A Comparative Evaluation of Models and Specification Languages for Embedded System Design♠

Ioannis Panagopoulos[1], George Papakonstantinou[1] and Nikitas Alexandridis[2]

[1] Computer Systems Laboratory, School of Electrical and Computer Engineering
National Technical University of Athens, 11633 Athens, Greece
Email: ioannis@cslab.ece.ntua.gr
[2] Department of Electrical and Computer Engineering, George Washington University, Washington DC, 20052 USA

Abstract. The HW/SW Codesign approach in the design of embedded systems and their increasing complexity has turned the need of simplifying the specification of both the desired behaviour and the final implementation a very crucial task in the design process. Towards this effort programmers and designers have introduced a plethora of model-specification language pairs that can effectively reduce the complexity of the captured functionality, raise the level of abstraction and support several design tasks (such as verification, performance estimation etc). Due to the lack of a set of well-defined features for the comparative evaluation of those pairs industry is still reluctant in taking advantage of their full potential. Additionally, design teams are introducing new models and specification languages which sometimes add nothing new to already existing ones. Although VSIA' s [13] model taxonomy has been proven very useful in classifying models for the specification of the system's implementation our approach deals with the specification of the initial desired behaviour. By introducing two evaluation axes for models, introducing a 3 dimensional space for the taxonomy of specification languages and classifying the latter with respect to the model they are used to express, we present a complete feature-based approach that can be used not only for the selection of the most appropriate model-specification language pair for the design at hand but also for the classification and evaluation of new models and specification languages introduced for embedded system design.

1 Introduction

System Level Design is the process through which a behavioral specification is captured at a high level of abstraction and is progressively transformed (through various transformation tasks) to a structural implementation of a desired system [1, 2]. Since a specific behavior can be transformed to a very large number of possible implementations, a set of constraints is provided by the designer to restrict his/her search in only a subspace of potential target candidates. The whole process, from expressing the behavioral specification and the constraints to the implementation of the desired system is referred to as a "*design cycle*".

♠ This work was partially supported by a grant from the NSA/Lucite's program "IP Tool"

A design cycle can be carried out using various "*design methodologies*". Each one of them defines a specific model for capturing the desired behavior, several transformation steps towards the final implementation that can either be performed automatically or with user interaction and a specific model to express the final implementation. Today several design methodologies exist [3, 4, 5, 6, 7, 8, 9] claiming to automate or facilitate the fulfillment of various tasks of the design process. Each one of them follows a different approach, by providing different models-specification languages for expressing the desired behavior, different methods for estimating the target system's features (under various design constraints) and different transformation steps (manual or automatic) that can lead to a set of potential target implementations. Behavioral and structural specifications are defined through models and expressed using specification languages. Transformations in terms of mapping one model to another or moving towards a lower level of abstraction are performed automatically or with user-interaction to present design alternatives. Then, according to the resulting modeled system's features a selection is performed on the most suitable one to be implemented.

Due to the lack of a set of well defined features for the evaluation of existing models-specification languages and selection of the most appropriate pair for the design at hand, some of those methodologies tend to use models which are either completely new introduced by the methodologies themselves, or existing ones, more suitable for expressing the final implementation's structure rather than the initial desired behavior. This in most cases leads to situations where several design methodologies are using (in several steps of the design process from capturing the behavior of the system to mapping it to the final implementation) different specification languages for the same underlying models in a fashion that seems like "reinventing the wheel". For example in [7, 9] two C++ clones are proposed (SystemC and BachC) which are both used to express an object-oriented model without any clear differentiation of the characteristics and advantages of using the one as opposed to the other. In other cases new models are presented that are adding nothing new to the older ones (in a semantic aspect) apart from an easier visualization of the final implementation (i.e. the behavioral layer of Verilog [10] or VHDL [11] is based on the same structural model that the SpecC [12] language is using without any clear explanation on why one of the three should be preferred). This paper deals with the role of models and specification languages in the System Level Design process. More specifically, it complements the VSIA's taxonomy on models [13] by introducing two evaluation axes for the classification and evaluation of models for the behavioral specification and categorizes specification languages according to their features and the model they are used to express. The complete feature-based system presented can either be used for the selection of the most appropriate model-specification language pair, depending on the design at hand, or facilitate the classification and evaluation of new pairs introduced in the future.

The rest of the paper is organized as follows. First we present the model taxonomy proposed by VSIA [13]. Then we extend that taxonomy with two additional axes to capture additional but important features of models for capturing the desired behavior. The resulting classification of models is presented in Sect. 4. In Sect. 5 we present the three axes used for the classification of specification languages and their relation to models. Finally using the introduced comparative evaluation we present a case study which illustrates the usefulness of the existence of such a feature-based evaluation.

2 The VSIA's Model Taxonomy

Today several models exist [14, 15], each one with its own expressive power and useful characteristics in specifying systems. We have surveyed 18 of them *Finite State machines (FSM)* [16] ,*Hierarchical Concurrent Finite State Machines (HCFSM)* [17, 18], *Finite State Machines with DataPath (FSMD)* [19] , *Codesign Finite State Machines (CFSM)* [20], *Heterogeneous finite state machines* [18], *Petri-Nets* ,*Timed Petri-Nets, Time Petri-Nets, Timed place transition net,* [21, 22], *Data Flow Graphs, Control-Data Flow Graphs* [23], *Synchronous Data-Flow Family* [24], *Cyclo-static data-Flow models (CSDF)* [25], *FlowCharts* [26], *Structure oriented models* [10, 11, 12], *Heterogeneous process oriented models, CSP* [27],*Object Oriented models* (UML) [28] which we believe can summarize the descriptive power offered by any model existing today.

In the effort of classifying and evaluating those models we have adopted the axis based taxonomy proposed by VSIA [13], extending it by two additional axis to capture two additional characteristics of models, that of expressive power and conceptual form of their building blocks. VSIA [13] has introduced a taxonomy based on the level of the abstraction each model allows for the expressed behavior. This abstraction level has been presented by the use of five axes whose resolution is based on the level of detail in capturing the following system's characteristics:

- **Temporal Resolution:** Defines the accuracy level used by the model for capturing time.
- **Data Resolution:** The level of abstraction used for the data used in the computations defined by the model.
- **Functional Resolution:** Defines the type of the description used for specifying the system's behavior (Digital logic Boolean operations, algorithmically and mathematically)
- **Structural Resolution:** The level of detail used in capturing the structure of the system.
- **Software Programming Resolution:** Defines the level of abstraction used in order to program the model's behavior.

Additionally, VSIA taxonomy makes a clear distinction between the *external* and *internal* details of a model by evaluating their abstraction levels in both cases. External details are related to the level of detail exploited by the model to present the way it communicates with the environment while internal details are related to the level of detail used to describe the internal computations performed by the model. Fig. 1 shows those axes and their corresponding resolution metrics.

3 Extending the Taxonomy

We have used that taxonomy to classify every one of the 18 models and it was proven that those 5 axes are not enough to classify existing models today. For example in Fig. 2 *FSMs* and *Petri-Nets* have exactly the same position in the 5 dimensional space proposed by VSIA while they can be considerably different in terms of expressiveness when used to capture a resource constraint type of behavior.

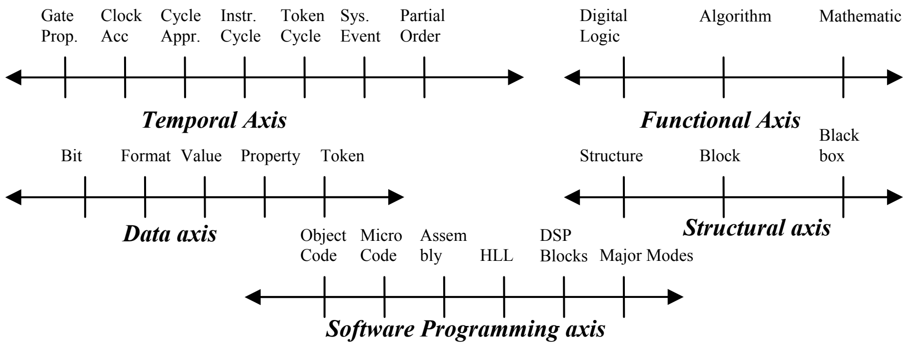

Fig. 1. The VSIA resolution axes

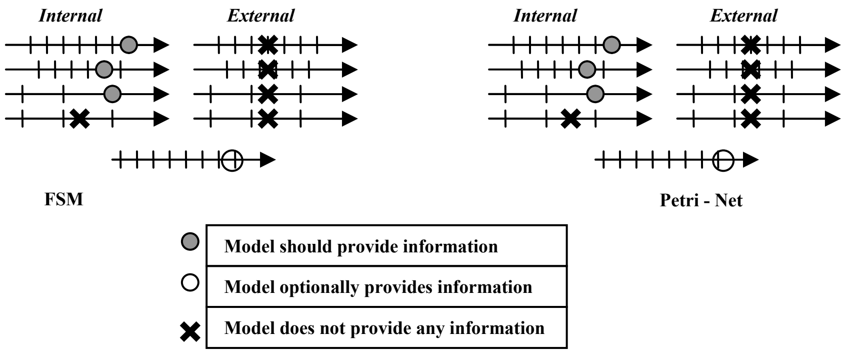
Internal
External
Internal
External
FSM
Petri - Net
Model should provide information
Model optionally provides information
Model does not provide any information

Fig. 2. FSM and Petri-Nets classified using VSIA taxonomy

Therefore we consider it imperative to extend the taxonomy proposed by VSIA to evaluate the "*expressive characteristics*" of a model that span through all levels of abstraction in order to provide to the designer a clear justification on the use of one as opposed to the use of another. In addition, VSIA's taxonomy on models does not provide a classification on models based on the nature of the computation entities they use (states, processes, objects, etc). Such classification is very important to enable the selection of the one that is closer to the conceptual understanding of the desired behavior. For those reasons we have introduced the "*expressive*" and "*type*" axes to support such evaluation-classification.

a) The *type axis* (Fig. 3) is the one to determine the ability of the model to capture different conceptual classes of the modeled behavior based on the nature of the computation entities they use. The classes available are: State oriented models (for systems conceived as being in a number of possible states), Petri-Nets (where parts of the system are competing on the use of limited resources), Data Flow graphs (where systems can be conceived as performing calculations on input data), Activity Oriented (when the system can be expressed as a set of subsequent activities), Structure Oriented (when the system can be conceived as a number of interacting components), Process Oriented (when it is more convenient to capture concurrent communicating processes for the desired behavior) and finally Object Oriented models (where systems are conceived as a set of objects and interfaces interacting to each other).

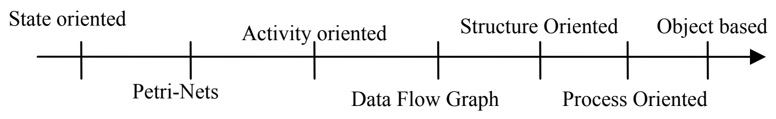

Fig. 3. The resolution of the "type axis"

This axis is highly related to the decision the designer has made on how the system should be conceived depending on the nature of the modeled behavior and its characteristics.

b) The *expressive* axis: Although the *type* axis enables the designer choose one model from the other, it still lacks the ability to distinguish between for example *FSMs and Codesign FSMs* (both state oriented) in terms of their expressive power. For that reason we have introduced the *expressive axis*. The expressive axis is used to evaluate the model's expressive power for the behavior and design task at hand depending on the complexity introduced by the model when a specific characteristic is needed (concurrency, implicit communication, data or control oriented etc) and the method that will be used for the fulfillment of a specific design task (verification through simulation or mathematical analysis, performance estimation etc) Consider for example the *FSM* and *Hierarchical Concurrent FSM* model used to express the behavior of a two elevator controller in a building consisting of three floors. Using a Finite State machine model which does not support concurrency we would need 9 states to represent all possible floor combinations for the location of the two elevators (each state represents a specific location for both elevators). Using Concurrent Finite State Machines we just need to use 6 states in total, 3 for the FSM capturing the location of the first elevator and another three capturing the location of the second. In other words, the "*concurrency*" characteristic of the expressiveness of the model is stronger on HCFSM since it reduces the number of states from 9 to 6 and as a result reduces complexity.

4 Resulting Evaluation Based on the "Expressive" Axis

We have located all possible expressive characteristics (such as concurrency, implicit communication, behavior completion, etc) that may be important for a specific design task and evaluated each model according to them. The evaluation has been captured using ranking values whose meaning and use is illustrated in Table 1. Generally, the greater the value, the better the model's ability to capture each "*expressive*" characteristic. On the first column of Table 1 we present each characteristic along with a brief explanation of what it defines while the other columns provide a brief description of the meaning of a specific score. Blank entries represent the absence of the corresponding ranking value for the specific characteristic. Choosing several behaviors and expressing them using every possible model we come up with the evaluation presented in Table 2.

Ranking Value	3	2	1	0
Data-Oriented *(The model's power in capturing data entities)*	Data manipulation allowed by the model	Data entities are mainly used for control decisions	High cost in complexity	No
Control-Oriented *(The model's power in capturing control flow related signals)*	-	Control is captured using control signal	Control is based on data value comparisons	No
Concurrency *(The model's power in expressing concurrent execution)*	Concurrent paths and resource conflicts can be expressed	Different paths can be expressed that are executed concurrently	Concurrency can be expressed implicitly (complexity increase)	No
Implicit Communication *(data sharing on concurrent models)*	-	-	Yes	No
Explicit Communication *(control signals among concurrent models)*	-	-	Yes	No
Synchronous *(Flow of execution can be time dependent)*	-	-	Yes	No
Discrete-Event *(Flow of execution defined through events)*	-	-	Yes	No
Behavioral Hierarchy (*Hierarchical models in the same design)*	-	-	Yes	No
Behavioral Completion *(There is a way to specify the completion of the execution of the model)*	-	Yes	High cost in complexity	No
Heterogeneity *(Hierarchy supports different models)*	-	-	Yes	No
Support by Languages *(Whether there exist languages that express*	-	More than two languages	At least one language	One language
Formal Verification *(Formal methods for correctness)*	-	-	Yes	No
Exception Handling *(Handling of undeterministic events)*	-	Yes	High cost in complexity	No

Table 1. Explanation of the ranking in **Table *2***

When a specific task has to be performed for a specific behavior the designer applies weights ranging from 0 to 1 to every characteristic (to indicate its importance of the task in question) and calculates the total weighted sum for every model to establish a ranking using the calculated scores. This ranking is the one to be captured by the "*expressive axis*".

5 Specification Languages

The role of Specification Languages is complementary to the role of models. Specification languages serve as an alternative way for expressing the behavior of the system using a selected model. While the behavior is usually conceived by the designer having a specific model in mind, the specification language allows the behavior to be captured in a formal way by providing strict syntactic rules and semantics which can be used to perform valuable performance estimation and verification tasks for the

Model	Data-oriented	Control-oriented	Concurrency	Implicit Communic	Explicit Comm	Synchronous	Asynchronous (Discrete-Event)	Behavioral Hierarchy	Behavioral Completion	Heterogeneity	Support by Languages	Formal Verification	Exception Handling
FSM	1	2	1	0	0	1	1	0	1	0	2	1	1
HCFSM	1	2	2	0	1	1	1	0	1	0	2	1	1
FSMD	2	2	1	0	0	1	1	0	1	0	1	1	1
CFSM	2	2	2	1	1	1	1	0	1	0	1	1	1
HFSM	2	2	2	1	1	1	1	1	1	1	1	0	1
PN	1	2	3	0	1	1	1	0	1	0	1	1	1
Timed PN	1	2	3	0	1	1	1	0	1	0	1	1	1
Time PN	1	2	3	0	1	1	1	0	1	0	1	1	1
Timed-place PN	1	2	3	0	1	1	1	0	1	0	1	1	1
DFG	3	0	0	0	0	0	1	0	1	0	2	1	0
CDFG	3	1	0	0	1	0	1	0	1	0	2	1	0
Flowcharts	3	1	0	0	0	0	1	1	1	1	2	1	0
SDF	3	0	2	1	0	0	1	0	0	0	2	0	0
CSDF	3	0	2	1	0	0	1	0	0	0	2	0	0
Structure	3	2	2	1	1	1	1	1	0	1	2	0	2
OBM	3	1	1	1	0	0	1	1	0	1	2	0	2
CSP	1	2	2	1	1	0	1	1	1	1	1	1	0
Heter/ous CSP	2	2	2	1	1	0	1	0	1	1	1	1	0

Table 2. Evaluating the expressiveness of the model

design process. Specification languages are always bound to a specific model for which they provide all the syntactic constructs needed to express the properties and the interaction of its building blocks. Although it is possible to use the same specification language to capture more than one model this is usually possible in lower levels of the specification hierarchy. In other words each language is more effective in capturing a model used at the top-most level of the hierarchy, while it is still capable in capturing other models at lower levels. Considering for example VHDL, while a plethora of models can be captured by the language (process oriented, data flow), the language follows the properties of a structural model at its top most level. This tight relationship leads us to the conclusion that the expressiveness of a specification language can be measured by locating its underlying model and performing the classification introduced in the previous section.

As previously mentioned, specification languages serve as valuable tools for various tasks of the design process. It is crucial to locate their most important characteristics, which can assist in performance estimation analysis or verification during the Hardware/Software Codesign phase. We have located the three most important ones which are listed below:

- **Executability** is related to whether a behavior expressed using a specification language can be compiled and executed for the computer. The ability to do so, gives the designer the possibility to create a simulator for the modeled behavior which he/she can exercise to measure its performance or verify its correctness.
- **Synthesizability** is related to the existence of synthesis tools accepting as input the specification language and producing the actual implementation. Such power

is useful in Hw/Sw Codesign since parts that need to be executed in hardware have to be expressed in some synthesizable language. A special "Methodology" value is used to represent the fact that for a specification language there exists a specific methodology that automatically leads to a synthesizable representation of the desired system.

- **Semantics** define the meaning of the syntactic constructs of the language. Depending on their nature they can be used to translate the statements of the language to properties and relations in a mathematical domain for verification (Formal denotational Semantics), express interconnections of components of the model (informal denotation semantics), use the already defined meaning of the programming domain (programming language semantics) or be translated to mathematical expression used to calculate performance characteristics of the behavior (performance related semantics).

In Table 3 each specification language is classified according to those three characteristics.

Language	Semantics	Executable	Synthesizable
Esterel [29]	Formal denotational	No	No
SpecCharts [30]	Formal denotational	Yes	No
StateCharts[31]	Formal denotational	Yes	No
RSML	Formal denotational	Yes	No
Petri Net[21]	Formal denotational	Yes	No
C	Programming Language	Yes	Methodology
Verilog[10]	Informal denotational	Yes	Yes
VHDL[11]	Informal denotational	Yes	Yes
Verilog[10]	Informal denotational	Yes	Yes
HardwareC[32]	Informal denotational	Yes	Yes
SystemC[33]	Informal denotational	Yes	Methodology
SuperLog[34]	Informal denotational	Yes	Methodology
SpecC[12]	Informal denotational	Yes	Methodology
JAVA[35]	Programming Language	Yes	Methodology
C++[9]	Programming Language	Yes	Methodology
UML[28]	Programming Language	No	Methodology
OpenJ[36]	Programming Language	Yes	No
SDL[25]	Programming Language	Yes	No
CSP[27]	Formal denotational	No	No
ROSETTA[37]	Performance Related	No	No

Table 3 Classifying specification languages

For the specification languages surveyed, their relation to models (the hierarchical top-most model they are most effective in expressing) is displayed in Fig. 4.

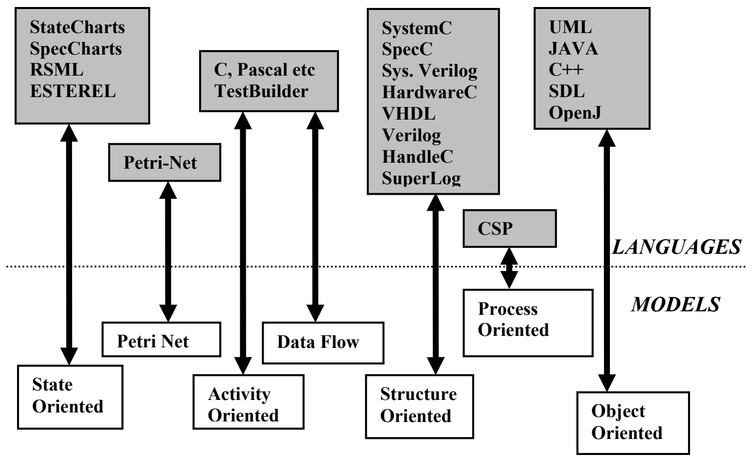

Fig. 4. Relation between models and specification languages

Clearly the classification in Table 3 is insufficient for the selection of a single specification language for a specific design. A lot of them fall in the same category according to the selected features. The most important features though for the selection of a specification language are: the underlying model used (illustrated in Fig. 4), the design task that needs to be performed and the way it will be fulfilled (Table 3), the familiarity of the design team with a specific language and the existence of tools by the people who provide it. The last two issues are highly influenced by industry trends and introduced standards and can not be captured by a strict feature based categorization. A representative example is VHDL. Although Verilog provides similar constructs and falls in the same category, the acceptance of VHDL as a standard has tremendously biased its use against Verilog as a system specification language.

6 Illustrative Example

To illustrate the usefulness of the feature based selection of the most appropriate model-specification language pair we provide an example of the specification of an elevator controller.

Two elevators operate in a 3-story building (including the ground floor). People press a button at each floor to indicate that they want to use the elevator. Inside the elevator there are also three buttons to choose the desired floor. Those signals may occur only when the elevator is at one of the three floors and define a change in the state of the elevators. States of the system are conceived to correspond to the floor the elevator currently is. The second elevator shares the same input events with the first. The principle governing the movement of both elevators defines that the closest elevator satisfies a request from a floor. If both elevators are at the same floor then the first elevator handles the request. Since there are not any strict real time constraints one of the most important design tasks is that of verification. From the specification the designer concludes that in order to capture the behavior of such a system a model is needed with the following features: control oriented (1), concurrency (1), explicit communication (1), asynchronous (1), verification (1) and some other features such as support by languages (0.5), exception handling (0.2) and behavioral completion (0.6) which are needed but their availability is not that important. The number in the parentheses represents the weights to be applied to the corresponding features to present the ranking of models in the expressive axis (Fig. 5).

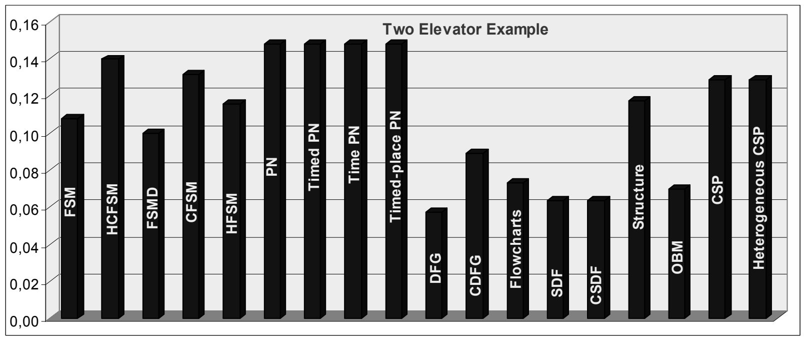

Fig. 5. Ranking of models with respect to the expressive axis and applied weights

The conceptual understanding of the specification implies that a state oriented or Petri net model is more preferred. As a result the "*type*" feature narrows our search to models of that category. By looking the expressive axis we conclude that the most appropriate model seems to be the Petri-Net (PN) followed by the HCFSM while a bad choice would be an FSM model. Fig. 6 illustrates the specification of the controller's behavior using those three models.

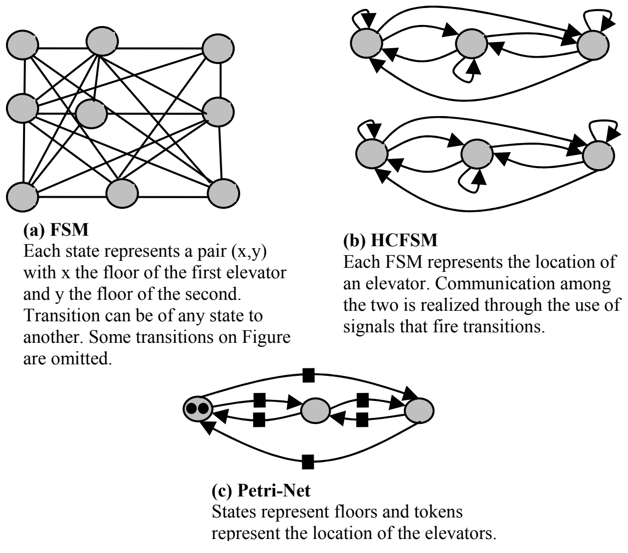
(a) FSM
Each state represents a pair (x,y) with x the floor of the first elevator and y the floor of the second. Transition can be of any state to another. Some transitions on Figure are omitted.
(b) HCFSM
Each FSM represents the location of an elevator. Communication among the two is realized through the use of signals that fire transitions.
(c) Petri-Net
States represent floors and tokens represent the location of the elevators.

Fig. 6. The specification of the elevator controller.

Clearly the complexity of the specification has decreased significantly in the Petri-Net specification. Thus it is obviously the most effective choice for capturing the desired behavior. Although Petri-Nets would be appropriate, the need of using a specification language for simulation purposes may possibly lead to the selection of HCFSM for the specification. From the relation of models to specification languages in *Figure 4* it is clear that a plethora of languages may be used for a state oriented model. In specific, one of the following languages are the most appropriate be used to capture a HCFSM: StateCharts, SpecCharts, RSML and ESTEREL. ESTEREL is not chosen since it is not executable so the designer may choose one of the other three for the specification depending on the availability of tools and support or his/her previous experience.

7 Conclusion

In this paper we have established a set of well-defined features for evaluating and classifying models and specification languages for the required behavior and design task in the System Level Design process. We have extended VSIA's taxonomy on models [13] by introducing two additional axes related to important features of models for capturing the desired behavior and evaluated existing models of computation according to them. We have also introduced a 3 dimensional space for the classification of specification languages and revealed the interrelationship of the latter with the models they express. The whole feature-based system can be proven a valuable tool for selecting the most appropriate model-specification language pair (the one that reduces the complexity of the expressed behavior and can support the needed design tasks) for a desired behavior. Moreover it can be used for the evaluation and classification of new models and specification languages introduced for System Level Design.

References

1. D.D. Gajski, F. Vahid, S. Narayan, J. Gong, *Specification and Design of Embedded Systems*, Prentice Hall, (1994)
2. H. Chang, L. Cooke, M. Hunt, G. Martin, A. McNelly, L. Todd, *Surviving the SoC Revolution*, Kluwer Academic Publishers, (1999)
3. "Overview of the PTOLEMY project", Technical Memorandum UCB/ERL M01/11, March 6, (2001)
4. "A Framework for Hardware-Software Co-Design of Embedded Systems", available at: www-cad.eecs.berkeley.edu/~polis
5. R.A. Bergamaschi, S. Bhattacharaya, R. Wagner, C. Fellenz, M. Muhlada, F. White, W.R. Lee, J.M. Daveau, "Automating the Design of SOCs Using Cores", *IEEE Design and Test of Computers*, 18(5):32-45, (2001)
6. J.A. Rowson, A.S. Vincentelli, "Interface-Based Design", *Proceedings ACM DAC Conference*, (1997) 178-183

7. T. Kambe, A. Yamada, K. Nishida, K. Okada, M. Ohnishi, A. Kay, P. Boca, V. Zammit, T. Nomura, "A C-based Synthesis System, Bach, and its Application", *Proceedings Asian South Pacific Design Automation Conference*, (2001) 151-155
8. A.D. Pimentel, L.O. Hertzberger, P. Lieverse, P. van der Wolf, E.F. Deprette, "Exploring Embedded-Systems Architectures with Artemis", *IEEE Computer Magazine*, 34(11):57-63 (2001)
9. D. Verkest, J. Kunkel, F. Shirrmeister, "System Level Design Using C++", *Proceedings DATE Conference*, Paris, France, (2000) 74
10. S. Brown, Z. Vranesic, *Fundamentals of Digital Logic With VERILOG Design*, Mc Graw Hill, (2003)
11. J.R. Armstrong, F.G. Gray, *VHDL Design: Representation and Synthesis*, 2nd Edition, Prentice Hall, (2000)
12. D. Gajski, J. Zhu, R. Domer, A. Gerstlauer, S. Zhao, "The SpecC Methodology", UC Irvine, Technical Report ICS-TR-99-56, (1999)
13. System Level Design Development Working Group, "VSI Alliance: Model Taxonomy", Version 2.1, July (2001)
14. C. Kern, M.R. Greenstreet, "Formal Verification in Hardware Design: a Survey", *ACM Transactions on Design Automation of Electronic Systems*, 4(2):123-193 (1999)
15. L.A. Cortes, P. Eles, Z. Peng, "A Survey on Hardware/Software Codesign Representation Models", SAVE Project Report, Department of Computer and Information Science, Linkoping University, Sweden, (1999)
16. F. Vahid, T. Givargis, *Embedded System Design: a Unified Hw/Sw Introduction*, John Wiley, (2000)
17. B. Lee, E.A. Lee, "Hierarchical Concurrent Finite State Machines in Ptolemy", *Proceedings International Conference on Application of Concurrency to System Design*, Fukushima, Japan, (1998) 34-40
18. B. Lee, E.E. Lee, "Interaction of Finite State Machines and Concurrency Models", *Proceedings of the IEEE*, (1998) 1715-1719
19. G. DeMichelli, M. Sami, *Hw/Sw Codesign*, Kluwer AcademicPublishers, (1999)
20. H.C.C. Hsiech, Formal Methods for Embedded System Design, PhD dissertation, University of California at Berkeley, (2000)
21. L.A. Cortes, P. Eles, Z. Peng, "A Petri-Net based Model for Heterogeneous Embedded Systems", *Proceedings NORCHIP Conference*, (1999) 248-255
22. M. Sgroi, L. Lavagno, "Synthesis of Embedded Software Using Free-Choice Petri Nets", *Proceedings ACM DAC Conference*, (1999), 805-810
23. G. DeMicheli, *Synthesis and Optimization of Digital Circuits*, Mc Graw Hill, (1996)
24. F. Slomka, M. Dorfel, R. Munzenberger, "Generating Mixed Hardware/Software Systems from SDL Specifications", *Proceedings IEEE Codesign Conference (CODES),* (2001) 116-121
25. T.M. Parks, J.L. Pino, E.A. Lee, "A Comparison of Synchronous and Cyclo-Static Dataflow", *Proceedings Asilomar Conference on Signals and Systems*. Vol.1, Pacific Grove, CA (1995) 204-210
26. D.D. Gadjski, L. Ramachandran, "Introduction to High Level Synthesis", *IEEE Design and Test of Computers*, (1994) 44-54
27. S. Schneider, *Concurrent and Real-time Systems: The CSP Approach*, John Wiley, (2000)
28. G. Martin, L. Lavagno, J.L. Guerin, "Embedded UML: a Merger of Real-time UML and Co-design", *Proceedings IEEE Codesign Conference (CODES),* (2001) 23-28
29. G. Berry and L. Cosserat, "The ESTEREL synchronous programming language and its mathematical semantics", Ecole Nationale Superieure de Mines Paris, (1984)
30. F. Vahid, S. Narayan, "SpecCharts: a Language for System Level Synthesis, *Proceedings CHDL Conference*, (1991) 145-154

31. Heimdahl M.P.E, Keenan D.J., "Generating Code from Hierarchical State-based Requirements", *Proceedings 3rd IEEE International Symposium on Requirements Engineering*, (1997), 210–219
32. D.C. Ku, G. DeMichelli, "Hardware-C A Language for hardware Design", Technical Report CSL-TR-88-362, Computer Systems Lab, Stanford University, (1988)
33. A. Fin, F. Fummi, M. Signoretto, "SystemC: a Homogenous Environment to Test Embedded Systems", *Proceedings IEEE Codesign Conference (CODES),* Copenhagen, Denmark, (2001) 17-22
34. P. Flanke, S. Davidmann, "Superlog, a Unified Design Language for System-on-Chip", 2000, ASPDAC
35. R. Helaihel, K. Olukotun,, "Java as a Specification Language for Hardware-Software Systems", *Proceedings ICCAD Conference*, (1997)
36. J. Zhu, D. Gajski, , "OpenJ: An Extensible System Level Design Language", *Proceedings Design Automation and Test Conference in Europe*, (1999)
37. P. Alexander, C. Kong, "ROSETTA: Semantic Support for Model-Centered Systems-Level Design", *IEEE Computer Magazine*, November (2001)