translations are different for different protocols and usually involve the manipulation of probabilistic functions. However, the ability to perform this refinement is subject to the capability of characterizing the interaction among the different layers of the protocol solution using a mathematical framework. The formalism and the capability of offering end-to-end guarantees instead of local guarantees are what distinguish our approach from the previous protocol design for WSNs. More detailed descriptions on how this optimization problem is solved, together with the initialization algorithms for the instantiated network for the different protocols are available in Bonivento et al. [2005, 2006].

4.3 Comparison with Other Approaches

Since we propose a design methodology that supports all phases of WSN design from application to implementation, there is quite a large body of related work. For sake of brevity, we outline only some recent approaches while we refer to Bonivento et al. [2006] for a more detailed analysis.

A system-level approach to the design of WSNs was recently presented in Polastre et al. [2005]. A platform called SP is proposed between the link and the network layer. The SP should provide adequate modularity for the nodes to support different MAC and routing layers. The philosophy is similar to the Internet "everything over IP," where in this case it would be "everything over SP." Although this is a very interesting architecture for best effort networks, we believe it is not appropriate for control applications where E2E guarantees are required. Our top-down approach and synthesis method are customized for control applications.

An attempt at raising the level of abstraction was presented in Yu et al. [2005], where a classification for node communication mechanisms was introduced to allow for a higher-level description of the network algorithms. In Bakshi and Prasanna [2004], the proposed methodology is based on a bottom-up part for the description of network algorithms, a top-down part to describe the application, and a mapping process to deploy software onto the nodes. Although the overall method is compatible with the platform-based design paradigm advocated in this article, the layers of abstraction are quite different. Our approach emphasizes the control-based nature of WSN applications and offers a rigorous semantics and set of primitives to interpret timing issues at a very high level, hence providing a well-defined level of abstraction for the application designer.

5. CONCLUSIONS

We strongly believe that system-level design needs a major breakthrough to resolve the many issues facing the electronics industry. No matter what this breakthrough is going to be, we believe that it will be based on rigorous

mathematical foundations and a unifying approach that can be applied to all levels of abstraction of design. In this article, we presented a formalization for platform-based design, a design approach proposed for all levels of system design, and examples of the application of this formalized approach to communication synthesis, an important step complex designs in itself towards making both feasible in a reasonable time and correctly.

REFERENCES

AHONEN, T., SIGUENZA-TORTOSA, D. A., BIN, H., AND NURMI, J. 2004. Topology optimization for application-specific networks-on-chip. In *Proceedings of the 2004 International Workshop on System Level Interconnect Prediction*. ACM Press, New York, 53–60.

BAKSHI, A. AND PRASANNA, V. 2004. Algorithm design and synthesis for wireless sensor networks. In *Proceedings of the International Conference on Parallel Processing*.

BALARIN, F., LAVAGNO, L., PASSERONE, C., SANGIOVANNI-VINCENTELLI, A. L., SGROI, M., AND WATANABE, Y. 2002. Modeling and designing heterogeneous systems. In *Concurrency and Hardware Design, Advances in Petri Nets*. Springer Verlag, London, 228–273.

BENINI, L. AND MICHELI, G. D. 2002. Networks on chips: A new SOC paradigm. *Computer 35*, 1, 70–78.

BERTOZZI, D. AND BENINI, L. 2004. Xpipes: A network-on-chip architecture for gigascale systems-on-chip. *IEEE Circuits Syst. 4*, 2, 18–31.

BERTOZZI, D., JALABERT, A., MURALI, S., TAMHANKAR, R., STERGIOU, S., BENINI, L., AND MICHELI, G. D. 2005. NOC synthesis flow for customized domain specific multiprocessor systems-on-chip. *IEEE Trans. Parallel Distrib. Syst. 16*, 2, 113–129.

BONIVENTO, A., CARLONI, L., AND SANGIOVANNI-VINCENTELLI, A. 2005. Rialto: A bridge between description and implementation of control algorithms for wireless sensor networks. In *Proceedings of the 5th ACM International Conference on Embedded Software*. Jersey City, NJ.

BONIVENTO, A., CARLONI, L., AND SANGIOVANNI-VINCENTELLI, A. 2006. Platform based design for wireless sensor networks. To appear in *MONET*.

BONIVENTO, A., FISCHIONE, C., AND SANGIOVANNI-VINCENTELLI, A. 2006. Randomized protocol stack for ubiquitous networks in indoor environment. In *Proceedings of the IEEE Consumer Communications and Networking Conference*. Las Vegas, NV.

BONIVENTO, A., FISCHIONE, C., SANGIOVANNI-VINCENTELLI, A., GRAZIOSI, F., AND SANTUCCI, F. 2005. Seran: A semi random protocol solution for clustered wireless sensor networks. In *Proceedings of the International Conference on Multi-Agent Systems*. Washington, DC.

CHAKI, S., RAJAMANI, S., AND REHOF, J. 2002. Types as models: Model checking message-passing programs. In *Proceedings of the 29th ACM Symposium on the Principles of Programming Languages*.

CHAKRABARTI, A., DE ALFARO, L., HENZINGER, T. A., JURDZINSKI, M., AND MANG, F. Y. C. 2002. Interface compatibility checking for software modules. In *Proceedings of the 14th International Conference on Computer-Aided Verification (CAV)*. Lecture Notes in Computer Science, vol. 2404. Springer Verlag, 428–441.

CHAKRABARTI, A., DE ALFARO, L., HENZINGER, T. A., AND STOELINGA, M. 2003. Resource interfaces. In *Proceedings of the 3rd International Conference on Embedded Software*. Lecture Notes in Computer Science, vol. 2855. Springer Verlag, New York.

CHANG, H., COOKE, L., HUNT, M., MARTIN, G., MCNELLY, A. J., AND TODD, L. 1999. *Surviving the SOC Revolution. A Guide to Platform-Based Design*. Kluwer Academic Publishers, Norwell, Mass.

CHARIKAR, M. AND KARAGIOZOVA, A. 2005. On non-uniform multicommodity buy-at-bulk network design. In *STOC '05: Proceedings of the 37th Annual ACM Symposium on Theory of Computing*. ACM Press, New York, 176–182.

DALLY, W. J. AND TOWLES, B. 2001. Route packets, not wires: On-Chip interconnection networks. In *Proceedings of the Design Automation Conference*. Las Vegas, Nev., 684–689.

DE ALFARO, L. AND HENZINGER, T. A. 2001. Interface theories for component-based design. In *Proceedings of the 1st International Workshop on Embedded Software*. Lecture Notes in Computer Science vol. 2211, Springer Verlag, 148–165.

ERNST, R., HENKEL, J., AND BENNER, T. 1993. Hardware-software cosynthesis for microcontrollers. *IEEE Des. Test 10*, 4, 64–75.

FERRARI, A. AND SANGIOVANNI-VINCENTELLI, A. L. 1999. System design: Traditional concepts and new paradigms. In *Proceedings of the 1999 IEEE International Conference on Computer Design*. IEEE Computer Society, Washington, DC.

GAJSKI, D., VAHID, F., NARAYAN, S., AND GONG, J. 1998. Specsyn: An environment supporting the specify-explorerefine paradigm for hardware/software system design. *IEEE Trans. VLSI 6*, 1, 84–100.

GAREY, M. AND JOHNSON, D. 1979. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman and Company.

GASTEIER, M. AND GLESNER, M. 1998. Generation of interconnect topologies for communication synthesis. In *DATE '98: Proceedings of the Conference on Design, Automation and Test in Europe*. IEEE Computer Society, Washington, DC, 36–43.

GUPTA, R. K. AND MICHELLI, G. D. 1993. Hardware-software cosynthesis for digital systems. *IEEE Des. Test Comput.* (Sept.), 29–41.

HU, J., DENG, Y., AND MARCULESCU, R. 2002. System-Level point-to-point communication synthesis using floorplanning information. In *Proceedings of the Asia South Pacific Design Automation/VLSI Design Conference*.

HU, J. AND MARCULESCU, R. 2003. Energy-Aware mapping for tile-based NOC architectures under performance constraints. In *Proceedings of the Asia and South Pacific Design Automation Conference*.

HU, J. AND MARCULESCU, R. 2004. Dyad: Smart routing for networks-on-chip. In *Proceedings of the 41st Annual Conference on Design Automation*. ACM Press, New York, 260–263.

LAHIRI, K., RAGHUNANTHAN, A., LAKSHMINARAYANA, G., AND DEY, S. 2004. Design of high-performance system-on-chips using communication architecutre tuners. *IEEE Trans. CAD 23*, 5, 620–636.

LI, Y. AND WOLF, W. 1998. Hardware/Software co-synthesis with memory hierarchies. In *ICCAD '98: Proceedings of the 1998 IEEE/ACM International Conference on Computer-Aided Design*. ACM Press, New York, 430–436.

MURALI, S. AND MICHELI, G. D. 2004. Sunmap: A tool for automatic topology selection and generation for NOCS. In *Proceedings of the 41st Annual Conference on Design Automation*. ACM Press, New York, 914–919.

ORTEGA, R. B. AND BORRIELLO, G. 1998. Communication synthesis for distributed embedded systems. In *ICCAD '98: Proceedings of the 1998 IEEE/ACM International Conference on Computer-Aided Design*. ACM Press, New York, 437–444.

PASSERONE, R. 2004. Semantic foundations for heterogeneous systems. Ph.D. thesis, University of California, Berkeley.

PASSERONE, R., DE ALFARO, L., HENZINGER, T. A., AND SANGIOVANNI-VINCENTELLI, A. L. 2002. Convertibility verification and converter synthesis: Two faces of the same coin. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*.

PASSERONE, R., ROWSON, J. A., AND SANGIOVANNI-VINCENTELLI, A. L. 1998. Automatic synthesis of interfaces between incompatible protocols. In *Proceedings of the Design Automation Conference*, San Francisco, Calif.

PINTO, A., CARLONI, L. P., AND SANGIOVANNI-VINCENTELLI, A. L. 2002. Constraint-Driven communication synthesis. In *DAC '02: Proceedings of the 39th Conference on Design Automation*. ACM Press, New York, 783–788.

PINTO, A., CARLONI, L. P., AND SANGIOVANNI-VINCENTELLI, A. L. 2003. Efficient synthesis of networks on chip. In *Proceedings of the 21st International Conference on Computer Design*, 5.

POLASTRE, J., HUI, J., LEVIS, P., ZHAO, J., D. CULLER, S. S., AND STOICA, I. 2005. A unified link abstraction for wireless sensor networks. In *Proceedings of the 3rd International Conference Embedded Networked Sensor Systems*.

PRAKASH, S. AND PARKER, A. C. 1992. Synthesis of application-specific heterogeneous multiprocessor systems (abstract). In *Proceedings of the 19th Annual International Symposium on Computer Architecture*. ACM Press, New York, 434.

RHODES, D. L. AND WOLF, W. 1999. Co-Synthesis of heterogeneous multiprocessor systems using arbitrated communication. In *Proceedings of the 1999 IEEE/ACM International Conference on Computer-Aided Design*. IEEE Press, Piscataway, NJ, 339–342.

ROWSON, J. A. AND SANGIOVANNI-VINCENTELLI, A. 1997a. Interface-Based design. In *Proceedings of the Design Automation Conference.*

ROWSON, J. A. AND SANGIOVANNI-VINCENTELLI, A. L. 1997b. Interface-Based design. In *Proceedings of the 34th Design Automation Conference*, 178–183.

SGROI, M., SHEETS, M., MIHAL, A., KEUTZER, K., MALIK, S., RABAEY, J., AND SANGIOVANNI-VINCENTELLI, A. 2001. Addressing the system-on-a-chip interconnect woes through communication-based design. In *Proceedings of the Design Automation Conference*.

SGROI, M., WOLISZ, A., SANGIOVANNI-VINCENTELLI, A., AND RABAEY, J. 2004. A service-based universal application interface for ad-hoc wireless sensor networks. In *Whitepaper*, U.C. Berkeley.

SHIMIZU, K. AND DILL, D. L. 2002. Deriving a simulation input generator and a coverage metric from a formal specification. In *Proceedings of the Design Automation Conference*. New Orleans, La.

SRINIVASAN, K., CHATHA, K. S., AND KONJEVOD, G. 2004. Linear programming based techniques for synthesis of network-on-chip architectures. In *Proceedings of the IEEE International Conference on Computer Design*, 422–429.

VAZIRANI, V. 2003. *Approximation Algorithms*. Springer Verlag, Berlin.

VINCENTELLI, A. S. 2002. Defining platform-based design. *EEDesign of EETimes*.

WANG, H., PEH, L.-S., AND MALIK, S. 2005. A technology-aware and energy-oriented topology exploration for on-chip networks. In *Proceedings of the Design Automation and Test in Europe Conference*, 1238–1243.

WANG, H.-S., ZHU, X., PEH, L.-S., AND MALIK, S. 2002. Orion: A power-performance simulator for interconnection networks. In *Proceedings of the 35th Annual ACM/IEEE International Symposium on Microarchitecture*. IEEE Computer Society Press, Los Alamitos, Calif., 294–305.

WYNANTS, C. 2001. *Network Synthesis Problems*. Kluwer Academic, Hingham, Mass.

YEN, T.-Y. AND WOLF, W. 1995. Communication synthesis for distributed embedded systems. In *ICCAD '95: Proceedings of the 1995 IEEE/ACM International Conference on Computer-Aided Design*. IEEE Computer Society, Washington, DC, 288–294.

YU, Y., HONG, B., AND PRASANNA, V. 2005. Communication models for algorithm design in wireless sensor networks. In *Proceedings of the IEEE International Parallel and Distributed Processing, Symposium*.

Received February 2006; accepted May 2006