# Specification Modeling Methodologies for Reactive System Design

**Ambar Sarkar** **Computer Science** 
**Ronald Waxman** **Electrical Engineering** 
**James P. Cohoon** **Computer Science**

University of Virginia, Charlottesville, VA 22903
email: { ambar | ronw | cohoon@virginia.edu }

## Abstract

The goal of this paper is to investigate the state-of-the-art in specification-modeling methodologies applicable to reactive-system design. By combining the specification requirements of a reactive system and the desirable characteristics of a specification-modeling methodology, we develop a unified framework for evaluating any specification-modeling methodology applicable to reactive-system design. A unified framework allows the designer to look at the spectrum of choices available and quickly comprehend the suitability of a methodology to the specific application.

Using the unified framework, we study a number of representative methodologies, identifying their respective strengths and weaknesses when evaluated for the desired characteristics. The differences and relationships between the various methodologies is highlighted. We find our framework to be quite useful in evaluating each methodology. A summary of our observations is presented, together with recommendations for areas needing further research in specification modeling for reactive systems. Two such areas are improving model continuity and providing better complexity control, especially across different abstraction levels and modeling domains. We also present a description of each methodology studied in the unified framework.

## 1. Introduction

A *reactive system* is one that is in continual interaction with its environment and executes at a pace determined by that environment. Examples of reactive systems are network protocols, air-traffic control systems, industrial-process control systems etc.

Reactive systems are ubiquitous and represent an important class of systems. Due to their complex nature, such systems are extremely difficult to specify and implement. Many reactive systems are employed in highly-critical applications, making it crucial that one considers issues such as reliability and safety while designing such systems. Designing reactive systems is considered to be problematic, and pose one of the greatest challenges in the field of system design and development.

In this paper, we discuss specification-modeling methodologies for reactive systems. Specification modeling is an important stage in reactive system design where the designer specifies the desired properties of the reactive system in the form of a specification model. This specification model acts as the guidance and source for the implementation. To develop the specification model of complex systems in an organized manner, designers resort to specification modeling methodologies. In the context of reactive systems, we can call such methodologies *reactive-system specification modeling methodologies*.

Given the myriad of specification methodologies available today, each different from the other in its chosen formalism, analysis techniques, methodological approaches etc., we establish a framework that allows one to study the merits and demerits of any specification methodology for reactive systems.

By combining the specification requirements of a reactive system and the desirable characteristics of a specification-modeling methodology, we develop a unified framework for evaluating any specification-modeling methodology applicable to reactive-system design. A unified framework allows the designer to look at the spectrum of choices available and quickly comprehend the suitability of a methodology to the specific application.

In order to identify the specification requirements of reactive systems, we enumerate their characteristics. During the specification process, a conceptual model for each of these characteristics is developed by the designer. The combination of the developed conceptual models results in a specification model. In order to enumerate the desirable attributes of a specification-modeling methodology, we broadly categorize them under the three categories, namely: supporting complexity control; developing and analyzing the specification; and maintaining model continuity with respect to other modeling stages. The problem of maintaining model continuity can be decomposed into three subproblems: checking conformance of the specification model with those developed in other design stages, making the specification visible at all other design stages, and providing back-annotation of design details into the specification.

Using the unified framework, we study a number of representative methodologies, identifying their respective strengths and weaknesses when evaluated for the desired characteristics. The differences and relationships between the various methodologies is highlighted. We find our framework to be quite useful in evaluating each methodology. A summary of our observations is presented, together with recommendations for areas needing further research in specification modeling for reactive systems. Two such areas are improving model continuity and providing better complexity control, especially across different abstraction levels and modeling domains. We also present a description of each methodology studied in the unified framework. The methodologies studied are applicable to the specification of reactive systems.

## 2. Characteristics of reactive systems

Reactive systems follow what can be called the stimulus-response paradigm of behavior: on the occurrence of stimuli from its environment, the reactive system typically responds or reacts by changing its own state and generating further stimuli. Reactive systems are typically control dominated, in the sense that control-related activities form a major part of the reactive system's behavior.

Reactive systems have typically been contrasted with transformational systems. The behavior of a transformational system can be adequately characterized by specifying the outputs of the system that result from a set of inputs to the system. In contrast, the behavior of a reactive system is characterized by the notion of *reactive behavior* [HLN88, Pnu86]. To describe reactive behavior, the relationship of the inputs and outputs over time should be specified. Typically, such descriptions involve complex sequence of system states, generated and perceived events, actions, conditions and event flow, often involving timing constraints.

For example, given an adder, its behavior is easily defined as producing an output which is the sum of its inputs at any given time. This behavior is unaffected by the environment in which the adder operates, and therefore the output, given a set of inputs, remains the same regardless of time. In contrast, a traffic signal continually monitors its traffic and establishes traffic flow based on its current traffic pattern. It would be difficult to describe the behavior of the traffic signal by specifying the output (traffic signals) as a result of the input (current traffic pattern), since outputs may depend on how the traffic pattern varies over time.

We now present important characteristics of reactive systems.

- *State-transition oriented behavior:*

  Reactive systems are intrinsically state based, and transition from one state to another is based on external or internal events. The concept of states is an useful tool to model the relationship between the inputs and outputs over time.
- *Concurrent in nature:*

  Reactive systems generally consist of concurrent behaviors that cooperate with each other to achieve the desired functionality. Concurrency is further characterized by the need to express communication, synchronization, and nondeterminism among concurrent behaviors.
- *Timing sensitive behavior:*

  Two categories of timing characteristics can be identified: namely, functional timing and timing constraints. Functional timings represent actual time elapsed in executing a behavior. Functional timings may change with the implementation. Guessing correct functional timing associated with a specification is therefore difficult during the specification stage. Timing constraints, on the other hand, specify a range of acceptable behavior that the implementation is allowed to exhibit. The constraints are typically externally imposed on the system, and all correct implementations of the system must obey such constraints.

  A special class of reactive systems are real-time systems, which have the added attribute of temporal correctness in addition to the functional correctness of a reactive system
- *Exception-oriented behavior:*

  Certain events may require instantaneous response from the system. This requires the system to typically terminate the current mode of operation and transition into a new mode. In some cases, such as interrupt handling, the system is required to resume in the original state at which the interrupt occurred.
- *Environment-sensitive behavior:*

  Since the response of a reactive system depends heavily on the environment in which it operates, a reactive system can often be characterized by the environment. For example, the specification of a data-communication network designed for handling steady network-traffic can be expected to quite different than one designed for bursty traffic.
- *Nonfunctional characteristics:*

  Reactive systems often are characterized by properties which are nonfunctional in nature, such as reliability, safety, performance, etc. These properties are often not crucial to the system's functionality, but are often considered important enough to evaluate alternative implementations.

The expression of all the characteristics of a reactive system should be directly supported by the language of specification. Since a specification methodology is typically associated with a specific set of specification languages, the effectiveness of the methodology lies in how well its specification languages support the expression of the above characteristics. In addition to the language, the methodology plays an important role in developing the representation in a methodical rather than a haphazard manner.

## 3. Specification modeling methodology requirements

To appreciate the requirements of a specification modeling methodology, one must understand its role in the overall design process. The design process of a reactive system can be segmented into three major phases: the specification phase, the design phase, and the implementation phase. In the specification phase, the requirements of the system under design are formulated and documented as a specification. In the design phase, the possible implementation strategies are considered and evaluated. Finally, in the implementation phase of design, the specification is realized as a product. Note that even though the three phases may be initiated in the order we mention them, these phases often overlap. Overlapping of phases implies that during the design process, one phase may not be completed before the next phase is initiated. Another point to note is that in some methodologies, it can be difficult to distinguish the three phases from one another, especially when the difference in the levels of abstraction between the specification and implementation is small. In such cases, the specification is often a reflection of the implementation, and the process of developing specification may reflect the design phase.

We are concerned with the specification phase, where the designer (henceforth known as the specifier) typically develops a model of the system called the specification model. A specification model, or simply, a *specification* is a document that prescribes the requirements, behavior, design, or other characteristics of a system or system components. By developing and analyzing the model, the specifier makes predictions and obtains a better understanding of the modeled aspects of the system.

We now precisely define a specification-modeling methodology. A *specification-modeling methodology* is a coherent set of methods and tools to develop, maintain and analyze the specification of a given system. A method, in the context of specification modeling, consists of three components. The first component is an *underlying mode*l which is used to conceptualize and comprehend the system requirements. The second component is a *set of languages* that provides notations to express the system requirements. Finally, the third component of a method is a *set of techniques* ranging from loosely specified guidances to detailed algorithms that is needed to develop a complete specification from preliminary concepts.

In this paper, we focus mostly on the methods. The tools are important. However, the tools are primarily concerned with providing support for the methods. Therefore, the tools can be characterized by the method component of a methodology and are not considered separately.

There are three key requirements of a specification modeling methodology. First, it should be supported by languages that are appropriate for specifying requirements of the system. Second, it should provide assistance in controlling the complexity of specifying nontrivial systems. Third and finally, the methodology should also support the usefulness of the specification model across the design and implementation phases.

## 4. Specification modeling methodology for reactive systems

Based on the characteristics of a reactive system and the requirements of a specification-modeling methodology, we synthesize the necessary attributes of a reactive-system specification modeling methodology. These attributes are presented in Figure 1.

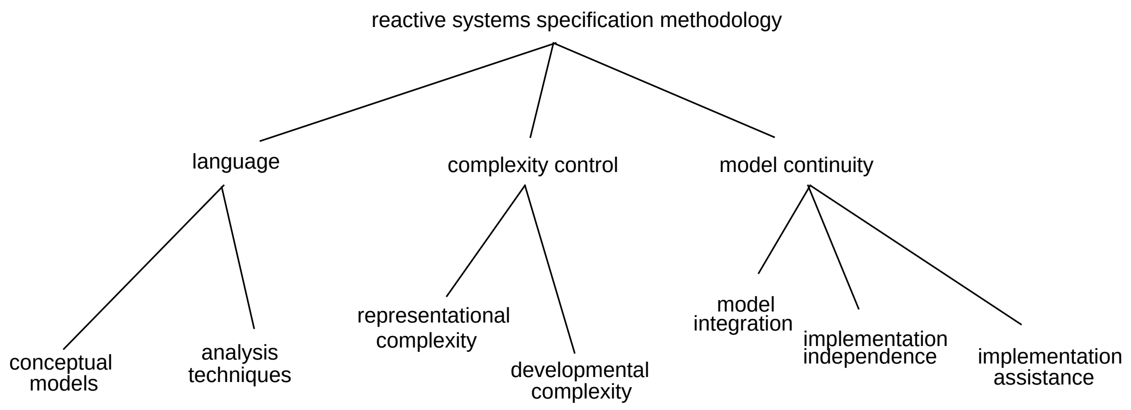
reactive systems specification methodology
language
complexity control
model continuity
conceptual models
analysis techniques
representational complexity
developmental complexity
model integration
implementation independence
implementation assistance

**Figure 1. Attributes of a reactive-system specification methodology**

There are three major attributes of an reactive system specification modeling methodology: language attribute, complexity-control attribute, and model-continuity attribute.

The language attribute represents the modeling languages that supported the methodology. This attribute distinguishes reactive system specification modeling methodology from other specification methodologies, since the chosen languages should provide appropriate conceptual models and analysis techniques applicable to reactive systems. There are two dimensions of the language attribute. The conceptual-models dimension determines the available conceptual models for expressing reactive systems. The analysis-technique dimension determines the kind of support available for checking the specification consistency.

The second attribute represents the support available in the methodology for complexity control. Complexity-control is necessary for any design methodology that is applicable to design problems of nontrivial complexity. There are two dimensions along which complexity control should be supported: representational complexity and developmental complexity. Representational complexity deals with the understandability of the developed specification, whereas developmental complexity provides support for developing the specification in an organized and productive manner.

The third attribute of reactive system specification modeling methodology, support for model-continuity, distinguishes a reactive system specification modeling methodology as a specification methodology, instead of a design methodology. The specification modeling methodology should be more focussed towards developing and maintaining a specification model instead of a proposed implementation. Support of model-continuity should be considered along three dimensions: integrated modeling, implementation independence, and implementation assistance. Support along these three dimensions ensures that the usefulness of the specification model is maintained beyond the specification modeling stage of a design. Since a considerable investment of time and effort is made in developing a specification model, extending the useful life of the specification model will increase the usefulness and appeal of the associated specification methodology.

To be effective, a reactive system specification modeling methodology must strongly support all three of the attributes mentioned above. We describe each of these components in the following sections. The strengths and weaknesses of a reactive-system specification-modeling methodology can be quickly identified by evaluating the strengths of each of these components in the methodology.

## 4.1. Support for specification languages

The primary purpose of a specification-modeling language is to unambiguously express the desired functionality of a specified system. As discussed in Section 2, a reactive system possesses a number of characteristics. To express and comprehend these characteristics, a number of conceptual models are used. A specification language is defined by the conceptual models it offers the specifier to express these characteristics. In this section, we present the major conceptual models available for each reactive system characteristic. A specification language typically offers one conceptual model for a given characteristic, based on its targeted domain of application. For a good summary of available language techniques, see [BT85, Dav88, GVN93].

In addition to providing conceptual models, the specification language also offers support for analyzing and reasoning about the specification. The type of analysis and reasoning mechanisms that are available depend mainly on the specification language used. We think the two most important language characteristics are whether the language is based on a sound mathematical formalism and whether the specification is executable.

### 4.1.1. Available conceptual models

For each reactive system characteristic, we present the popular conceptual-model alternatives. When evaluating a specification-modeling methodology, one should identify if it provides conceptual models for each reactive-system characteristic. One should then check if the conceptual models that are supported by the specification-modeling methodology are appropriate given the application domain and specifier preference.

For the purpose of evaluation, we consider a methodology suitable for reactive systems specification as long as it offers at least one conceptual model for expressing the following characteristics:

- *System views*

  There are three views that are complementary to each other: *activity*, *behavior*, and *entity* view. In the activity view, the specification represents the activities that occur in the system. Activity in a system is closely tied to the flow of data in a system. In the behavior view, the specification represents the ordering and interaction of these activities. Behavior in a system is often represented in terms of states and transitions, or the flow of control. Finally, in the entity view, the entities in the system, are identified. These entities represent the system components that are responsible for the activities and behavior in the system. Entities in a system are often represented as the data-items present in a system.
- *Specification style*

  There are two primary styles of specification: *model*-oriented and *property*-oriented [Win90]. In a model-oriented specification, the system is specified in terms of a familiar structure such as state-machines, processes, or set theory. In a property oriented specification, the system is viewed as a black-box, and the properties of the system are specified in terms of the directly observable behavior at the interface of the black-box. Generally speaking, model-oriented specifications are considered easier to understand than their property-oriented counterparts. On the other hand, property-oriented specifications are considered less implementation dependent (discussed in Section 4.3.3), since no assumption is made about the internal structure or contents of the system.

- *Concurrency*

  Reactive systems generally consist of concurrent behaviors that cooperate with each other to achieve the desired functionality. Concurrency is further characterized by the need to express communication and synchronization among concurrent behaviors.

  *Communication among concurrent behaviors*

  Communication between concurrently acting portions of a system is usually conceptualized in terms of *shared-memory* or *message-passing* models. In the shared-memory model, a shared medium is used to communicate information. The communication is initiated by the sender process writing into a shared location, where it is available immediately for all receiver processes to read. In the message-passing model, a virtual channel is used, with "send" and "receive" primitives used for data transfer across that channel. Both shared-memory and message-passing models are interconvertible: each model can be expressed in terms of the other model.

  *Synchronization among concurrent behaviors*

  In addition to exchanging of data, the concurrent components of a system need to be synchronized with one another. Such synchronization is needed since the concurrent components often need to coordinate their activities, and have to wait for other components to reach certain states or generate certain data or events. Synchronization may be achieved using control constructs or communication techniques. Examples of control constructs are fork-join primitives, initialization techniques, barrier synchronizations etc. Examples of using communication techniques for synchronization are global-event broadcasting, message passing, global status detection etc.

  In addition to communication and synchronization, a specification language often supports expression of nondeterminism among concurrent behaviors [Dij75, MP91]. We consider nondeterminism an attribute for complexity control rather than a reactive system characteristic, since it allows the specifier to focus on the allowable alternative behaviors without committing to any specific choice. This choice is determined at a later stage depending upon the implementation.
- *Timing constraints*

  Timing constraints are an important part of any reactive system behavior and can be specified either *direct*ly or *indirect*ly. Timing constraints can be specified directly as inter-event delays, data rates, or execution time constraints for executing behaviors. Supporting temporal logic [MP91] can be seen as a direct specification technique. Indirect specification of timing constraints occurs when the actual constraint is implied through a collection of specification language constructs. This approach is followed in Statecharts [Har88], where timing constraints are specified by a combination of states, transitions, and time-outs.
- *Modeling time*

  Reactive system behavior is often specified in terms of their time metric (E.g. wait for 5 minutes to warm up electromechanical equipment). Consideration of time in a separately modeled entity increases the ease with which the specifier can specify such timing behavior, instead of referring to time indirectly.
- *Exception handling*

Certain events may require instantaneous response from the system. This requires the system to typically terminate the current mode of operation and transition into a new mode. In some cases, the system is required to resume in the original state at which the interrupt occurred. Interrupt handling is one such case.

Exception handling can be provided through explicit specification language constructs. Examples are text-oriented exception handling mechanisms in Ada programming language or graphics-oriented mechanisms such as complex transitions and history operators in Statecharts.

- *Environment characterization*

  Since the reactive system is expected to be in continual interaction with its environment, it is important to be able to characterize the environment. Since the environment itself is reactive in nature, one may choose to model it using the same specification language. The operational environment of a reactive system can therefore be specified as an explicit *model* or as a set of *properties*. When specified as a separate model, the environment is seen as a separate entity that interacts with the model of the system under design. For property oriented characterization, the system's operational environment can be specified via hints about various operational conditions such as inter-arrival of events (expected frequencies, timings), expected work-loads, etc.
- *Nonfunctional characterization*

  In addition to providing conceptual models for specifying the functionality of the system, a specification methodology reactive system should also provide support for expressing nonfunctional characteristics such as maintainability, safety, availability etc. Mechanisms to represent hints, properties, or external constraints that a system should follow should be provided.

### 4.1.2. Analysis techniques

There are two main considerations for the support of the methodology for analyzing and comprehending the specified reactive system behavior: support for *formal analysis* and support for *model executability.*

First, we examine whether the language itself has a sound mathematical basis. Having a sound mathematical basis for the specification language enables one to automatically check for inconsistencies in the specification itself. Given the advances in formal techniques and the ever increasing number of safety critical reactive systems being designed, we consider that support for formal analysis of the specification is important. We examine if the specification language has a strong mathematical formalism as its basis. A number of formalisms are available: Petri-nets, finite state machines, state diagram, temporal logic, process algebras, abstract data types etc. A specification methodology may offer several of these formalisms as a choice for analyzing its specification models. In this paper, we observe whether the specification-modeling methodology chose a language which has a formal basis.

Second, given the typical nontrivial complexity of the systems being designed today, executability of the specification is a big help in improving the comprehensibility and robustness of the specification. Executability also offers the ability to experiment with preliminary prototypes of the system under design which is useful to validate the specification against the requirements of the system.

### 4.2. Complexity control

One of the main requirements of a design methodology is to be able to control the complexity of the design process. Support for complexity control in a specification-modeling methodology can exist along two dimensions. The first dimension is the representational complexity, which makes the specification itself concise, understandable, and decomposable into simpler components. The second dimension is developmental complexity, which supports the development of the specification in an incremental, step-wise refined manner.

#### 4.2.1. Representational complexity

Support for this dimension is usually dependent on the specification language chosen. We list this dimension separately under complexity control, since it allows us to separate the complexity control aspect of a specification language from its support for expressing reactive system characteristics.

- *Hierarchy*

  The notion of hierarchy is an important tool in controlling complexity. The basic idea in hierarchy is to group similar elements together and to create a new element that represents this group of similar elements. By introducing the common behavior in this way, multiple levels of abstractions can be supported.
- *Orthogonality*

  A complex behavior can often be decomposed into a set of orthogonal behaviors. By supporting such decomposition in the representation, significant improvement in clarity and understandability can be attained.
- *Representation scheme*

  The representation scheme plays an important role in the understandability of a specification. We make the distinction between *graphical* and *textual* representation schemes. By graphical scheme, we imply visual formalisms where both syntactic and semantic interpretations are assigned to graphical entities. For example, Statecharts, Petri-nets etc. are such visual formalisms. In our opinion, graphical representation schemes are preferable to a textual ones since the former allow the specifier to visualize the system behavior more effectively, especially during execution of the specification. For many textual approaches, however, tools exist today that transform the textual approach into a graphical approach.

#### 4.2.2. Developmental complexity

In addition to the representation of system behavior, a specification methodology must also support the evolution of the specification model from initial conceptualization of system requirements. We think the following dimensions should be supported.

- *Nondeterminism*

  By incorporating nondeterminism in a controlled manner, the specification can leave details to the implementation and final stages. For example, in a typical producer-consumer type system, if requests for both element insertions and element deletions from a buffer arrive simultaneously, the specification may non-deterministically select either operation to be executed first. The commitment to an actual choice in such a scenario is deferred to later design stages. Nondeterminism thus supports a evolutionary approach to specification development. In addition to the incorporation of nondeterminism, a specification methodology should also provide mechanisms to detect and resolve nondeterminism. It is often difficult to detect nondeterminism in specifications of complex systems, given the complex interrelationships of the behaviors of the system components. If left undetected and consequently unresolved, nondeterminism is a potential source of ambiguity.

- *Perfect synchrony hypothesis*

    Perfect-synchrony hypothesis [Ber91] implies that a reactive system produces it outputs synchronously with its inputs. In other words, the outputs are produced instantaneously after the inputs occur. This assumption of perfect-synchrony is borne out in many cases, especially if the inputs change at rates slower than the system can react. For example, in a clocked system, if the clock cycle is long enough, the system gets a chance to stabilize its output values before the next clock event occurs.

    Based on the assumption of perfect-synchrony hypothesis, specification languages can be divided into two types: *synchronous* and *asynchronous*. In synchronous languages, time advances only if explicitly specified. In asynchronous languages, time advances implicitly. Examples of synchronous languages are Statecharts, Esterel etc. Examples of languages based on asynchronous hypothesis are those based on concurrent programming languages such as Ada, SREM etc.

    Perfect-synchrony assumption makes the specification concise, composable with other specifications, and in general lend the specification to a number of elegant analysis techniques.

    However, the assumption of perfect synchrony may not be valid at all levels of abstractions especially if the reaction of the system is complicated. For example, if the reaction is a lengthy computation, clearly assumption of perfect synchrony will be violated, as the input may change before the computation is completed.

- *Developmental guidance*

    Guidance for model development is helpful in identifying the next step in the process of specifying a system. One may do it *bottom up*, where the primitives are first identified and then combined. Another guidance is in the form of a *top-down* approach, where a specification is decomposing into smaller and more detailed components. Yet another approach is called the *middle-out* approach, which combines both top-down and bottom-up approaches.

    Typically, development style is a matter of choice and not strictly enforced. In some methodologies, however, this can be enforced. Enforcing the style may interfere with designer creativity.

### 4.3. Model-continuity

Significant effort is involved in the development and debugging of a model of the system under design. Once the model has been developed and analyzed, however, it is often discarded and is not revisited in the remainder of the design process. Such a limited useful life of a specification model tends to make the corresponding modeling methodology unpopular with designers. The reason for the limited usefulness of a specification model is the problem of maintaining model continuity.

*Model continuity* can be defined as the maintenance of relationships between models created in different model spaces such that the models can interact in a controlled manner and may be utilized concurrently throughout the design process. The problem of maintaining model continuity for a specification can be divided into following three subproblems: model integration, implementation assistance, and implementation independence. Model integration addresses the challenge of making the specification model compatible with models developed during the design and implementation stages. Implementation assistance increases the usefulness of the specification by helping during the design and implementation stages. Implementation independence increases the useful life of the specification by not committing it to a particular design/implementation choice, thus avoiding restriction of creativity during the design process.

### 4.3.1. Model integration

Model integration can be subdivided into three attributes:

- *Conformance*

  The conformance attribute identifies how a methodology addresses the first subproblem of model continuity, namely: checking conformance among models developed. The methodology should provide either a simulation-based support or an analysis-based support for checking conformance between the models (or both).

  We categorize conformance checking along two dimensions: *vertical* and *horizontal*. Vertical-conformance checking involves validating conformance between models representing different levels of abstraction. Horizontal-conformance checking involves validating conformance between models representing different modeling domains. To be effective, the methodology must provide support for checking conformance along both dimensions. For example, one should be able to check the conformance between an algorithmic-level model and a logic-level of a system. This is a example of vertical-conformance checking. As an example of horizontal-conformance checking, one should also be able to check the conformance between a functional level behavioral model involving register-transfers and state-sequencers and a structural model involving ALUs, MUXs, registers etc.

- *Model interaction*

  The interaction attribute identifies how a methodology addresses the second and third subproblems of model continuity, namely: maintaining visibility of the specification model during the implementation phase and incorporating relevant details obtained from the implementation phase back into the specification model. Supporting such a high degree of interaction and information flow among these models requires integrated modeling across different levels of abstraction and modeling domains. By integrated modeling, we imply that the flow of information occurs in both directions across the model boundaries. This flow of information can occur during either integrated simulation or integrated analysis of both models. A methodology must support bidirectional information flow across model boundaries.

  Analogous to conformance checking, we categorize model interaction along two dimensions: vertical and horizontal. Vertical interaction occurs between models belonging to different levels of abstractions, whereas horizontal interaction occurs between models belonging to different domains of modeling. A methodology must provide mechanisms that support both vertical and horizontal model interactions.

- *Complexity*

  The complexity attribute addresses the problem of controlling complexity during the development and analysis of models throughout the design stages. Given the considerable complexity of models representing nontrivial systems, support for this attribute is necessary for effective implementation of the conformance and interaction attributes.

Complexity control is primarily achieved by supporting a hierarchy of representations. Support of hierarchy significantly reduces design time, as the designer is allowed to provide less detail in creating the original representation. For adding or synthesizing further information, he or she can then use automated or semi-automated design aids. In addition to the addition or the synthesis of details, a hierarchical approach allows the designer to quickly identify what portion of the design should be expanded upon, without necessarily expanding the rest of the system. This incremental-expansion approach is of tremendous advantage when the expanded representation is radically different from the original representation. By enabling incremental modifications, a hierarchical representation improves designer comprehension of the effect of change on the original model.

Similar to conformance checking and model interaction, model complexity can also be divided into two dimensions: vertical and horizontal. Abstraction of a lower-level model into a higher-level is an example of managing vertical complexity. Combination of models from different modeling domains into a unified representation is an example of managing horizontal complexity. The hierarchical representation must possess capability to manage both horizontal and vertical complexity.

### 4.3.2. Implementation assistance

The task of developing an implementation from a specification is complex. As a result, there has been a significant research in the automated synthesis of implementations from specifications[BT85, CPT89, CR85, HD88, LB91, TM91, WWD92]. There are two prime motivations for implementation assistance: reduction of designer effort and increase in implementation accuracy. By supporting automated/semi-automated techniques for the synthesis of a design/implementation, there is a significant reduction in required designer effort. Also, an automated technique avoids human errors that can be introduced otherwise during the manual design process. Synthesis of efficient implementations from system-level specifications is still immature. Another limitation of current synthesis techniques is that they are generally based on the structure of the specification, thus limiting design space and therefore producing less optimal solutions.

### 4.3.3. Implementation independence

According to [Win90], a specification has an implementation bias if it specifies externally unobservable properties of the system it specifies. A specification is therefore considered implementation independent if it lacks implementation bias. While evaluating a specification methodology, we examine how well it supports implementation independence. There are two key advantages of an implementation independent specification. First, it allows the specifier to focus on describing the behavior of the system, rather than how it is implemented. Second, it avoids placing unnecessary restrictions on designer freedom.

## 5. A survey of methodologies

We consider ten specification methodologies and evaluate their relative merits and demerits against the framework established in Section 4. Clearly, a number of other specification methodologies exist that have not been included in this report. However, we consider the presented set to be representative.

A few observations regarding these ten methodologies are in order. The opinions presented are subjective, especially when considering issues such as complexity control and model continuity. Many of the deficiencies pointed out are not necessarily fundamental to the methodologies themselves, but, in our judgement, need to be explicitly addressed by them. Each methodology therefore has the potential to evolve to the point where they support all the dimensions indicated in the framework. No relative judgement has been offered between any two methodologies.

The surveys are organized as follows. We introduce each methodology with a brief background of its development. We next mention the system views supported, followed by a brief description of the steps in the methodology. We conclude each survey with a description of a few noteworthy features that we consider are the methodology's strengths and weaknesses. These surveys are by no means exhaustive, and the reader is encouraged to follow up the references provided for further detail. The main features of the methodologies are summarized in Tables 1-4.

In each table, the rows represent the methodology and the columns represent the attributes. Each table cell presents a summary of how the corresponding specification methodology supports the attribute represented by the corresponding column. In Tables 1, 2, and 3, we describe the support of the surveyed specification methodologies for the language, complexity control, and model continuity attributes respectively. The three attributes are then summarized in Table 4.

### 5.1. Ward and Mellor's Methodology (SDRTS or RTSA)

Ward and Mellor's methodology [SWC94], called *S*tructured *D*evelopment for *R*eal-*T*ime *S*ystems (SDRTS), was developed for the specification and design of real-time applications. Since the methodology is an extension of the Structured Analysis [DeM79] methodology for real-time systems, it is also called *R*eal-*T*ime *S*tructured *A*nalysis (RTSA). A similar methodology can be found in Hatley and Pirbhai's [HD88] approach.

There are two system-views adopted by RTSA: Data-Flow Diagrams (DFD) and Control-Flow Diagrams (CFD). DFD is used to model the activities in the system, whereas the CFD is used to express the sequencing of these activities. The CFD itself is defined in terms of a finite-state model such as FSM or decision table.

The methodology is based on structured analysis, one of the earlier approaches to express system requirements graphically, concisely, and minimally redundant manner. To develop a specification of the system, two models are created: the environment model and the system model. The environment model describes the operational context in which the system operates and the events to which the system reacts. The system model, which expresses the system's behavior, is then developed through successive refinements.

The language attribute is not supported well, since several aspects of reactive system behavior cannot be modeled conveniently. For example, there is no direct support for specifying timing constraints or handling exceptions elegantly.  A lack of any formal method support makes the methodology less useful in specification and design of critical systems. Lack of orthogonality makes it harder to conveniently represent and understand the behavior of complicated systems. The model-continuity attribute is also not well supported.

## 5.2. Jackson System Development (JSD)

Jackson System Development [Jac83] methodology, originally developed for program design [Jac75], is considered suitable for the design of information systems and real-time systems [Cam86]. In its current stage, the methodology covers specification, design and implementation stages.

The methodology is based on entity modeling. The system requirements are expressed as Jackson's diagram for entities. The diagram presents a time-ordered specification of the actions performed on or by an entity. A system specification diagram is also created, which is a network of processes that model the real world. A process communicates with others by transmitting data and state information. It is also possible to model time explicitly by introducing explicit delays.

The basic modeling philosophy is that the structure of a system to be designed can be determined from the structure and evolution of data it must manage. The methodology consists of two phases: specification and design. In the specification phase, the environment is described in terms of entities (real world objects the system needs to use) and actions (real world events that affect the entities). These actions are ordered by their expected sequence of occurrences and represented with Jackson diagrams. The actions and entities are then represented as a process network using system specification diagrams. The connection between these processes and the real world are defined. The creation of the initial process network can be seen as the end of specification phase.

During the design phase that follows the specification phase, the process network is successively elaborated by identifying further processes that are needed to execute the actions associated with the entities described in the Jackson Structure Diagram. The completed process network represents the final design which is then mapped to a set of hardware/software components.

The JSD methodology supports model continuity by carrying the specification through further well defined design steps. However, timing considerations come very late, almost after the design phase. The specification is implementation dependent, since it is closely tied to an implementation, thereby reducing designer freedom. The methodology lacks support for expressing several reactive-system characteristics such as exception handling. Neither does it support any formal analysis or well defined execution semantics. The main strength of the methodology lies in its complexity-control attribute, especially regarding its developmental guidance and representational elegance.

## 5.3. Software Requirements Engineering Methodology (SREM)

Software Requirements Engineering Methodology (SREM) [AAH82] was developed for creation, checking and validation of specifications of real-time and distributed applications for data processing.

The SREM method is useful for specification making use of structured finite-state automata called requirement nets (r-nets). R-nets express the evolution of outputs and final state starting from inputs and current state. Both inputs and outputs are structured as sets of messages, communicated by an interface connected to the environment.

To develop a specification, the interface between the system and the environment and the data-processing requirements are specified. The initial description is produced using r-nets. Functional details, timing and performance constraints are then added. Next, validation and coherency checks are performed on the specification. A final feasibility study is conducted to guarantee that the specification will result in a feasible solution.

The behavior of a reactive system is well-represented by this methodology. The addition of performance specifications and timing constraints are also beneficial. However, hierarchical decomposition of the specification is not supported. As a result, the task of specification requires too much detail. The attribute of model-continuity is supported since both the specification and implementation can be considered in an integrated manner for testing, feasibility analysis, supporting fault-tolerance etc. Both implementation assistance and implementation independence are supported by the methodology since it allows one to study multiple implementation schemes for a given specification. However, the methodology ties the specification too closely to the possible implementation in terms of its structure. As a result, the support for model continuity can be considered to be limited, since the number of implementations considered is limited by the structure of the specification.

## 5.4. Object Oriented Analysis (OOA)

OOA stands for Object Oriented Analysis, and is based on the object-oriented paradigm of modeling. The world is modeled in terms of classes and objects that are suitable to express the problem domain. Different schemes have been suggested for OOA [CY90, MP92, SM88], they differ mostly in terms of notations and heuristics.

OOA can be seen as an extensions of data or information modeling approaches. The latter approaches focus solely on data. In addition to modeling data, OOA also concerns data transformations.

The major steps in OOA consist of identifying objects, their attributes, and the structure of their interrelationships. The entire specification is developed as a hierarchy of modules, where each module is successively refined both horizontally and vertically into further modules. The vertical refinement adds further properties to a module, whereas the horizontal refinement identifies a set of loosely-coupled, strongly-cohesive interacting sub-modules that define the behavior of the original module. The implementation of these modules is, however, not considered at the specification stage.

The OOA main strength lies in the complexity-control attribute. To exploit the strength of OOA, however, it should be combined with languages that support expression of reactive system characteristics and have both formal and operational semantics. For application to the domain of reactive systems, we find approaches that combine languages suitable for expressing reactive systems with object-oriented design principles. Such an example can be found in [Col92].

## 5.5. Specification and Description Language (SDL)

Specification and Description Language is a design methodology that has been standardized by CCITT [CCI88, FO92], and is used for specifying and describing many types of systems. SDL is standardized, semiformal, and can be used both as a graphical or a textual language. SDL is primarily used for telecommunication systems [SSR89].

SDL provides three views of a system: structural, behavioral and data. It is the behavioral view of the system that is used to specify the system's reactive nature. The system is modelled as a number of interconnected abstract machines. The machines communicate asynchronously.

The specification models are developed for the three system-views as follows. The structural model is generated hierarchically, starting from a *block* that is recursively decomposed into a number of *blocks* connected together by *channels*. At the topmost level, the *system block* has *channels* that allow interfacing with the system's environment. The behavioral model is a set of *processes* that are extensions of deterministic finite-state machines. The interaction between these *processes* is done via *signals*. These *processes* can be dynamically created and collectively represent the system behavior. Temporal ordering between the *signals* used in inter-*process* communication is specified using message-sequence charts, which are useful for debugging the specification. Finally, the data is modeled as an Abstract Data Type, where one describes the available data-operations and data-values but not how they are implemented.

SDL supports the perfect-synchrony hypothesis and has an associated formal semantics. However, it does not support all the reactive system characteristics. For example, exception handling is not directly supported since the inputs are typically consumed by a process only when the receiving process is ready to process the input. Thus, if an exception condition is communicated as an input to the process, it may not be acted upon immediately. Rather, the exception will be handled when the receiving process is ready to process the arriving exception. The complexity-control attribute is well supported. Model-continuity is not well supported in the methodology.

### 5.6. Embedded Computer Systems (ECS)

The Embedded Computer Systems (ECS) methodology [HLN88, LB91] is based on the three views of system modeling: activities, control, and implementation. Express-VHDL [ILO93] is used as a computer-aided design tool for this methodology.

ECS supports both behavioral and functional decomposition of the system's specification. The system behavior is expressed using the visual formalism of Statecharts [Har87], an extension of FSM that significantly reduces the representational state-space explosion problem encountered by ordinary FSMs. This reduction is achieved due to the conceptual models of concurrency, hierarchy, and complex transitions supported by Statecharts. A history operator, useful for expressing interrupt-handling, is also provided to significantly reduce the representational complexity.

The system is functionally decomposed using activity charts, and is viewed as a collection of interconnected functions (activities) organized in a hierarchy. The activity charts visually depict the flow of information in the system, with the control of flow being represented by the associated Statecharts model.

To develop the specification, the methodology recommends a top-down and iterative analysis that gradually expresses all the requirements of the system. Conceptually, a system is decomposed into a number of subsystems, each carrying out a functionality, and a controller that coordinates the activities between these subsystems. The behavior of each system is represented by a Statecharts model. Each state in the Statecharts model can be refined further into AND and OR states. The default-entry states, needed synchronizations, associated timing constraints, etc. are specified next.

Both the language and complexity-control attributes are well supported by the ECS methodology. ECS is based on the language of Statecharts which is very suitable for expressing reactive system requirements in an implementation-independent manner. Support for the executability and some static and dynamic analysis is provided at each stage of development. However, model continuity, especially along the model-integration dimensions, is not supported.

### 5.7. Vienna Development Method (VDM)

VDM (Vienna Development Method) [LB91] is an abstract model-oriented formal specification and design method based on discrete mathematics [HI88]. The formal specification language of VDM is known as META-IV [Ber91].

The specification is written as a specification of an abstract data type. The abstract data type is defined by a class of objects and a set of operations to act upon these objects while preserving their essential properties. A program is itself specified as an abstract data type, defined by a collection of variables and the operations allowed on these variables. The variables make the notion of state explicit, as opposed to property-based methods.

The specification development process is closely tied to the design process. The steps of the specification methodology is as follows. A formal specification is developed using the META-IV language. Once the specification is checked via formal analysis and found to be consistent, the specification is refined and further decomposed into what is called a *realization*. The realization is checked against the original specification for conformance. The specification is iteratively and step-wise refined until the realization is effectively a complete implementation.

This method supports model continuity very well. The main drawback, in our opinion, is that it is relatively difficult to understand and conceptualize a reactive system from a VDM specification. Neither is the specification executable. The structure of the specification has a large impact on the implementation, making the methodology weak on the implementation independence since the implementation is largely determined by how the specification is decomposed. It is not clear how one can incorporate independently developed implementations and check for their conformance against the specification.

## 5.8. Language Of Temporal Ordering Specification (LOTOS)

LOTOS (language Of Temporal Ordering Specification) is an internationally standardized formal description technique, originally developed for the formal specification of OSI (Open Systems International) protocols and services.

The specification is based on two approaches: process algebras and abstract data types. The process-algebra approach is concerned with the description of process behaviors and interactions and is based on Milner's Calculus of Communicating Systems [Mil80] and Hoare's work on Communicating Sequential Processes [Hoa85]. The abstract data type approach is based on ACT-ONE [EM85], which deals with the description of data structures and value expressions. The resulting specifications are unambiguous, precise, and implementation independent.

A system is specified by defining the temporal relationships among the interactions that make up its externally observable behavior. These interactions are between processes, which act as black-boxes. A black-box is an entity capable of performing both internal actions and external actions. The internal actions are invisible beyond its boundaries whereas the external actions are observable by an *observer* process. Interactions between these processes is achieved using *events*. The processes are specified using process algebra approach, which allows the description of behavior as a composition of basic behaviors using a rich set of combining rules.

Being property-oriented, LOTOS supports implementation independence, as it specifies behavior of the system that can be observed only externally. The structure of the implementation is not restricted by the specification. However, being property-oriented also makes it hard to conceptualize the internal states of reactive system. Further, it also becomes hard to generate an implementation from the specification. The concept of time is also not directly supported.

## 5.9. Electronic Systems Design Methodology (MCSE)

MCSE (Methodologie de Conception des Systemes Electroniques) is a methodology for the specification, design, and implementation of industrial computing systems. The methodology is characterized by its top-down approach to the design of real-time systems, and is structured into several steps from system specification to system implementation.

Three kinds of specifications are produced during the specification process. First, functional specifications include a list of system functions and a description of the behavior of the system's environment. Second, operational specifications concern the performance and other implementation details that are to be used in the system. Third and finally, technological specifications include specifications of various implementation constraints such as geographic distribution limitations,

**Table 1: Support for language attribute by specification-modeling methodologies**

| Specification Modeling Methodologies | Support for language attribute [Section 4.1] | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Available conceptual models[a, b] [Section 4.1.1] | | | | | | Analysis techniques [Section 4.1.2] | |
| | System views | Specification style | Timing constraints | Modeling time | Exception handling | Environment characterization | Formal analysis | Model executability |
| **SDRTS [Section 5.1]** | activity+behavior | model | indirect | limited | limited | model | limited | limited |
| **JSD [Section 5.2]** | entity | model | direct | supported | limited | model | limited | limited |
| **SREM [Section 5.3]** | behavior | model | direct | supported | limited | property | semiformal | supported |
| **OOA [Section 5.4]** | entity+behavior | model | indirect | limited | limited | limited | limited | limited |
| **SDL [Section 5.5]** | entity+behavior | model | indirect | supported | limited | limited | supported | supported |
| **ECS [Section 5.6]** | activity+behavior | model | indirect | supported | supported | model | supported | supported |
| **VDM [Section 5.7]** | entity | model | indirect | limited | supported | limited | supported | supported |
| **LOTOS [Section 5.8]** | entity+behavior | property | indirect | limited | supported | property | formal | limited |
| **MCSE [Section 5.9]** | activity+behavior | model, property | direct | supported | limited | model | semiformal | limited |
| **ISPME[Section 5.10]** | activity+behavior | model | indirect | supported | supported | model | formal | supported |

**a. All methodologies surveyed here support concurrency.**
**b. All methodologies surveyed here show limited support for nonfunctional characteristics**

Table 2: Support for complexity-control attribute by specification-modeling methodologies

| Specification Modeling Methodologies | Support for complexity control [ Section 4.2 ] | | | | | |
|---|---|---|---|---|---|---|
| | Representational complexity [ Section 4.2.1 ] | | | Developmental complexity [ Section 4.2.2 ] | | |
| | Hierarchy | Orthogonality | Representation scheme | Nondeterminism | Perfect-synchrony Assumption | Developmental guidance |
| **SDRTS [Section 5.1]** | supported | limited | graphical | limited | asynchronous | top down |
| **JSD [Section 5.2]** | supported | supported | graphical | limited | asynchronous | top down |
| **SREM [Section 5.3]** | limited | limited | graphical | limited | asynchronous | bottom up |
| **OOA [Section 5.4]** | supported | supported | textual | limited | asynchronous | top down |
| **SDL [Section 5.5]** | supported | supported | graphical | supported | synchronous | top down |
| **ECS [Section 5.6]** | supported | supported | graphical | supported | synchronous | top down |
| **VDM [Section 5.7]** | supported | supported | textual | supported | synchronous | top down |
| **LOTOS [Section 5.8]** | supported | supported | textual | supported | synchronous | top down |
| **MCSE [Section 5.9]** | supported | supported | graphical | limited | synchronous | top down |
| **ISPME [Section 5.10]** | supported | supported | graphical | supported | synchronous + asyn-chronous | top down bottom up |

**Table 3: Support for model-continuity attribute by specification-modeling methodologies**

| Specification Modeling Methodologies | Support for model-continuity attribute [ Section 4.3] | | | | |
|---|---|---|---|---|---|
| | Model integration [Section 4.3.1] | | | Implementation assistance [Section 4.3.2] | Implementation independence [Section 4.3.3] |
| | Conformance | Interaction | Complexity | | |
| **SDRTS [Section 5.1]** | limited | vertical | vertical | limited | limited |
| **JSD [Section 5.2]** | limited | vertical | vertical | limited | supported |
| **SREM [Section 5.3]** | limited | vertical | vertical | supported | limited |
| **OOA [Section 5.4]** | limited | limited | limited | limited | supported |
| **SDL [Section 5.5]** | limited | limited | limited | supported | supported |
| **ECS [Section 5.6]** | limited | limited | limited | supported | supported |
| **VDM [Section 5.7]** | supported | vertical | vertical | supported | limited |
| **LOTOS [Section 5.8]** | limited | vertical | limited | limited | supported |
| **MCSE [Section 5.9]** | limited | limited | supported | limited | supported |
| **ISPME [Section 5.10]** | supported | supported | supported | supported | supported |

**Tabe 4: Summarizing attributes of reactive-system specification-modeling methodologies**

| **Specification Modeling Methodologies** | **Language** **[ Section 4.1]** | **Complexity control** **[Section 4.2]** | **Model continuity** **[Section 4.3]** |
|---|---|---|---|
| **SDRTS [Section 5.1]** | limited | limited | limited |
| **JSD [Section 5.2]** | limited | supported | limited |
| **SREM [Section 5.3]** | supported | limited | limited |
| **OOA [Section 5.4]** | limited | supported | limited |
| **SDL [Section 5.5]** | supported | supported | limited |
| **ECS [Section 5.6]** | supported | supported | limited |
| **VDM [Section 5.7]** | supported | limited | limited |
| **LOTOS [Section 5.8]** | supported | limited | limited |
| **MCSE [Section 5.9]** | supported | supported | limited |
| **ISPME [Section 5.10]** | supported | supported | supported |

interface characteristics etc.

There are two main parts in the specification process: environment modeling and system modeling. In the environment-modeling part, the environment is first analyzed to identify the entities that are relevant to the system. Next, a model is created representing the identified entities and their interactions, thus providing a functional description of the environment. In the system-modeling part, the system under design is first delimited in terms of its inputs and outputs. Next, a functional specification of the system is developed, which describes the functions to be carried out by the system on its environment. This functional specification is developed by characterizing the system in terms of system inputs and outputs, system entities, or system activities.

The MCSE approach provides a well defined methodology for developing a specification. Another strength of this approach lies in the fact that it allows a multitude of system views and modeling approaches for system specification. However, there is a lack of support for formal techniques. While several modeling techniques and system views are supported, a coherent integration of these diverse approaches is not supported.

### 5.10. Integrated Specification and Performance Modeling Environment (ISPME)

The Integrated Specification and Performance Modeling Environment [Sar95] is an evolving specification modeling methodology that supports a strong interaction between the specification phase of a design process with design and implementation phases. As a result of this interaction, there is an increased and better communication of design intent among these phases.

ISPME is based upon the language of Statecharts, similar to ECS modeling methodology. As a result, it supports the behavioral view of the system. However, ISPME also supports complementary modeling, where some aspects of the system are modeled using Statecharts, while the remaining aspects are represented as a performance model. The performance model is developed using ADEPT [AWJ92, AWW90], which is based on an extension of Petri-nets [Pet81]. Since a performance-model can coexist with the Statecharts specification, ISPME supports the activity view for a system.

The methodology supports the development of a complete implementation from the specification in a incremental and iteratively refined manner. Each increment represents a proposed implementation of a component of the Statecharts model. The performance model of the proposed implementation is verified against its Statecharts counterpart. At each iteration, both the Statecharts and the ADEPT models can be refined, since it is possible that one may encounter inconsistencies between the specification and the implementation. As a result of this integrated-modeling approach, the methodology extends the specification phase to later design stages. Such extension improves communication of design intent between various phases of design.

The ISPME approach supports all three attributes of a specification modeling methodology for reactive systems. First, by adopting the Statecharts language, it is able to support both the language and the complexity attributes. Second, due to its support for integrated modeling with ADEPT, the model interaction component of the model-continuity attribute is well supported. Since the performance model developed is independent in terms of structure from the Statecharts model, implementation independence is supported. Further, implementation assistance is supported since the Statecharts model itself may be synthesized into an implementation.

## 6. Summary

We summarize our conclusions drawn from studying Tables 1-4. For each table, we looked down each column representing a feature and tried to identify if the feature was well supported by the methodologies. Wherever possible, we identify correspondences between features and attempted to present general observations and recommendations.

### 6.1. Language (Table 1)

There are a number of languages available today that are suitable for expressing reactive system characteristics [BD87, Ber91, FJ91, GVN93, Har92, Mar91, ZS86]. However, several of the methodologies we examined did not incorporate such languages. Another interesting feature we observed is that several methodologies support more than one of the following system views: entity, activity, and behavior. For example, ECS, LOTOS, SDL support two of the possible three system views.

Most of the studied methodologies supported explicit models. We do not find this surprising, as it is typically easier, in our view, to conceptualize reactive system behavior in terms of internal states. However, external or property oriented models, such as those created in LOTOS, makes it easier to characterize the system in an implementation-independent way.

Direct support for expressing timing constraints or a model of wall-clock time was not available in several cases. Several methodologies did not directly support an explicit model of time and supported temporal ordering between events instead. Another specification requirement of reactive systems, namely, exception handling, was often not well supported by the specification languages.

Almost none of the approaches allow the characterization of the reactive system's environment. We find that most of the later and developing methodologies adopted languages that allow formal specifications. However, operational semantics is not supported in some cases, especially in the case of LOTOS, where there is no explicit model.

### 6.2. Complexity control (Table 2)

Representational complexity is generally well supported by the methodologies. In a few cases, neither hierarchy nor orthogonality were supported to the fullest extent. Most methodologies chose visual schemes for representing the specifications, and in some cases such as ECS, associated semantics with the graphical objects that represented the specifications.

Developmental complexity is also reasonably well supported. However, nondeterminism and perfect-synchrony were not well supported in some cases. In some cases, the assumption of perfect synchrony was not made clear. Most formal specification approaches adopted the concept of perfect synchrony.

For guidance in the development of specifications, most methodologies supported the top-down approach. Some allowed an iterative approach, and few supported a bottom up approach. An incremental refinement of the specification was supported in some cases.

### 6.3. Model continuity (Table 3)

In our opinion, this is the weakest dimension for most methodologies. Except for ISPME[Sar95], none support the integrated modeling dimensions, especially when one considered models that were dissimilar. Implementation-independence of the specification was generally supported. Implementation assistance was not limited in cases where the methodology was non-formal or was based on an external model.

Support for conformance checking, model interaction, or model complexity was limited. We also observed that in some cases the implementations were derived from the structure of the specification. In approaches that support external models, implementations were not usually derived directly from the specifications as the latter had no internal structure. Many of the approaches were also more applicable to software, rather than both hardware and software.

### 6.4. Overall (Table 4)

Overall, we feel that the languages and complexity control dimensions are reasonably well supported today, even though there is room for improvement. Except for ISPME, the model-continuity dimension is not supported well, especially when one considers dissimilar modeling domains.

## 7. Recommendations

Based on the observations above, we make several recommendations regarding what we feel most specification methodologies need to specifically address today. In an overall sense, we think approaches that support all the views, express all reactive system characteristics using a visual formalism, which has an associated formal and operational semantics, would be ideal. Further, this methodology should allow integrated modeling with other models and strongly support implementation independence. Automated generation of possible implementations should also be highly desirable.

We categorize our recommendations according to the dimensions presented earlier.

### 7.1. Specification languages

In addition to what is supported by the state of the art, specification language(s) adopted by the methodology should incorporate the following guidelines:

#### 7.1.1. Conceptual models

- *should support entity, behavior and activity views at the same time*

  All three views of a system: entity, behavior and activity, should be supported, since all three views are complementary to each other. Supporting more than one view would possibly entail either supporting more than one specification language, or choosing a language that incorporates more than one view.
- *should support both model-oriented and property-oriented specifications*

  While it is typically easier to understand and relatively easier to generate implementations from an model-oriented specification, property-oriented specifications are less implementation dependent and offer the implementor to be more creative in generating solutions. We therefore recommend supporting both model-oriented and property-oriented specifications in an integrated manner.
- *should specify timing constraints explicitly*

  To improve the understandability and to reduce the chances or erroneous specifications, it is preferable to be able to specify timing constraints such as maximum or minimum execution times, data transfer rates, or inter-event constraints directly. We recommend a simple and flexible specification scheme be chosen.
- *should model passage of time explicitly*

  While temporal ordering may often be sufficient for the purpose of proving properties of the system, it may be easier to understand and express required behavior in terms of explicit time. Therefore, a methodology should model time explicitly.

- *should support exception handling*

  Exception handling mechanisms should also be provided in the specification language.
- *should characterize the environment explicitly*

  Some allow the representation of the environment as a separate model. We believe that the environment should be characterized as a property-oriented specification, where the environmental characteristics are specified as properties and hints.

### 7.1.2. Analysis techniques

- *should support formal semantics*

  Supporting a specification language with formal semantics will result in unambiguous and rigorously verifiable specifications. Formal semantics will also support automated techniques for checking inconsistencies in the specification and the synthesis and verification of implementations. There has been an increased awareness of the importance of incorporating formal specifications into software development [FKV94]. This trend of awareness should continue to evolve and encompass both software and hardware development.
- *should support operational semantics*

  By supporting operational semantics, the methodology will enable the generation of early prototypes of the system under design, which would allow early validation of user requirements without necessarily committing to design decisions.

## 7.2. Complexity control

### 7.2.1. Representational complexity

- *should support visual formalisms*

  As pointed out in [Har92, Har88], visual formalisms assist in representing conceptual constructs that specifiers and designers have in mind during the design stages of complex reactive systems.

### 7.2.2. Developmental complexity

- *should support bottom up development*

  We found that the existing specification methodologies rarely allow the composition of specifications in a bottom up manner. We believe that supporting a bottom up approach will assist in specification reuse, where one may potentially create new specifications from a collection of preexisting specifications.
- *should make explicit assumption of perfect-synchrony hypothesis*

  The assumption of perfect-synchrony has been implicitly adopted by some methodologies. However, this assumption is not always valid, and can cause design scenarios to remain unexplored [Sar95]. The methodology should allow one to explore such cases by explicitly choosing or rejecting the hypothesis.

## 7.3. Model continuity

### 7.3.1. Integrated modeling

- *should support conformance checking between models*

  By supporting conformance between a specification model and models developed during later stages, divergence of these models from specified behavior can be detected.
- *should allow integrated simulation and formal analysis*

  Currently most methodologies simulate and analyze the specification model in isolation from other models. Simulating and analyzing these models in an integrated manner will result in a synergistic modeling environment. This synergy will be due to an increased interaction between the two models that will result in the exchange of information that may not be available to the models individually. The specification methodologies should exploit simulation languages such as VHDL [IEE88], which are able to represent digital systems at various levels of abstractions and modeling domains.
- *should allow integrated representation and refinement across different design stages*

  To facilitate a stepwise, iteratively refined design process across all design stages, an integrated representation scheme is needed where one can represent the specification, design, and implementation steps at various stages of development.

### 7.3.2. Implementation independence

- *should extend to include both software and hardware*

  Most specifications are either biased towards hardware or towards software. An integrated representation scheme is required.
- *should allow incorporation of implementation details without sacrificing implementation independence*

  While the specifications are themselves implementation independent, some design processes tend to modify the original specifications by annotating it with implementation dependent information. To compare different implementations for the same specification, one should be able to devise a scheme that allows the inclusion of implementation dependent details in a generic manner. The scheme should support the inclusion of implementation information from different alternatives. For the example of such a technique, see [Sar95].

### 7.3.3. Implementation assistance

- *should provide automated support for generating implementations*

  To extend the useful life of the specification, the designers should be able to generate, automatically if possible, reasonably good implementations from the specification.

## 8. Summary and Conclusions

We presented a framework to evaluate specification modeling methodologies for reactive systems. The framework has been synthesized by combining the characteristics of reactive systems and the requirements of a specification modeling methodology. Using this framework, we examine ten methodologies. Such an evaluation technique proved useful, since we were able to study the individual strengths and weaknesses of each methodology in a common-evaluation framework. Further, we were able to make recommendations based on our survey that we believe should be incorporated by both existing and emerging methodologies.

## 9. Bibliography

AAH82 M. W. Alford, J.P. Ansart, G. Hommel, L. Lamport, B. Liskov, G. P. Mullery, F. B. Schneider. *Distributed Systems. Methods and Tools for Specification*. Lecture Notes in Computer Science, Springer-Verlag 1982.

AWJ92 Aylor, J. H. and Waxman, R. and Johnson, B.W. and R.D.Williams,. The Integration of Performance and Functional Modeling in VHDL. In *Performance and Fault Modeling with VHDL.* Schoen, J. M., Prentice Hall, Englewood Cliffs, NJ 07632, 1992, pages 22-145.

AWW90 Aylor, J. H. and Williams, R. D. and Waxman, R. and Johnson, B. W. and Blackburn, R. L. A Fundamental Approach to Uninterpreted/Interpreted Modeling of Digital Systems in a Common Simulation Environment. *UVa Technical Report TR # 900724.0,* University of Virginia, Charlottesville, USA, July 24, 1990.

BD87 S. Budkowski and P. Dembinski. *An Introduction to Estelle: A Specification Language for Distributed Systems*. Computer Networks and ISDN Systems. North-Holland, Vol. 14, 1987.

Ber91 G. Berry. *A Hardware implementation of pure Esterel*. Digital Equipment Pars Research Laboratory, July 1991.

BCD88 Brayton, R. K. and Camposano, R. and De Micheli, G. and Otten, R. H. J. M. and van Eijndhoven, J. T. J. The Yorktown Silicon Compiler System. In *Silicon Compilation*. Gajski, D.D., Addison-Wesley, 1988.

BHS91 F. Belina, D. Hogrefe, and A. Sarma. *SDL with Applications from Protocol Specifications*. Prentice Hall, 1991.

BJ78 D. Bjorner, C.B. Jones. *The Vienna Development Method: The Meta-Language.* Lecture Notes in Computer Science. No. 61. Springer-Verlag. 1978.

BT85 Blackburn, R. L. and Thomas, D. E. Linking the Behavioral and Structural Domains of Representation in a Synthesis System. *DAC 85*:374-380.

Cal93 J.P. Calvez. *Embedded Real-Time Systems: A Specification and Design Methodology.* Wiley Series in Software Engineering Practice. 1993.

Cam86 J.R. Cameron. “An overview of JSD”. *IEEE Transactions on Software Engineering*. Vol SE-12. No. 2. February 1986.

CCI88 CCITT. *Recommendation Z.100: Specification and Description language (SDL*). Volume X, Fascicle X.1, Blue Book, October 1988

Col92 D. Coleman. “Introducing Objectcharts or How to Use Statecharts in Object-Oriented Design". *IEEE Transactions on Software Engineering*. Vol. 18, No. 1. Jan 1992

CPT89 Chu, C. M. and Potkonjak, M. and Thaler, M. and Rabaey, J. HYPER: An Interactive Synthesis Environment for High Performance Real Time Applications. *Proceeding of the International Conference on Computer Design*, pages 432-435, 1989.

CR85 Camposano, R. and Rosenstiel, W. A Design Environment for the Synthesis of Integrated Circuits. *11th EUROMICRO Symposium on Microprocessing and Microprogramming* 1985.

CY90 P. Coad, E. Yourdon. *Object-Oriented Analysis*. Prentice-Hall, 1990.

Dav88 A. Davis. " A Comparison of Techniques for the Specification of External System Behavior". *Communications of the ACM*. Vol 31, No. 9. 1988.

DeM79 T. DeMarco. *Structured Analysis and System Specification*. Yourdon Computing Series, Yourdon Press, Prentice Hall, 1979.

DG89 Dutt, N. D. and Gajski, D. D. Designer Controlled Behavioral Synthesis. Proceedings of the 26th Design Automation Conference, pages 754-757, 1989.

DH89 D. Drusinsky and D. Harel. *Using Statecharts for hardware description and synthesis*. In IEEE Transactions on Computer-Aided Design, 1989.

Dij75 E.W. Dijkstra. "Guarded commands, nondeterminacy, and formal derivation of programs". *Communications of the ACM,* Vol 18, No. 8. 1975.

EM85 H. Ehrig, B. Mahr. *Fundamentals of Algebraic Specification - 1*. EATCS Monographs on Theoretical Computer Science 6. Springer-Verlag. 1985.

FJ91 L.M.G. Feijs and H.B.M. Jonkers. *Specification and Design with COLD-K*. LNCS 490, pp. 277-301.

FKV94 M.D. Fraser, K. Kumar, V.K. Vaishnavi. "Strategies for Incorporating Formal Specifications in Software Development". *Communications of the ACM*. Vol. 37, No. 10. Oct 1994 p 74-86.

FO92 O. Færgemand, A. Olsen. *New Features in SDL-92*. Tutorial, Telecommunications Research Laboratory, TFL, Denmark. 1992.

GVN93 D.D. Gajski, F. Vahid, and S. Narayan. *A system-design methodology: Executable-specification refinement*. In Proceedings of the European Conference on Design Automation (EDAC), 1994.

Hal93 N. Halbwachs. *Synchronous Programming of reactive Systems*. Kluwer Academic Publishers, 1993.

Har92 D. Harel. "Biting the Silver Bullet: Toward a Brighter Future for System Development”. *IEEE Computer*. Vol. 25, No. 1. Jan 1992, pages 8-24.

Har88 D. Harel. “On Visual Formalisms”. *Communications of the ACM*. Vol. 31, No. 5. 1988 p 514-530.

Har87 D. Harel. “Statecharts: A Visual Formalism For Complex Systems”. *Science of Computer Programming, Vol 8*. 1987, pages 231-274.

HD88 D.C. Hu and G. DeMicheli. *HardwareC - a language for hardware design*. Stanford University, Technical Report CSL-TR-90-419, 1988.

HLN88 D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, M. Trakhtenbrot. “Statemate: A working environment for the development of complex reactive systems”. *Proceedings of 10th International Conference on Software Engineering*. Singapore, 11-15 April 1988, p 122-129.

HI88 S. Hekmatpur, D. Ince. *Software Prototyping, Formal Methods, and VDM*. International Computer Science Series, Addison-Wesley Publishing Company, 1988.

Hoa85 C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, London. 1985.

HP87 D. J. Hatley, I. A. Pirbhai. *Strategies for Real-time System Specification*. Dorset House Publishing, New York, 1987.

IEE88 IEEE. *IEEE Standard VHDL Language Reference Manual*. IEEE Inc., NY, 1988.

ILO93 i-Logix Inc. *ExpressVHDL Documentation, Version 3.0*. 1992.

ISO89 ISO/IS 8807. *Information Processing Systems - Open Systems Interconnection: LOTOS - A Formal Description Technique*. 1989.

Jac83 M.A. Jackson. *System Development*. Prentice-Hall, 1983.

Jac75 M.A. Jackson. *Principles of Program Design*. Academic Press, 1975.

LB91 Lor, K. E. and Berry, D. M. Automatic Synthesis of SARA Design Models from System Requirements. *IEEE Transactions on Software Engineering* 17(12):1229-1240 December 1991.

LW88 J. Z. Lavi, M. Winokur. “Embedded computer systems: requirements analysis and specification: An industrial course”. *Proceedings of SEI Conference, Virginia April 1988*. Lecture Notes in Computer Science: Software Engineering Education, No. 327. Ed. G. A. Ford, Springer-Verlag p 81-105.

Mar91 F. Maraninchi. *Argos: A graphical synchronous language for the description of reactive systems*. Report RT-C29, Univeriste Joseph Fourier, 1991.

Mil80 R. Milner. *A Calculus of Communicating Systems*. Lecture Notes in Computer Science 92. Springer-Verlag. 1980.

MP92 D.E. Monarchi, G.I. Puhr. “A Research Typology for Object-Oriented Ananlysis and Design". *Communications of the ACM.* Vol. 35, No. 9. Sep 1992.

| | |
|---|---|
| MP91 | Z. Manna, A. Pnueli. *The temporal logic of reactive and concurrent systems: specification.* Berlin Heidelberg New York: Springer. 1991. |
| Pet81 | J.L. Peterson. *Petri Net Theory and the Modeling of Systems.* Engelwood Cliffs, NJ, Prentice Hall, Inc., New York. 1981. |
| Pnu86 | A. Pnueli. "Application of temporal logic to the specification and verification of reactive systems." *Current Trends in Concurrency. Lecture Notes in Computer Science. Eds: de Bakker et al.* Vol. 224, No. 9. Sep 1992. |
| Sar95 | A. Sarkar. *An Integrated Specification and Performance Modeling Approach for Digital System Design*. Ph.D. Thesis. University of Virginia. Charlottesville, U.S.A. 1995. |
| SM88 | S. Schlaer, S.J. Mellor. *Object-Oriented Systems Analysis*. Yourdon Press, 1988. |
| Spi88 | J.M. Spivey. *Understanding Z: A specification Language and its Formal Semantics.* Cambridge University Press. 1988. |
| SSR89 | R. Saracco, J. Smith, R. Reed. *Telecommunication Systems Engineering using SDL*. Elsevier Science Publishers. 1989. |
| SST90 | E. Sternheim, R. Singh, and Y. Trivedi. *Hardware Modeling with Verilog HDL*. Automata Publishing Company, Cupertino, CA, 1990. |
| SWC94 | A. Sarkar, R. Waxman, and J.P. Cohoon. *System Design Utilizing Integrated Specification and Performance Models*. Proceedings, VHDL International Users Forum, Oakland, California, May 1-4, 1994, pp 90-100. |
| TM91 | D.E. Thomas and P. Moorby. *The Verilog Hardware Description Language*. Kluwer Academic Publishers, 1991. |
| Win90 | J.M. Wing. "A specifier's introduction to formal methods". *IEEE Computer, Vol 23, No. 9.* 1990. pp 8-24. |
| WWD92 | Woo, N. and Wolf, W. and Dunlop, A. *Compilation of a single specification into hardware and software*. AT&T Bell Labs, 1992. |
| WM85 | P. T. Ward, S. J. Mellor. *Structured Development for Real-time Systems, Vols 1 & 2*. Yourdon Computing Series, Yourdon Press, Prentice Hall 1985. |
| ZS86 | P. Zave and W. Shell. *Salient features of an executable specification language and its environment*. IEEE Transactions on Software Engineering 12(2):312-325 Feb, 1986. |