Pre-print version

C. Alippi, V. Piuri, F. Scotti, "High-level design of composite systems", in *Soft computing in software engineering: theory and applications*, Springer, pp. 183-220, 2004.

ISBN: 3540220305

High-level design of composite systems

C. Alippi, V. Piuri and F. Scotti

Cesare Alippi:
Dept. Electronics and Information, Politecnico di Milano, piazza Leonardo da Vinci 32, 20133 Milano, Italy

Vincenzo Piuri, Fabio Scotti:
University of Milan, Department of Information Technologies
via Bramante 65, 26013 Crema (CR), Italy

Composite systems, encompassing soft-computing and conventional processing modules, are attractive approaches to tackle complex applications, especially those requiring adaptability and generalization abilities.

This chapter presents an innovative high-level design methodology that provides a comprehensive framework to achieve the most efficient and effective solutions. The system is first specified at a high-abstraction level by using description languages suited to capture the system behavior and the other non-functional constraints. In the computational paradigm co-design the system description is then partitioned into modules, described with various computational paradigms, and modules are completely configured. The result is an algorithmic description of the whole system, from which conventional hardware/software co-design techniques can derive the final implementation.

7.1 Introduction

Conventional information processing approaches are often highly efficient in solving complex applications. Despite their effectiveness, the resulting algorithms need to be completely specified and, hence, usually show poor flexibility and adaptability in correspondence with unforeseen operating conditions. Conversely, soft-computing techniques have proven to be useful both in achieving intrinsic adaptability and learning the desired system behavior from experience.

A solution solely based on conventional or soft-computing techniques is not optimal in many real applications where, for instance, it lacks in performance, accuracy and adaptability. In these cases some modules composing the information processing flow can be easily and efficiently described and realized by means of conventional solutions, while others take advantages from soft-computing methods by exploiting learning ability and embedded adaptability. A synergic cooperation of conventional and soft-computing methods is often the winning approach to implement complex heterogeneous information processing computation; we define the resulting system as *composite*, i.e., composed of modules of both kinds (Alippi et al. 1999).

To design a composite system the designer has to specify its functionalities, partition the same into sub-systems having manageable size and suited to be solved with a single computational paradigm (either conventional or soft-computing), specify and implement each module accordingly, and finally integrate them. This high-level design approach resembles the conventional hardware/software co-design methodologies (De Micheli and Sami 1995) that are well assessed and widely adopted in the industry when the system behavior has been completely defined (the system computational flow is partitioned in modules to be implemented in hardware and in software components to be synthesized on the system hardware itself).

In today composite systems design, the designer has to manually partition the composite system into homogeneous modules to be implemented with a single computational paradigm by using a trial-and-error approach since no automatic comprehensive design tool is available. Then, he/she can exploit various development tools and techniques to synthesize each module, according to their specific nature. For conventional components the traditional hardware/software co-design environments can be adopted. For soft-computing modules specialized techniques and tools are required; when they have been completely configured the result is an algorithmic description which can be fed into hardware-software co-design environments (Alippi *et al*. 1999).

In carrying out composite systems decomposition the designer is completely responsible of the whole design phases, starting from partitioning the system into modules, each realized with the most suited computational paradigm, and carrying out the final integration. Taking care of all the interdependencies among the resulting modules is an additional element, which further characterizes the design difficulty. As a consequence, the designer task, without any comprehensive methodological guidance and integrated development environment support, must posses a strong knowledge

and awareness of all soft-computing and partitioning techniques. The designer's expertise, experience, and abilities must be highly specialized and differentiated in order to effectively and efficiently deal with this heterogeneous world.

The chapter aims at introducing an innovative homogeneous, comprehensive high-level design methodology for composite systems that supports high-level specification, synthesis, optimization and system validation. The chapter presents also the integration of the available development tools into a global and flexible design framework.

First of all, the desired behavior of the composite system is specified by means of a high-level description language that is able to capture and characterize the behavior of a specific sub-system in the most natural way (e.g., by examples, algorithms) (Alippi *et al*. 1999, Gajski *et. al.* 1995, Hilfinger and J. Rabaey 1992, Perry 1991). The designer may also specify non-functional characteristics (e.g., latency, throughput, accuracy, power consumption) to ensure satisfaction of application constraints (Ashenden 2002, Gajski *et al.* 1995, Newton *et al.* 2002).

The high-level description of each sub-system is then captured and modeled by the computational paradigm that is the most suited to represent the expected behavior and non-functional constraints. Without loss of generality, in this chapter we focus the attention on neural networks; extension to other soft-computing paradigms is feasible and can be achieved by introducing the corresponding design methods and tools. Sub-systems can be partitioned and grouped in order to enhance the quality of the resulting overall system.

The computational paradigm adopted for each sub-system needs to be fully configured on the corresponding specifications (e.g., a neural network must be trained). Independently from the kind of paradigm adopted to model the sub-system description, the resulting information processing consists of well-defined sequences of operations.

The above activities can therefore be viewed as a *multi-paradigm co-design process*, in which the system's high-level specifications are transformed into the system's comprehensive algorithmic description. This complex process consists of the exploration of the design space in order to analyze the different alternative paradigms and the related hardware/software implementations, by allowing for the reuse of existing modules, exploiting available information and experience gained in similar applications, monitoring the solution's quality (e.g., accuracy, computational complexity, latency, throughput, and other features) at a high- abstraction level.

Then, the conventional *hardware/software co-design process* can be used to select the most appropriate implementation, possibly by restructuring the overall system description. By starting from the algorithmic description given in a formal language (e.g., C, C++, VHDL, Verilog, SystemC), this design activity produces the physical description of implementation (i.e., the hardware architecture – encompassing, e.g., microprocessor-based structures, ASIC components, FPGA modules, memory banks – and the related software modules).

The situation can be summarized as in Figure 1 where it is described the complete design process for composite systems, from the high-level description down to the hardware/software implementation. In particular, it points out the relative position of the *multi-paradigm co-design* onto the state of the art of the conventional high-level system design (*HW/SW co-design*).

The structure of the chapter is as follows. Section 2 provides the module characterization and the system's high-level specification. Section 3 introduces the high-level design methodology. In section 4, the use of this methodology is applied to some design problems, including some industrial cases.

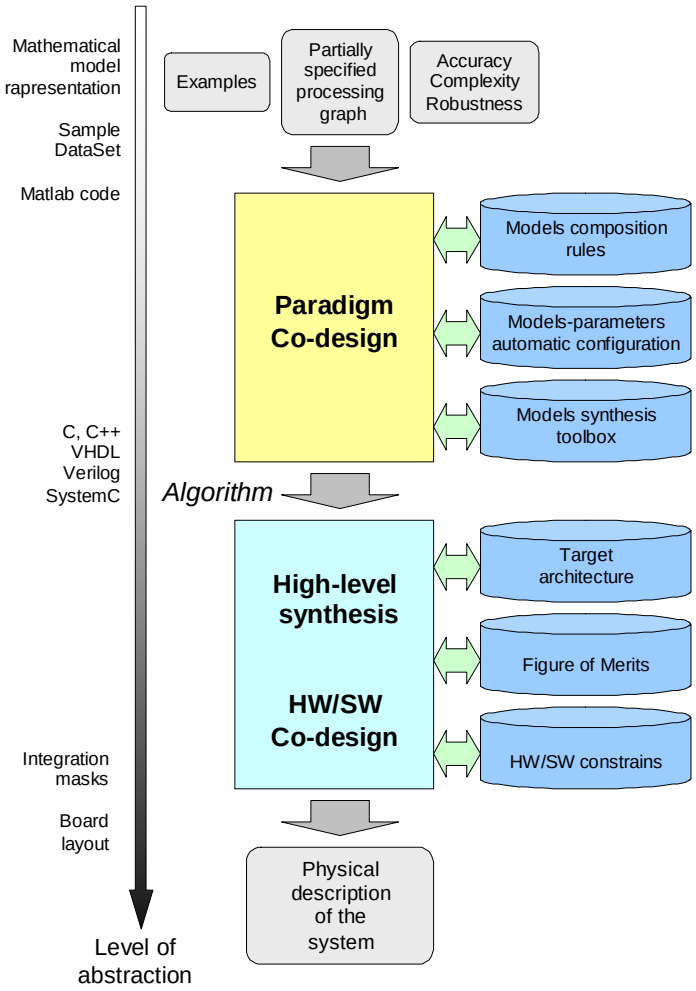

Fig. 1. The high-level design of a composite system. The lowest module represents the state-of-the-art in high-level hardware/software co-design. The highest module refers to the multi-paradigm co-design.

7.2 High-level specification of composite systems

To support high-level design of composite systems a suited specification methodology is needed. Such methodology allows the designer to characterize the behavior and the non-functional characteristics of the system in a straightforward way.

When the system is rather simple, well-defined, and its operations are clear, the designer finds intuitive to describe the desired behavior and constraints by means of an algorithm coded in a structured language. When mathematical relationships are well known among inputs and the desired output of the sub-system (system), an equation-based description may be fully satisfactory and easy to be given. Conversely, when the solution is too complex to be described or unavailable and only a set of examples are given, specification by examples is the most suitable description approach. The adopted methodology and the related specification language must be able to describe the system by using any of these characterizations in order to allow the designer to focus on the problem specification rather than on the difficulties of the language used to provide the specification.

In addition, the design methodology must partition the complex system into sub-systems (*simple modules*) so that each of them has manageable size and can be specified effectively and efficiently by using a single specification approach. This leads to a specification in which simple modules are connected together in a graph completely describing the system.

In this section the concept of simple module is first introduced, then the graph structure for complex composite system specification is given. Finally, the comprehensive specification framework is presented to express homogeneously all application needs for the composite system.

Nomenclature

Systems that can be naturally, fully, and accurately described in a homogeneous way by means of a single specification approach are called simple modules. A module can be characterized at various *levels of abstraction*, according to the amount of features that the designer must provide.

A simple abstract module can be described at one of the following levels:

- *Algorithm*: the module is completely specified and characterized by the sequence of operations that must be performed to generate the desired outputs from the given inputs. This description can be implemented either in software by transforming the algorithm into a program or in hardware by realizing, e.g., an ASIC or an FPGA-based solution.
- A *configured model*, $\overline{M}$, is an equation-based model, in which both the topology and the model parameters are available. Examples are the FFT transform, the butterworth low-pass filter working at a given frequency,

and a neural network that has been completely trained on a given data set. Since all operations are exactly known, the description of a configured model is a final algorithm.

- A *family of models*, M_i, is a group of models characterized by a common structure with fixed topology, but with unspecified configuration parameters. A configured model is obtained by fixing the parameters, e.g., by applying a suited parameters configuration procedure on a given training data set (*training phase*). Examples are a low-pass filter with an unspecified cutting frequency, and a neural network with unspecified interconnection weights but fixed topology.
- A *union of model families*, $\Gamma_M = \{M_1, M_2, ..., M_N\}$, is the common structure of a group of model families in which neither the topology nor the parameters have been set. If there is a hierarchical ordering among the model families (i.e., $\forall M_i \in \Gamma_M, M_1 \subseteq ... M_i ... \subseteq M_N$), the union is called a *hierarchy of model families*. The Nearest-Neighbor classifier (*kNN*) constitutes a union of model families $\{1NN_{Z_1}, 2NN_{Z_1}, 1NN_{Z_2}, ..., kNN_{Z_N}\}$, where, Z_i represents the different training matrix embedded in the *k*NN. The families are not nested and, hence, the union is not also a hierarchy. ARX and ARMAX systems with an unspecified number of delay taps constitute hierarchies of model families. A neural network family, M_i, with a single hidden layer of *i* neurons is another example of hierarchy. Higher complexity families degenerate in simpler ones by acting on some topological parameters. When the topology has been completely selected and fixed, the hierarchy (or union) becomes a family of models with unspecified model parameters.
- The *black box* is a model-free description of the desired behavior. The computation to be performed by the module is completely unknown, but the desired behavior is specified by a set of examples. An example is the set of input-output pairs that the box is expected to satisfy. The designer must select the union/hierarchy of model families that best capture the black-box hidden behavior.
- The abstraction levels range therefore from the lowest (algorithm) level where the whole computation is procedurally specified to the highest (black-box) level where only the external behavior is described by examples.

The design environment for composite system has supported the whole specification refinement and design process leading to the completely configured models, possibly starting from the black-box description of the

module behavior. This is accomplished by choosing the hierarchy of model families, the specific model family within such a hierarchy, the particular model within the envisioned family, and finally by configuring the model parameters.

The sequencing graph of a composite system

The specification of a complex system may become easier by partitioning the overall description in a set of interconnected simple modules, described by the *composite-system sequencing graph* (CSSG). Each module of the same composite system can be described at a different abstraction level, according to the approach that is most suited and natural to the designer. The sequencing graph defines the interconnections among modules, i.e., which are the inputs of each module and where the output produced by a module must be delivered. This implicitly defines also the precedence relationships in the execution order of the modules.

An example is given in Figure 2. The simple module is described with a black-box approach. The complex system is decomposed in various modules connected in a feed-forward scheme: the *NN* module represents model hierarchies (feed-forward neural networks), the *ALGO* module represents a conventional module (algorithm) written by the designer and *M* a black-box module.

The designer can refine an initial specification through a sequence of steps in which a complex module can be partitioned into smaller, simpler, and more accurate ones. The new modules introduced in this way are defined by using the most appropriate specification approach, are possibly different among them and even different from the original complex module.

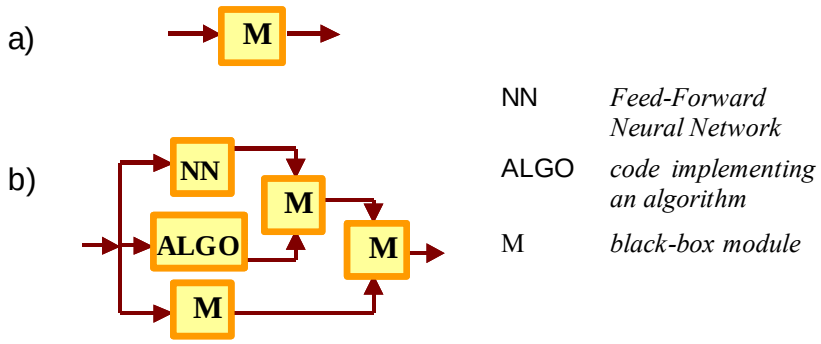

Fig. 2. Composite-System Sequencing Graphs: (a) a simple module, (b) a more complex case.

When the designer initially specifies the system at a very high abstraction level, he can use a black-box approach. If he is aware of any relationship or characteristic in the overall system, he can exploit it and create an initial structured description of the system consisting of modules and related CSSG.

During the subsequent specification refinements as well as in the subsequent multi-paradigm partitioning design phase, the designer transform the current CSSG into another CSSG where the modules are described at lower levels of abstraction. As already said in section 2.1, this is accomplished in the subsequent design phases by selecting the structure, the topology, and the parameters until the algorithmic description is achieved. In the specification phase, the abstraction level of a module may be lowered by *decomposing* the module into simpler ones. The decomposition of a module *M* can be performed in series or parallel by splitting the description of the initial module and characterizing each new module so as the combined specification coincides with the one of the original module. This operation can be iterated until simple modules are generated. In real applications CSSG partitioning is iterated only for very few iterations since further refinements do not generally provide significant improvements. Examples of CSSG decompositions are shown in Figure 3.

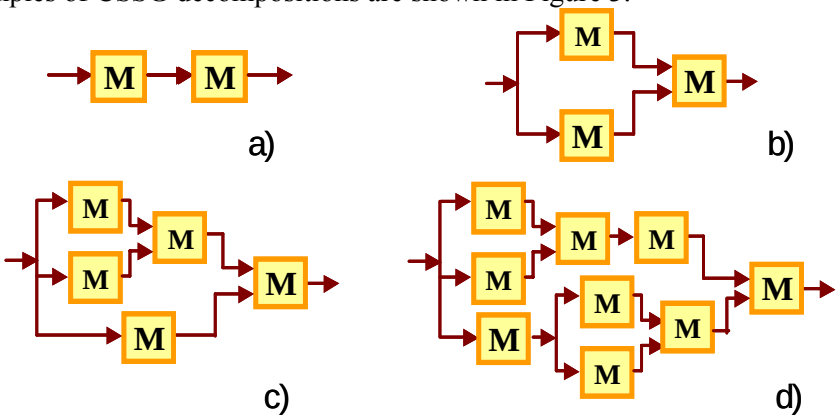

Fig. 3. Module decomposition: (a) series, (b) parallel, (c, d) iterative series-parallel decompositions.

Additional operations to manipulate the CSSG's can be introduced, e.g., moving modules along the CSSG paths, cloning modules, collapsing mod-

ules. These operations are directed to allow the final CSSG to be simplified yet granting the required performance.

If the execution order of two subsequent modules, M_1 and M_2, is not relevant, their order can be inverted by *moving* M_2 before M_1. This operation can be iterated so as to generate new configurations for the CSSG. Similarly, a module can be replicated by the *cloning* operator: identical copies replace the original one on the other side of the CSSG node. This move enables a further application of the moving operation.

Modules that are adjacent in the CSSG can be merged to generate a single module by means of the *collapsing* operator. The specification of the resulting module must coincide with the behavior described by the collection of the original interconnected modules. Namely, it is a suited combination of the specifications of the original modules; such a comprehensive specification takes into account the information flow defined by the portion of the CSSG among the original modules. Collapsing reduces the complexity of the CSSG and may allow the overall complexity of the resulting modules to be reduced and increasing the overall specification quality.

Behavioral and non-functional specifications

Specifications characterizing the behavior of a module are a collection of information about the input/output relationship to be implemented and the constraints it must be performed and implemented. For example, for a neural network module, the behavior is specified by means of a set of examples and a set of mathematical features associated with the desired relationship (e.g., smoothness, differentiability) or with environmental properties (e.g., the model of the noise affecting the data).

The main entities (*variables*) that can be used to express system features are related to behavioral characteristics, performance indices, and structural peculiarities.

The main *design variables* are:

– The model hierarchy or union, Γ_M. Each model family, M_i, belonging to Γ_M represents the set of functions $M_i(\theta)$ which can be obtained by allowing the vector of the free parameters of the model θ to span its domain. A particular model of the family is obtained by specifying the parameter vector θ (this procedure is carried out in the model configuration phase).

– The complexity measure, p, is a measure of the number of free parameters θ of the model associated with the family, M_p; for example: the number of hidden units of a feed-forward neural network, the VC dimension [9], the number of connections. This value characterizes the complexity of the model.
– The set of input/output examples, $Z_c = \{(x_i, y_i)\}$.
– The optimization figure of merit J is used to configure/ validate the model. For instance, it can be the mean squared error augmented with some penalty term such as the weight decay.

The configuration algorithm, C, that provides the best parameter vector by optimizing J. The choice of the configuration algorithm is related the structure of the model hierarchy. If the function to be configured is non-continuous or the model is not differentiable, a gradient descent based optimization algorithm cannot be directly used; this paradigm class is large enough to enclose the most relevant feed-forward and the recurrent neural networks. Soft-computing paradigms that generally present irregular internal structure (e.g., Bayesian networks, fuzzy systems) can be included as well by providing some limitations on their structure in order to be described as a hierarchy Γ.

The main *behavioral specification variables* are:

– The input/output data set, $Z_N = \{(x_i, y_i) | i = 1, 2, \ldots, N\}$, is the complete collection of available examples of the input/output relationship to be realized. In general, it contains the data set used to configure model Z_c. Depending on the envisaged configuration technique Z_N can be partitioned in order to dedicate some input/output examples to the model selection and validation. In the case of unsupervised learning the data set is
$Z_N = \{(x_i) | i = 1, 2, \ldots, N\}$.
– The accuracy, A, is a performance measure for model $M_i(., \theta)$. In systems trained from examples it is the generalization property. Mean square error or entropy-based figures of merit can be used to specify the accuracy constraints.
– The robustness, R, is a measure of the model sensitivity to the perturbations affecting the computation.
– The mathematical features of the input/output relationship, MF, contains all the analytical information related to the relationship itself that the system is supposed to realize; for example, they can be domain and co-domain constraints, or continuity and smoothness requirements for the model.

– The computational cost, *Cost*, is the computational complexity tolerated for the system. This complexity will constrain the number of instructions in a software realization, or the number of functional units in a hardware implementation.
– The main *physical specification variables* are:
– The number of bits, N_{bits}, is the number of bits used for the representation and processing of numerical quantities.
– The number of pins, N_{pins}, is the number of input/output pins of the physical device.
– The area, *Area*, is the maximum silicon area available to the component.
– The latency time, *L*, takes into account the maximum time allowed to the component for computing the output.
– The economical cost, γ, of the component realization.

Although the constraints on the physical variables can be accurately evaluated only in latest phases of the design activity, they have to be considered as early as possible in order to guide the initial design phases directly towards the most suited solutions from the beginning of the design process, thus limiting backtracking and the consequent increase of design efforts: the impossibility of satisfying a physical constraint may involve the redesign of parts of the system with additional design time and cost.

In real applications, considering the need to state specifications at the highest abstraction level, the most commonly used variables are: p, Z_c, and J (or – equivalently – Z_N and V) for the design variables, and *Cost* and R for the behavioral specification variables.

Evaluation of accuracy, complexity and robustness at a high abstraction level

To evaluate the module accuracy, A, we assume to have the sample data set, Z_N. The designer can partition this set in two sub-sets: the training data set, Z_{train}, and the validation data set, Z_{val}. In the case of unsupervised models the data set consists only of the input examples.

An index to measure the module accuracy is the mean squared error evaluated on Z_{val}. Other methods are the apparent error rate (McLachlan 1976), the random sub-sampling (Higleyman 1962), the w-fold crossvalidation (Higleyman 1962), or bootstrapping (Vapnik 1998).

To evaluate the computational complexity of a composite system a first approach consists in measuring the time needed to execute the code of the system by using a reference data set. The method is simple since it meas-

ures the time to execute the final module. In software simulation, this method may provide a very rough estimate of the computational complexity since real-time execution depends also on the activities of the operating system underlying the development environment for composite systems; these operations are not constant since they may be influenced by swapping and caching activities.

A better method for measuring the overall time actually spent evaluates the computational complexity as:

$$C = \alpha_1 N_{Float} + \alpha_2 N_{\sum} + \alpha_3 N_{Prod} + \alpha_4 N_{(R/W)_{Mem}} + \alpha_5 N_{(R/W)_{LUT}} \quad (1)$$

where N_x is the number of operations of type x, and α_x is number of elementary time units required to perform one operation of this type. For example, C can differently weight operations such as the integer sums and the multiplications, the floating point ones, and the access to the memory.

For a given system, a proper robustness index measures the variation in accuracy induced by perturbations affecting the parameters of the composite system (e.g., finite precision errors, physical production, process fluctuations) at very high level of abstraction is given in Alippi 1999 and Alippi 2002.

7.3 High-level design of composite systems

The high-level specification of the composite system modules must be transformed into a lower-level homogeneous description that completely characterizes the system behavior and the various implementation constraints. This final description consists of the algorithms (coded by using a programming language) describing the sequence of operations associated with the activities of the various modules. Such a description can be used as input for the hardware/software co-design phase, which will lead to the final synthesis of the composite system by using the conventional hw/sw co-design development environments.

The methodological approach

The activities to be accomplished for an effective composite system design can be summarized in four conceptual steps:

1. The *topological module decomposition* step generates the structure of the composite system starting from specifications. This is an iterative

process that partitions the modules into sub-modules having –possibly– different nature, until the system contains only simple modules (*SM*). Simple modules are the atomic entities that can be easily synthesized. For instance, each *SM* can be either a conventional algorithm or a neural network.

2. The *model family selection* step associates a specific model family to the unspecified conventional or neural network module. Selection rules identify the most promising family based on high-level information about the nature of the application and the module (e.g., the presence of dynamical behavior, the number of examples in the data set and the real-time constrains).
3. The *data set decomposition, optimization, and application* steps create and label the appropriate training data set to each module.
4. The *composite systems ranking* step evaluates the characteristics of all candidate composite systems by using a figure of merit provided by the designer. The maximally performing system is the first of the ranked list.

The first three steps are logically related each other since they are directed to create simple configured modules. These steps therefore constitute the *decomposition and configuration phases* of the design methodology. Although these steps are logically sequential, the design environment may need to execute some of them in parallel to achieve higher performance and quality. Figure 4 shows the detailed activities of the design methodology for this phase:

- the CSSG management,
- the module decomposition,
- the CSSG re-building,
- the CSSG simplification.

The fourth step is directed to estimate the quality of the overall results achieved by decomposition and configuration. Therefore, it can be viewed as the *evaluation phase* of the design methodology.

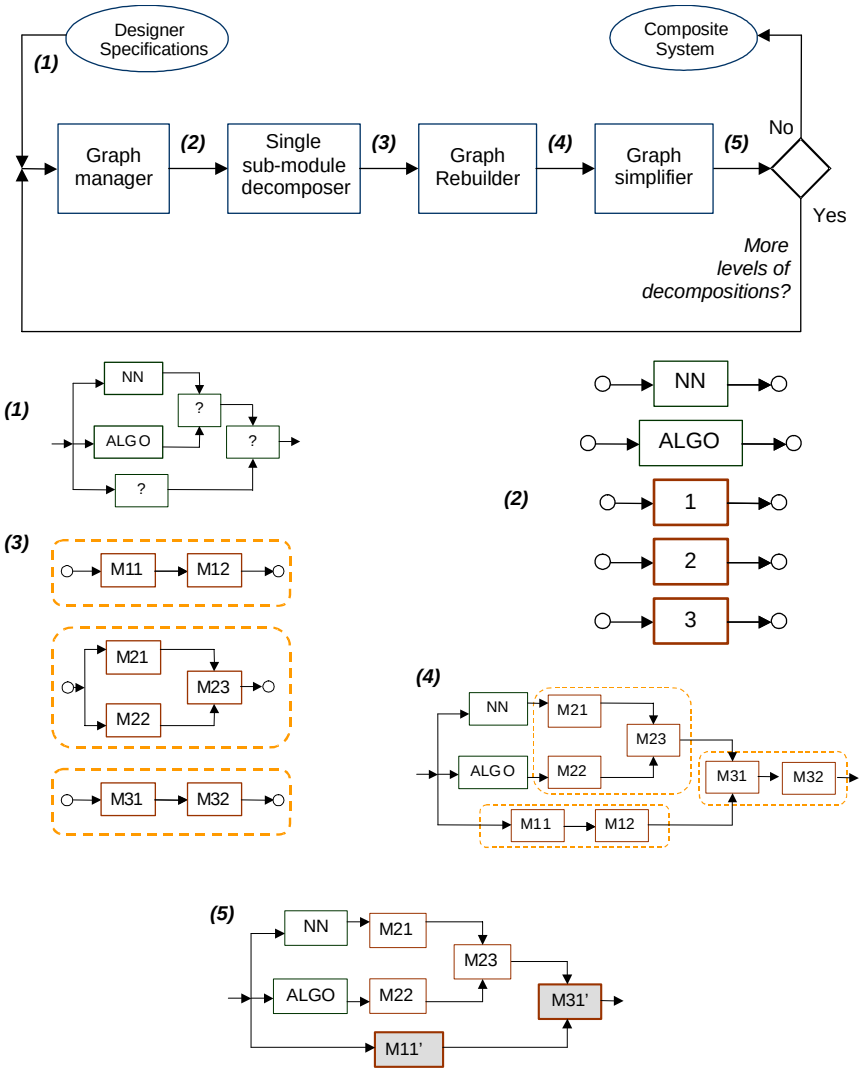

Fig. 4. The decomposition and configuration phase of the design methodology for composite systems. The activities are shown in the upper data flow graph. The CSSG at various stages are shown in the lower part.

Decomposition and configuration phase

Let's first analyze more in detail the overall activities that constitute the decomposition and configuration phase.

The *CSSG management* deals with the overall analysis and scheduling of the decomposition and configuration operations. The inputs of this task are the CSSG, the high-level specifications of the modules, and their data sets (see Figure 4, point 1). The main outputs produced by this task are the list of modules to be decomposed and their corresponding data sets.

The task consists of the following principal activities:

- selection of the modules that need to be decomposed,
- generation of the data set for the modules subject to decomposition (*module data set*),
- selection of the *decomposition modality* for each module to be decomposed.

The first task is accomplished by checking the level of abstraction of the modules and by identifying the non final ones (at the algorithm level). The following rules are applied:

- modules described at algorithmic level by the designer or characterized by models having fixed topology and parameters do not need to be further processed (they are already in the lowest description level which can be directly used in the hardware/software co-design phase);
- modules specified at the model family level or union/hierarchy of models do not need to be further decomposed, but only need to be configured and optimized;
- black-box modules must be decomposed and configured.

In the second step, the data set for training each of the decomposed modules are generated from the data sets of the original modules, possibly by simple selection of the appropriate input and output values of the examples or by more complex identification of the expected data at the input or output as they can be deduced from the available examples.

The third step focuses on the selection of the most appropriate modality to perform the topological decomposition for each of the selected modules (namely serial, parallel, or various alternatives). This is required to actually perform the single-module decomposition. If several alternatives are proposed for a module, all modalities must be verified and evaluated: we obtain a composite system for each of the decomposition modalities addressed for the envisaged module.

The *module decomposition* step receives the modules that must be decomposed and applies some decomposition rules to create new sub-modules. For each selected module, this task generates the partial CSSG describing how such module has been decomposed by applying the given rules and the data sets for the sub-modules (see Figure 4, point 3). Each of

these CSSG's must then be validated to verify whether they satisfy the application or not (see section 3.5).

The *CSSG re-building* step merges the initial composite system CSSG with the partial CSSG's of the modules that have been decomposed, thus generating a new CSSG for the whole composite system. This new CSSG has the same overall structure of the initial CSSG but contains expanded modules (see Figure 4, point 4).

The *CSSG simplification* step removes unnecessary modules that might arise during module decomposition. This "pruning" effect occurs when two subsequent modules collapse in a single one without impairing the system characteristics (see Figure 4, point 5). For example, this happens in two linear systems that are connected serially: they can collapse in a simple linear system. Similarly, the cascade of a linear system and a neural network can be collapsed into a single neural network, by suitably multiplying the weights of the input layer of the neural network by the coefficients of the linear system.

A collapsing phase requires manipulation of the data sets of the collapsed modules so as to merge them into a single data set for the new module. Besides, if the resulting module does not need further decomposition, it must be completely configured by apply the appropriate training procedure.

Further application of the decomposition and configuration phase may be performed if the modules produced by the application of the decomposition phase are not yet described in the lowest abstraction levels (namely, the algorithms or the fully configured model) or when the designer explicitly asks for that.

Modality of topological decomposition and choice of the models families

The core of the design methodology for composite systems summarized in the previous section is the module decomposition task. This task is responsible for partitioning the module into sub-modules, possibly described with different computational paradigms.

To perform this operation the topologies of the decomposition and the union/hierarchy of model families are needed. These two aspects are highly intertwined and lead to various combinations to be tested for finding the most satisfactory one (according to performance and constraints). The decomposition graph is obtained by applying some fundamental decomposition strategies to M modules such as the serial and the parallel de-

compositions. The possible modeling approaches are the ones introduced in section 2. The set of possible decompositions and related models constitute the candidate list to be tested for feasibility. To simplify creation of this list all knowledge about the application and the effective approaches should be exploited.

Figure 5.a presents the list of decomposition modalities chosen by the designer in the designing situation (*System Class*) *i*. The last two columns contain the hierarchies to be tested for each modality of decomposition proposed. For instance, in typical applications such as in classification and function approximation problems, only some decomposition modalities can be considered effective, the others being easily reducible. In this way, the designer drives the high-level methodology by specifying to which class of application the module belongs (e.g., if the module requires approximation of a static function there is no need to consider KNN classifiers or dynamic models). This will result in an automatic selection of fundamental decompositions and related models to be tested. The designer can augment these choices by adding further modeling strategies. An example is shown in Figure 5.b when the envisioned *system class i* is a classification system.

As shown in Figure 5.b, some of the proposed modalities of decomposition can be *monolithic*, *serial* and *parallel*. When a monolithic module has to be generated, all modeling strategies specified for that module must be envisaged. In other cases, we have to verify the partitioning solution to check if the partition is acceptable or can be discarded by taking into account some compatibility rules. These rules define possible topological constraints associated with the nature of the involved modeling strategies (e.g., two linear systems should not be decomposed in series or parallel, a static linear system should never be cascaded with a non-linear system) since the decomposition will not provide any advantage.

System Class: *i*		
Modality of decomposition	*Default* hierarchies of families	*Designer- proposed* hierarchies of families
Modality (1)	[M1, M2, M3]	[M7, M8, M9]
Modality (2)	[M4, M5]	[M8, M9]
Modality (3)	[M1, M2, M4]	-
Modality (4)	[M1, M3, M4]	[M7, M8, M9]
Modality (5)	[M1, M2]	[M7]

(a)

System Class: *CLASSIFIER*		
Modality of decomposition	*Default* hierarchies of families	*Designer- proposed* hierarchies of families
(1) Monolithic	[kNN, LIN, FFNN]	[LVQ, SVM, RBF]
(2) Serial, reinjection	[kNN, LIN, FFNN]	[LVQ, SVM, RBF]
(3) Serial, No-reinjection	[kNN, LIN, FFNN]	[LVQ, SVM, RBF]
(4) Parallel	[kNN, LIN, FFNN, PNN]	[LVQ, SVM, RBF]
(5) CSP	[kNN, LIN, FFNN, RBF]	-
(5) CSP	[kNN, FFNN]	-

(b)

Fig. 5. List if the modalities of decomposition and models families: (a) a generic application class, (b) a classification system.

In *serial decomposition* a module described by a parameterized model is replaced by the cascade of two modules characterized by suited parameterized models, possibly by re-injecting the primary inputs as well as the master's outputs to the slave module (see Figure 3).

When the serial decomposition is performed automatically, the training data set for the two new modules is not available and needs to be deduced from the data set $\{(I_i^M, O_i^M)\}$, characterizing the desired behavior of the decomposed module, *M*.

In the case of a model with supervised learning, the input/output pairs of the sub-modules can be computed by using the *interpolator/corrector method*. This technique consists of the following three steps. First, the master module, *M1*, is trained alone by using the input/output pairs available for the entire system: $\{(I_i^{M1}, O_i^{M1})\}=\{(I_i^M, O_i^M)\}$. According to its structural ability, this module *M1* will approximate the relationship contained in the training set. The output $\bar{O}_i^{M1}$ produced by the master module *M1* in correspondence with the input I_i^M will be the desired system output O_i^M plus an error, which depends on the data set, the adopted family, and the training method. Then, the output set $\{\bar{O}_i^{M1}\}$ of all outputs produced by module *M1* when the input set $\{I_i^M\}$ is applied is evaluated. Finally, the slave module, *M2*, is trained by using the input/output pairs $\{(I_i^{M2}, O_i^{M2})\}=\{(\bar{O}_i^{M1}, O_i^M)\}$.

A composite system trained to interpolate a strongly non-linear function with strict requirements imposed by the designer on its computational complexity is a typical example. An interesting approach envisages the serial composition of a linear sub-module and a non-linear corrector: this usually is able to better satisfy the complexity constraint. In general, this structure has a lower overall complexity when compared to a single comprehensive non-linear model for the un-decomposed "monolithic" non-linear module (in fact, the non-linear corrector has a much lower complexity than the monolithic model). The accuracy is usually identical (or sometime even better) than the monolithic model (Alippi, Casagrande *et al.* 2000, Alippi, Bertoni *et al.* 2000).

In the *parallel decomposition*, one module is replaced by N sub-modules connected in parallel, each of which receives the same inputs at the same time. Also in this case, the training data set for each sub-module must be created from the data set of the whole original module.

In the case of supervised training, the training data set can be developed by using the version of the *interpolator/corrector method* for parallel decomposition. Parallel sub-modules of the module M are trained one at time to approximate the residual error, by starting from the first sub-module for which the residual error is the whole desired function. Let's consider the training of the k-th parallel sub-module, Mk, k=1,2,…,N. Sub-module Mk is trained by using the training data set $\{(I_i^{M,k},O_i^{M,k})\}$ equal to the training data set, $\{(I_i^M,O_i^M)\}$, of the whole module if $k=1$, or the one with the residual output error, $\{(I_i^M,\ O_i^M - \Sigma_{j=1,k-1}\ \bar{O}_i^{M,j})\}$, for $1<k\leq N$.

In *serial-parallel decomposition*, module M is decomposed in a cascade of parallel sub-modules (*M1.1*, *M1.2*, …, *M1.N*) and a sub-module *M2*, possibly with re-injected primary inputs I_i^M. The set of sub-modules *M1.k* connected in parallel are globally trained as described for the master sub-module of the serial decomposition, by using the global data set $\{(I_i^M,O_i^M)\}$. Training each sub-module *M1.k* proceeds as described for the parallel decomposition. Then, global residual error generated by the set of parallel sub-modules is evaluated as $e_i^{M1} = O_i^M - \Sigma_{k=1,N}\ \bar{O}_i^{M,k}$. Finally, the slave module *M2* is trained to interpolate the desired output function as corrector by using the outputs that have been pre-processed by the parallel sub-modules (i.e., the residual errors e_i^{M1}) and, possibly, the primary inputs, I_i^M, of the module.

Data set optimization

When the data set has been generated for a module that needs to be configured, an optimization phase can help in restructuring such a data set so that accuracy, complexity, and robustness performance will be achieved by tuning the model parameters.

Various techniques can be used to achieve different optimization goals. For each optimization strategy a specific data set can be generated and used in the subsequent module synthesis. The overall scheme of this phase is shown in Figure 6. The techniques here considered are:

- the feature extraction and selection,
- the condensing dataset techniques,
- the conventional pre-processing.

The *feature extraction* step aims at extracting features from the input signals and creating additional features based on transformations or combinations of the original ones. In the latter case, we can envision principal component analysis (Zien *et al.* 2000), non-linear principal component analysis (Zien *et al.* 2000, Jones *et al.* 1992), neural networks (Jones *et al.* 1992), self-organizing maps (Hassoun, 1995), independent component analysis (Alippi 1999, Almeida *et al.* 2000, Amati *et al.* 1996, Amari *et al.* 1999), and multi-dimensional scaling (Mika *et al.* 1999).

Feature selection techniques are directed to select the input features subset that allows the model to achieve the highest accuracy. Among these approaches, we have exhaustive search (Oliveira and Sangiovanni-Vincentelli 1992), sequential forward selection (Pudil *et al.* 1994, Ripley 1996) "plus l-take away r" (Ripley 1996), and mutual information feature selection (Kwak and Choi 2002).

The computational complexity associated with the training phase for many classifiers is proportional to the number of samples contained in the training data set. To overcome this problem, *condensed dataset* techniques have been proposed in the literature to eliminate "redundant" samples from the data set in order to reduce its cardinality. Commonly used methods are: Hart algorithm (Kim *et al.* 1993), Hart and restart (Kim *et al.* 1994), mutual neighborhood value (Kuncheva and Jain 2000) and genetically optimized data set (Casillas *et al.* 2001).

Conventional pre-processing consists of, e.g., normalization, scaling, and re-mapping techniques.

Figure 6 shows how the composition of the presented methods can produce different datasets (DS_1, DS_2, …, DS_N) starting from the initial I/O pairs. All the produced datasets can be used to differently train the pro-

posed hierarchies of models producing models with different accuracies and complexities.

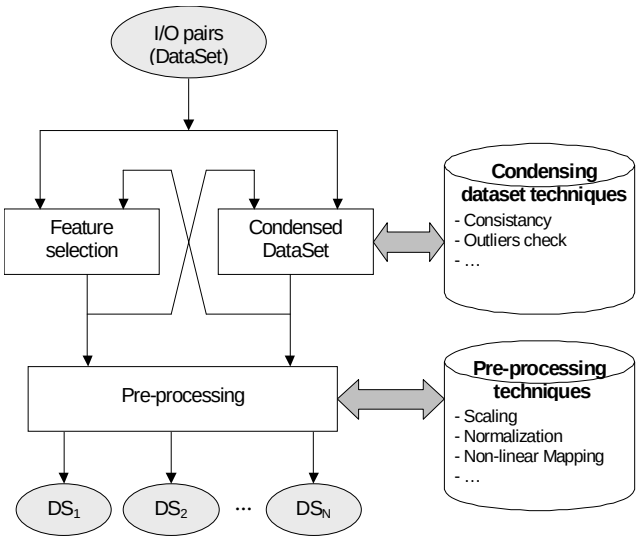

Fig. 6. Optimization of the training data set. Feature selection, feature extraction, condensed dataset analysis, and conventional pre-processing can be used. DS1, DS2, … , DSN are the various optimized data sets.

Module synthesis

When the model and the data set for a non-algorithmic module have been identified the final synthesis is directed to configure the model and generate the algorithmic representation, by taking into account the module specification and the alternative data sets produced by the optimization phase (see Figure 7) where the modules present in Figure 6 are enclosed into the pre-processing module in Figure 7. The most suited model according to the non-functional specifications will then be selected.

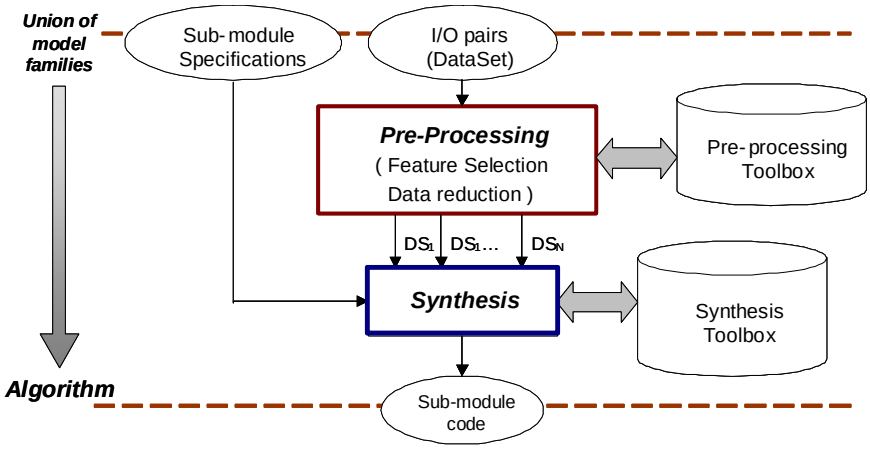

Fig. 7. The module synthesis in the methodology perspective.

The module synthesis of Figure 7 consists of the phases depicted in Figure 8 which transform the high-level description given at a black-box abstraction level to an algorithm level description. More in detail, we have:

- the topology processing phase that identifies the specific model to be adopted,
- the training phase that configures the model,
- the code translation phase that produces the algorithmic description in the standard representation to be used in hardware/software co-design.

Modules specified at intermediate levels can directly enter in the topology processing chain at their appropriate level.

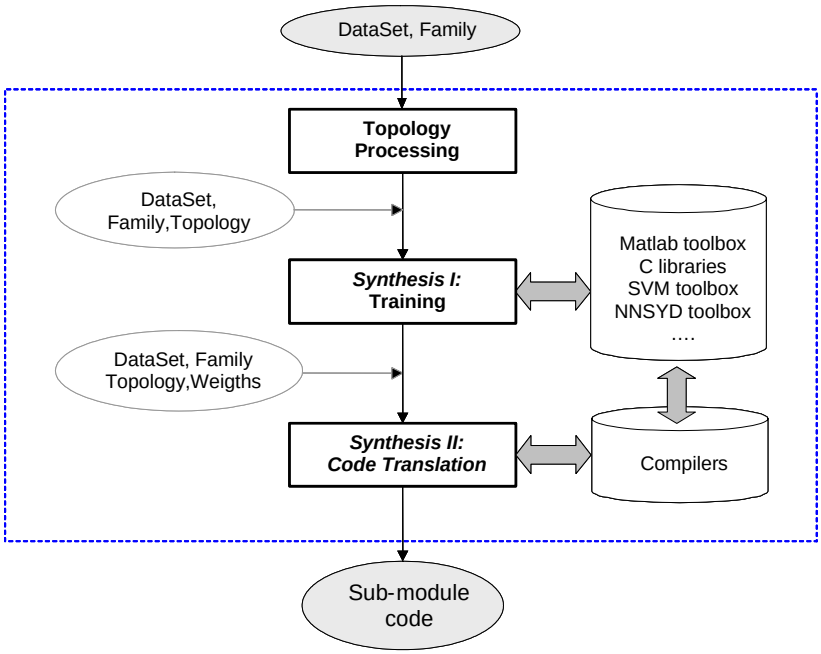

Fig. 8. Phases of the module synthesis

The *topology processing* phase selects the most suited family of models and the most suited model within the family, by starting from the hierarchy/union of model families. To efficiently and effectively address this task many theoretical issues must be considered, such as the number of degrees of freedom of the model with respect to the size of the training dataset and the duration of training.

To explore the various alternatives, a list of families to be experimented is produced. For example, we can consider a neural network union of models with a single hidden layer and N_1 to N_2 hidden neurons.

The list of families can be created by using a greedy approach: all combinations of the values of the parameters that characterize the hierarchy/union are considered and the related families are introduced in the list.

The *training phase* configures the parameters for each model generated in the previously mentioned list. The topology and the family are fixed. The training algorithm is one of those specified for the envisioned family in the design environment. Values of optional parameters can be specified to tune the behavior of the configuration algorithms, such as the regularization methods, the number of epochs, and the final training error.

Once the model has been completely configured, the corresponding model description that can be directly simulated in the design environment is available. This is a function that describes the model computation and receives as input parameters the actual values computed during the configuration phase.

To perform the subsequent hardware/software co-design synthesis, the algorithmic description of the model computation is needed. The last phase of the paradigm synthesis is therefore the *code generation*, which delivers the algorithm – in a high-level hw/sw description language – computing the outputs of the configured model.

Evaluation phase

The configured models produced by the module synthesis for each module must be tested to verify their accuracy, robustness, and complexity. This allows for excluding the solutions that do not satisfy the application constraints from the candidates to be considered to build the composite system. This can be done automatically by applying the appropriate evaluation functions associated with the hierarchy/union of model families adopted for each module. If more than one solution satisfies the constraints, the designer will be allowed to select the preferred one based on their cost, performance or accuracy.

Configuration and expandability of the design environment

To manage the application of the design steps in an ordered and standard way we designed the environment according to a modular philosophy based on the *table of symbols* and the related definitions for each hierarchy/union of models families. This allows the user for expanding the design environment by adding new hierarchies/unions of family models, new training and simulation functions. The overall structure of this table is shown in Figure 9. Adding a new hierarchy of families simply implies that a new column is inserted in the table. Conversely, adding new design constraints is obtained by adding a new row. This makes the environment easily expandable.

In particular, the list $\theta\{\ldots\}$ contains the list of parameters required to specify the model of the Γ hierarchy/union. For the given Γ and the dataset, the *topologicalproc()* function returns a list of lists $\theta\{\ldots\}$ to be used for the training. The trained model is obtained by invoking the $F^{\Gamma}_{train}()$

function by using as parameters $\theta\{\ldots\}$, the training-option list Opt^{Γ} and the dataset. The function $Fsim^{\Gamma}()$ produces the output of the system when invoked with the trained model and the model's input matrix. The function $Finj^{\Gamma}()$ allows the system for estimating the accuracy variation of the envisaged model when a specified perturbation of the computation is applied to it. Function $Robustness^{\Gamma}()$ measures the robustness of the considered model and uses $Finj^{\Gamma}()$. Function $Accuracy^{\Gamma}()$ returns the value of the accuracy of the model by receiving the given dataset. $Complexity^{\Gamma}()$ returns the complexity (in elementary computational units) of the envisaged model.

Symbol	***Hierarchy/Union 1***	***Hierarchy/Union 2***	…
Γ	*FF-NN*	*kNN*	…
θ{1} θ{2} θ{3} θ{4} …	Layer1: n_{r1} Layer1:*ActivFunction*$_1$ Layer2: n_{r2} Layer2:*ActivFunction*$_2$ …	k Norm - - …	… … … … …
U	hierarchy	union	⋯
TopologicalProc (·)	topologyFF (·)	topologyKNN (·)	⋯
$F^{\Gamma}{}_{train}(\cdot)$	trainFF (·)	trainKNN (·)	⋯
Opt^{Γ}	{'BayesRegulariz', 'Epochs', … }	-	⋯
$Fsim^{\Gamma}(\cdot)$	simFF (·)	simKNN (·)	
$Finj^{\Gamma}(\cdot)$	injectFF (·)	injectKNN (·)	⋯
$Accuracy^{\Gamma}(\cdot)$	accuracyFF (·)	accuracyKNN (·)	⋯
$Complexity^{\Gamma}(\cdot)$	complexityFF (·)	complexityKNN (·)	⋯
$Robustness^{\Gamma}(\cdot)$	robustnessFF (·)	robustnessKNN (·)	⋯

Fig. 9. Table of symbols for the modular configuration of the design environment.

7.4 Application of the high-level design approach and experimental results

To show the effectiveness and the usability of the proposed high-level design methodology for composite systems we present two examples. The first one is a real industrial application developed to monitor of the railway track profile wearing (Alippi, Casagrande *et al.* 2000). The application is based on a three-dimensional laser scanning and a track profile reconstruction. The second example concerns the design of a dynamic system by us-

ing the NNSYSID data set (Norgaard 1999), by focusing on the identification of the best trade-off among accuracy, complexity, and robustness. Results achieved by our methodology have shown to perform at least as those solutions created by experienced designers.

A monitoring system for detection of railway track profile wear

Checking the status of railway tracks is critical to ensure high operating safety, proper maintenance schedule, and low maintenance and operating costs. This operation consists of the analysis of the track profile, level, overall geometry, and waving profile. Traditional detection systems are based on mechanical tactile devices in contact with the track. In the TRACKS project funded by the European Union, an innovative contactless approach was developed by using a laser scanning and real-time image analysis for profile identification and reconstruction (figure 10).

The monitoring system consists of a laser source, whose beam is collimated by a suited optic lens into a light plane, two 512x512-pixel CCD cameras for complete optimal observation of one track, a digital processing system per camera, and a supervision system (figure 10a). Each digital processing system performs real-time profile filtering and extraction from images of the corresponding CCD camera. Hard real-time operation is needed since 200 track sections per second must be captured and processed to guarantee a sufficient accuracy of deformation localization at a normal convoy speed up to 180 km/h; in this condition, the image acquisition system generates at least 1.7 Gbit/s.

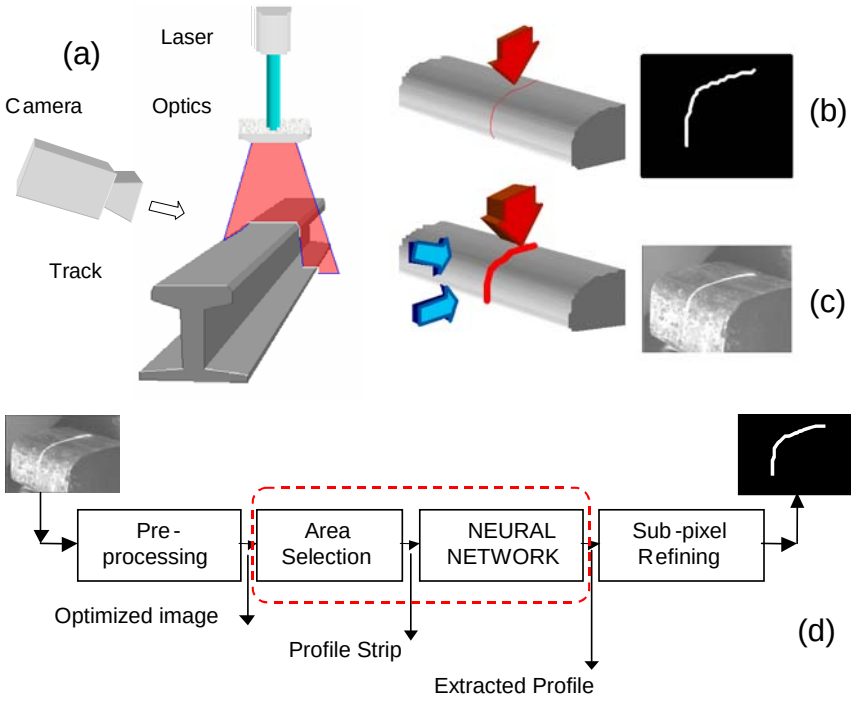

Fig. 10. A laser-based monitoring system for railway track wear: (a) the monitoring system, (b) the ideal track image, (c) the measured track image, (d) the composite system that extracts the track profile from the real images.

Accurate profile identification is a difficult task because of the presence of noise and environmental disturbances that modify the ideal profile reflection as observed by the camera (figure 10b). Real images (figure 10c) are affected by environmental light, multiple reflections, track oxidation, grease on the track, speckle effect due to track roughness, non-infinitesimal thickness of the laser plane scanning the track, optic aberrations, CCD sensor saturation, and image distortions due to vibrations. Due to the geometry of the monitoring system, the track profile appears only once in each column of the image. The position of the track profile in the image is identified by determining where the laser reflection has maximum intensity. In the ideal case the laser intensity distribution along the column is Gaussian. Localizing the maximum means therefore detecting – with the maximum likelihood – the position of the Gaussian profile and, consequently, the position of the maximum.

The composite processing system is shown in Figure 10d. The input image is first contrast balanced by the preprocessing module. To speed up the

system operation, the image strip containing the profile is selected by using an algorithmic approach and the remaining portions of the image are discarded. A neural network then interpolates the laser reflex within each image column of the small strip and accurately localizes its maximum. The global interpolation finally reconstructs the track profile by a using all the estimated maximum positions.

Let's assume that the designer whishes to optimize the selection of the area of interest, the extractor of the profile strip, and the neural interpolator (namely, the modules circled in Figure 10d).

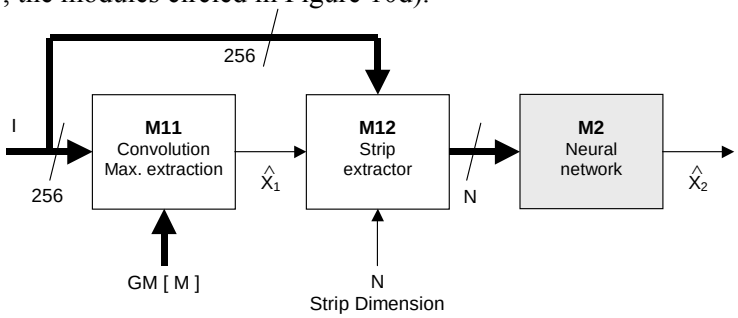

Fig. 11. The composite system sequencing graph of the TRACKS project

Application of our design methodology leads the designer to specify the sequencing graph shown in Figure 11, and the related parameters to be optimized. The selection of the area of interest in the CCD image is achieved by performing the following operations. The column intensity vector, I, with size fixed by the CCD dimension, is convolved with the Gaussian mask, GM; this mask has order M and must be optimized. Then, the position of the laser reflection, X_1, is roughly estimated by extracting the location of the maximum in the convolution $I \bullet GM$. In each column, I, of the image, the extractor finally identifies the strip of interest, S, having size N and centered in X_1. The accurate localization of the maximum reflex, X_2, is performed by the neural interpolator, whose topology, T, and weights, Θ_{NN}, have to be determined.

For the envisioned application, the designer sets the accuracy and the computational complexity requirements. The profile reconstruction error, E_1, must be lower than 0.27 pixels; this ensures that the accuracy on the actual track profile is 20μm, by taking in account the camera zoom. The computational load, C_1, must be lower than 10 elementary time units per pixel (see section 2.4), according to the target hardware architecture and the requested frame rate of 10 frames/sec. Besides, let's assume that the designer is interested in creating the most accurate composite systems as

possible. The composite system synthesis should therefore look for the solution that satisfies the above constraints and maximizes the accuracy.

The behavioral description of the systems is given by means of the training data set, Z_{Ntrain}, of examples that consists of images in all working condition, including rusty tracks, external reflections, laser out of focus, vibrations, and overnight condition; the validation data set, Z_{Nval}, is used to verify the quality of the solution. Each of these data sets contains 2,000 columns.

The CSSG adopted by the designer (figure 11) specifies the composite system as a serial decomposition in which a parameterized-functional module, *M1*, is followed by a black-box module, *M2*. Let's assume that the designer is interested to test the use of Radial Basis Function neural networks in implementing the system as a monolithic module in addition to the default hierarchies. This can be specified by properly setting the table of symbols as shown in Figure 12 in the modalities of decomposition table (upper left table).

The table of names for the TRACKS project is presented in Figure 12 (upper right) according to the term description given in Figure 9. The topological processing table in Figure 12 shows the lists of module parameters $\theta\{\ldots\}$ produced by invoking the function *topologyM1()* and *topologyRBF()*. These parameters are used to the modules [Mi, Mj] of the composite systems according to the topology specified into the last table in Figure 12.

System Class: *APPROXIMATION SYSTEM* ***Table of hierarchies***		
Modality of decomposition	*Default* hierarchies of families	*Designer- proposed* hierarchies of families
(1) Monolithic	[FFNN]	[RBF]

Table of names for the TRACKS project		
Symbol	***Hierarchy/Union 1***	***Hierarchy/Union 2***
Γ	*M1*	*RBF*
θ{1} θ{2}	Gaussian_Mask_Size: M Strip_Size: N	Neurons: n_r Spread: σ
U	hierarchy	union
TopologicalProc(·)	topologyM1(·)	topologyRBF (·)
$F^{\Gamma}_{train}(\cdot)$	train M1 (·)	trainRBF (·)
Opt^{Γ}	{'MaxEpoch'}	-
$Fsim^{\Gamma}(\cdot)$	simM1 (·)	simRBF (·)
$Finj^{\Gamma}(\cdot)$	injectM1 (·)	injectRBF (·)
$Accuracy^{\Gamma}(\cdot)$	accuracyM1 (·)	accuracyRBF (·)
$Complexity^{\Gamma}(\cdot)$	complexityM1 (·)	complexityRBF (·)
$Robustness^{\Gamma}(\cdot)$	robustnessM1 (·)	robustnessRBF (·)

Topological Processing for the TRACKS project			
Index	Code	*Family*	[θ{1}, θ{2}, θ{3}, …]
1	M1	*M1*	[3, 10]
2	M2	*M1*	[3, 15]
3	M3	*M1*	[5, 15]
4	M4	*M1*	[5, 20]
…	…	…	…
5	M5	RBF	[3, 5, 'hyperbolic']
6	M6	RBF	[3, 50, 'hyperbolic']
7	M7	RBF	[5, 10, 'hyperbolic']
…	…	…	…

Composite Systems to be trained for the TRACKS project			
Index	Modality	*Code1st module*	*Code2nd module*
1	(1) Monolithic	M1	M5
2	(1) Monolithic	M1	M6
3	(1) Monolithic	M1	M7
…	…	…	…
5	(1) Monolithic	M2	M5
6	(1) Monolithic	M2	M6
…	…	…	…

Fig. 12. The table of symbols characterizing the behavior for the environment.

The design environment supports the exploration of the solution space according to our design methodology. The topologies and the combination of families selected by the methodology are shown in Figure 13, with respect to the two figures of merit to be optimized (accuracy and complexity). The best composite system is characterized by M=5, N=9, and n_r=5 (number of hidden neurons), and σ=10 (variance of the RBF Gaussian). The computational complexity is about 7 elementary time units per pixel and the mean error is 0.06 pixel.

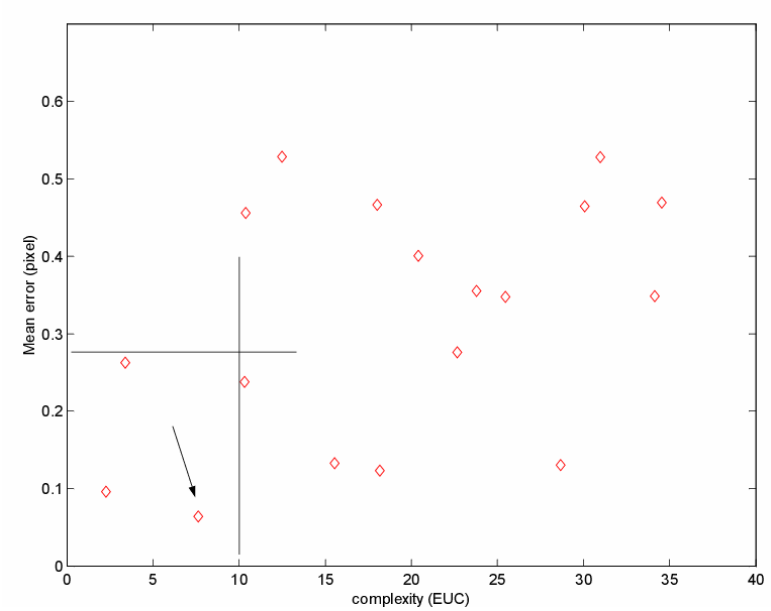

Fig. 13. The solution space of the TRACKS project given in the two quantities that must be optimized (the mean error versus complexity). The arrow indicates the most accurate solution that satisfies the constraints on maximum complexity and error.

Implementation of a dynamic system

To show the effectiveness of our design methodology also for dynamic systems we considered the application described by the NNSYSID data set and consisting in the identification of the open-loop stable, non-linear, continuous system (Norgaard 1999).

System Class: *DYNAMIC SYSTEM* *Table of hierarchies*		
Modality of decomposition	*Default* hierarchies of families	*Designer-proposed* hierarchies of families
(1) Monolithic	[ARX, NOE]	-
(2) Serial reinjection	[ARX, NOE]	-
(3) Serial No-reinjection	[ARX, NOE]	-

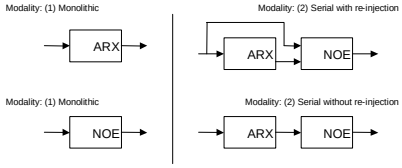

Table of names dynamic systems		
Symbol	***Hierarchy/Union 1***	***Hierarchy/Union 2***
Γ	*NOE*	*ARX*
$\theta\{1\}$	? regressor: n_a	? regressor: n_a
$\theta\{2\}$	x regressor: n_b	x regressor: n_b
$\theta\{3\}$	delay time: n_k	delay time: n_k
$\theta\{4\}$	Layer1: n_r	-
$\theta\{5\}$	Layer1:*ActivFunction*$_1$	-
$\theta\{6\}$	Layer1: 1	-
$\theta\{7\}$	Layer1:*Linear*	-
U	hierarchy	hierarchy
TopologicalProc(·)	topologyNOE (·)	topologyARX (·)
$F^{\Gamma}_{train}(\cdot)$	trainNOE (·)	trainARX (·)
Opt^{Γ}	{'SmallWeightDecay', 'MaxEpoch'}	-
$Fsim^{\Gamma}(\cdot)$	simNOE (·)	simARX (·)
$Finj^{\Gamma}(\cdot)$	injectNOE (·)	injectARX (·)
$Accuracy^{\Gamma}(\cdot)$	accuracyNOE (·)	accuracyARX (·)
$Complexity^{\Gamma}(\cdot)$	complexityNOE (·)	complexityARX (·)
$Robustness^{\Gamma}(\cdot)$	robustnessNOE (·)	robustnessARX (·)

Topological Processing			
Index	Code	*Family*	[θ{1}, θ{2}, θ{3}, …]
1	M1	*M1*	[3, 3, 1]
2	M2	*M1*	[4, 3, 1]
3	M3	*M1*	[4, 4, 1]
4	M4	*M1*	[5, 4, 1]
5	M5	RBF	[3, 3, 1, 3, 'hyperbolic']
6	M6	RBF	[3, 3, 1, 4, 'hyperbolic']
7	M7	RBF	[3, 3, 1, 5, 'hyperbolic']
…	…	…	…

Composite Systems to be trained			
Index	Modality	*Code1st module*	*Code2nd module*
1	(1) Monolithic	M1	-
2	(1) Monolithic	M1	-
3	(1) Monolithic	M1	-
…	…	…	…
7	(1) Monolithic	M7	-
8	(2) Serial	M1	M3
9	(2) Serial	M1	M4
10	(2) Serial	M2	M3
11	(2) Serial	M2	M4
…	…	…	…

Fig. 14. The designer specification for the standard dynamic system defined by the NNSYSID data set. (a) The modality of decomposition: from the black box to four serial configurations. (b) The table of symbols for the design environment.

The summary of the designer specifications is given in Figure 14. In particular, the designer asked for evaluating the accuracy by imposing an upper bound, *E1* to the mean squared error.

For complexity issues the designer specifies his own evaluation method by means of the function *User_Defined_Complexity*() and sets the upper bound, *C1*. For each candidate satisfying the requirements on accuracy and

complexity, the designer desired to look for the most robust composite system.

No structure was suggested for the composite system: as such , the black-box approach has been adopted. Since no value was specified in the third column of the table of symbols the default hierarchies of models families were used: the Auto-Regressive with eXternal input models (ARX) and the Neural network Output Error model (NOE) [12,13]. The modalities of decomposition defined by the designer led to generate four configurations of the composite system.

To select the best solution the candidates are plotted in the accuracy vs. complexity plane as shown in Figure 15. The monolithic ARX model created with different topologies does not provide sufficient accuracy. The more accurate solution is a composite system composed by an ARX module followed by a NOE module with re-injection (circle #2). The others plotted points represent systems balancing the two figures of merit and satisfying the designer constraints (less accurate but less complex solutions). Circle #1 enlightens the less complex model generated.

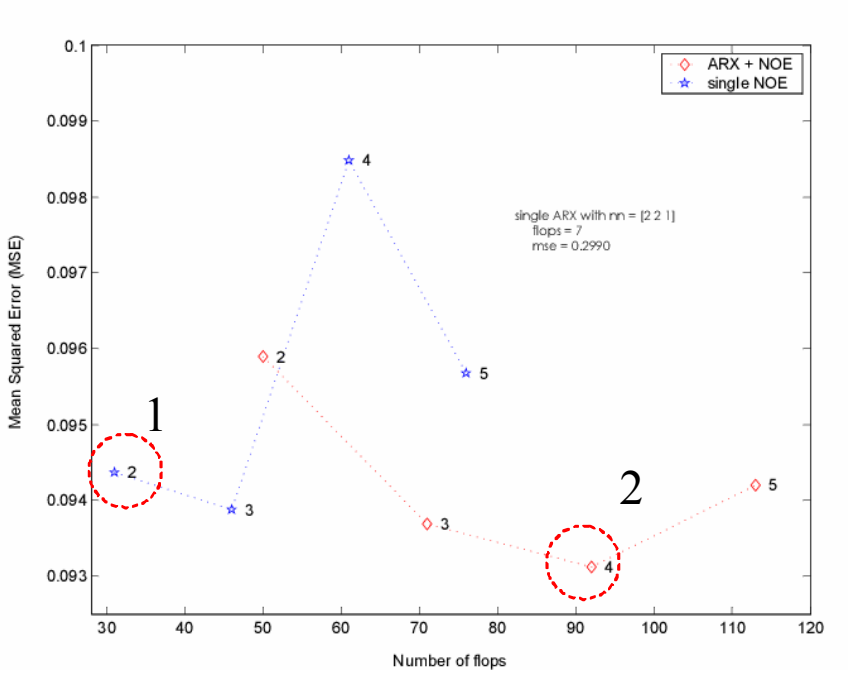

Fig. 15. Selecting the composite system by using the designer constrains. (a) NOE solutions are indicated with stars: the less complex one is identified by circle #1. (b) Diamonds refer to composite systems composed of an ARX module followed by a NOE module: the best solution is identified by circle #2.

7.5 Conclusion

In this chapter we presented an innovative methodological approach to the high-level design of composite systems, encompassing algorithmic and neural network modules. The whole system is first specified at a high-abstraction level by using description languages suited to characterize the various aspects of the system itself. Then, a design methodology performs the computational paradigm co-design, i.e., partitions the overall description into modules with various computational paradigms and configures each of them. Non-functional specifications are used to guide this partitioning process. The result is an algorithmic description of the whole system. This description can be directly entered in the conventional methodologies for hardware/software co-design in order to produce the final implementation encompassing dedicated and programmable parts as well as the possible related software. The proposed methodology has been im-

plemented into a prototypal environment for classification and dynamic systems by using the *Mathworks Matlab* language. The prototypal environment integrates the hierarchies of models and their relative training and validation functions present into the *Matlab toolboxes* and integrates external toolboxes such as kNN classifiers and SVMs. In addition, in the environment are available some of the most common methods to pre-process the dataset (feature selection, and condensation, linear and non-linear mapping). The use of this methodology has been shown effective and efficient in several real application cases.

References

Alippi, C., (1999), "FPE-based Criteria to Dimension Feedforward Neural Networks," *IEEE-Transactions on Circuits and Systems: Part I, Fundamental theory and applications*, vol. 46, no. 8, pp. 962–973.

Alippi, C. (2002), "Randomised algorithms: A system-level, poly-time analysis of robust computation," *IEEE-Transactions on Computers*, vol. 51.

Alippi, C., Bertoni, G.M., Diani, G.M., Piuri, V. and Scotti, F. (2000), "A Composite System Design Methodology for Instrumentation and Embedded System," in *Proc. IEEE Instrumentation and Measurement Technology Conference, Baltimore, MD, USA*. IEEE, pp. 1086–1089.

Alippi, C., Casagrande, E., Piuri, V. and Scotti, F., (2000), "Composite Real-Time Image Processing for Railways Track Profile Measurement," *IEEE Transactions on Instrumentation and Measurement*, vol. 49, pp. 559–564.

Alippi, C., Ferrari, S., Piuri, V., Sami, M., and Scotti, F. (1999) "New Trends in Intelligent System Design for Embedded and Measurement Applications," *IEEE Instrumentation and Measurement Magazine*, vol. 2, pp. 36–44.

Almeida, L. (2000), "Linear and nonlinear ICA based on mutual information," in *Proc. IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (AS-SPCC)*, Lake Louise, Canada, pp. 117–122.

Amari, S.I., Cichocki, A. and Yang, H.H. (1996), "A new learning algorithm for blind source separation," in *Advances in Neural Information Processing Systems 8*, pp. 757–763. MIT Press.

Amari, S.I. (1999), "Natural gradient learning for over- and undercomplete bases in ICA," *Neural Computation*, vol. 11, no. 8, pp. 1875–1883.

Ashenden, P. (2002), *The Designer's Guide to VHDL,* Morgan Kaufmann.

Casillas, J., Cordan, O., Del Jesus, M. J., Herrera, F. (2001), "Genetic feature selection in a fuzzy rule-based classification system learning process for high-dimensional problems", *Information Sciences*, vol.136, pages 135-157.

De Micheli, G., and Sami, M.G. (1995), *Hardware/Software Codesign*, Kluwer Academic Publishers.

Gajski, D., Dutt, N., Wu, A. and Lin, S. (1992), *High-level Synthesis: introduction to Chip and System Design*, Kluwer Academic Publishers.

Gajski, D., Vahid, F., Narayan, S. and Gong, J. (1995), *Specification and design of Embedded Systems*, Prentice Hall.

Hassoun, M.H. (1995), *Fundametals of Artificial Neural Networks*, The MIT press.

Higleyman, W. H. (1962), "The design and analysis of pattern recognition experiments," *Bell System Technical Journal*, vol. 41, pp. 723–744.

Hilfinger, P. and Rabaey, J. (1992), *DSP Specification Using the Silage Language*, Kluwer Academic Publishers.

Jones, M.J. (1992), "Using recurrent networks for dimensionality reduction," Tech. Rep. AITR-1396, MIT.

Kim, B.S., Lee, S. H. and Kim, D. K. (1003), "Determination of initial configuration for LVQ by usign CNN," in *Proceedings of 1993 International Joint Conference on Neural Networks*. IEEE.

Kwak, N. and Choi, C.H. (2002), "Input feature selection for classification problems," *IEEE-Transactions on Neural Networks*, vol. 13, no. 1, pp. 143–159.

Kuncheva, L. I. and Jain, L. C. (2000), "Designing classifier fusion systems by genetic algorithms," *IEEE-EC*, vol. 4, no. 4.

McLachlan, G.,(1976) "The bias of the apparent error rate in discriminant analysis," *Biometrika*, vol. 63, pp. 239–244.

Mika, S., Ratsch, G., Scholkopf, B., Smola, A., Weston, J. and Muller, K.-R. (1999), "Invariant feature extraction and classification in kernel spaces," in *Advances in Neural Information Processing Systems*, Cambridge, MA, MIT Press.

Newton, A. R., Rabaey, J., Keutzer, K., Malik, S. and Sangiovanni-Vincentelli, A. (2000), "System level design: Orthogonolization of concerns and platform-based design," *IEEE Transactions on Computer- Aided Design of Integrated Circuits and Systems*, vol. 19, no. 12.

Norgaard, M. (1999), "Neural network based system identification toolbox," Tech. Report. 95-E-773, Institute of Automation, Technical University of Denmark.

Oliveira, A. L. and Sangiovanni-Vincentelli, A., (1992), "Constructive induction using a non-greedy strategy for feature selection," in *Proceedings of the Ninth International Conference in Machine Learning*, Aberdeen, Scotland, pp. 355–360, Morgan Kaufmann.

Perry, D. L. (1991), *VHDL*, McGraw-Hill, New York.

Pudil, P., Novovicova, J. and Kittler, J. (1994), "Floating search methods in feature-selection," *PRL*, vol. 15, no. 11, pp. 1119–1125.

Ripley, S. (1996), "Pattern recognition and neural networks", in *Statistics Cambridge: University Press*, pp. 1065–1076.

Vapnik,V. (1998), *Statistical Learning Theory*, Wiley.

Zien, A., Ratsch, G., Mika, S., Scholkopf, B., Lengauer, T. and Muller, K.R. (2000), "Engineering support vector machine kernels that recognize translation initiation sites," *BioInformatics*.